Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA

Sistema de Clasificación Paralelo
basado en un
Ensamble de tipo Mezcla de Expertos

Tesis que presenta:

M. en C. Benjamin Moreno Montiel

para obtener el grado de:

DOCTOR EN CIENCIAS
(Ciencias y Tecnologías de la Información)

Asesor de la Tesis

Dr. René Mac Kinney Romero

Jurado calificador:

Presidente: Dr. Eduardo Morales Manzanares: _____

Secretario: Dr. René Mac Kinney Romero: _____

Vocal: Dra. Graciela Román Alonso: _____

Vocal: Dr. Eduardo Rodríguez Flores: _____

Vocal: Dr. Hugo Jair Escalante Balderas: _____

Ciudad de México

Agosto 2017

Resumen

Resolver el problema de clasificación sobre grandes cantidades de datos representa un gran desafío en Minería de Datos, que solamente un número reducido de algoritmos de clasificación pueden manejar. En este trabajo se presenta un Sistema de clasificación paralelo basado en un ensamble de tipo mezcla de expertos (*Parallel Classification System based on an Ensemble of Mixture of Experts - PCEM*), que da solución a este desafío. El *PCEM* que se propone utiliza una arquitectura de cómputo paralelo del tipo MIMD (Multiple Instruction and Multiple Data Stream), basado en un conjunto de procesos que se comunican a través de mensajes. En el *PCEM* se emplean un conjunto de esquemas paralelos de clasificadores tradicionales, de allí el nombre de Ensamble de Tipo Mezcla de Expertos. También se utiliza un esquema paralelo de un Algoritmo Genético, lo cual representa un aporte con respecto a los métodos existentes que permiten asignar un peso diferente a cada clasificador. Mediante esta asignación de pesos, se aplica un criterio de votación ponderada para combinar las clasificaciones individuales del ensamble. El *PCEM* es un algoritmo novedoso, ya que mediante éste podemos realizar una clasificación sobre grandes cantidades de datos, obteniendo bajos tiempos de ejecución y altos niveles de rendimiento. Mediante una serie de pruebas se comprobaron los resultados del *PCEM* contra un conjunto de clasificadores tradicionales y paralelos, obteniéndose resultados favorables para el *PCEM*, en tiempos de ejecución e índices de medidas de rendimiento.

Palabras Claves: Clasificadores basados en Ensambles, Algoritmos Genéticos, Aprendizaje Maquinal, Clasificación, Computación Paralela, Mezcla de Expertos, Minería de Datos.

Abstract

The classification of large amounts of data is a challenging task in machine learning, which only some few classifiers can handle. In this paper we propose a Parallel Classifier based on Mixture of Experts (PCME)to handle this challenging task. The PCME is a novel algorithm since it allows us to classify large amounts of data with low execution times and performance measures (accuracy, lift, precision and recall) better than other classifiers. In previous work we saw that heterogeneous ensemble models were better but at a explores steep time cost. This work using parallel programming to overcome the time cost problem. We used the MIMD (Multiple Instruction and Multiple Data Stream) architecture, that uses a set of process that communicate via messages. PCME is implemented using parallel schemes of traditional classifiers, for the mixture of experts, and using also a parallel version of a Genetic Algorithm to implement a weighted voting criterion. We performed a series of tests with a set of databases that place the PCME as a very competitive classifier.

Keywords: Data Mining and Classification and Classifiers based on Ensemble and Machine Learning and Parallel Computing

Agradecimientos

Un agradecimiento muy especial a mi Asesor el Dr. René Mac Kinney Romero, por todos los conocimientos que comparte conmigo y por la dedicación y el gran apoyo brindado en la elaboración de la presente Tesis.

Al Consejo Nacional de Ciencia y Tecnología, CONACyT, por la beca otorgada para estudios de posgrado.

A la Universidad Autónoma Metropolitana - Unidad Iztapalapa y a todos los Académicos que han sido parte importante de mi formación formación académica y de investigación.

A los revisores de esta Tesis y sinodales en el examen de grado: Dr. Eduardo Morales Manzanares, Dra. Graciela Román Alonso, Dr. Eduardo Rodríguez Flores y Dr. Hugo Jair Escalante Balderas, quienes con sus comentarios y sugerencias hicieron de esto, un mejor trabajo.

Dedicado a...

Recibid mi enseñanza, y no plata; Y ciencia antes que el oro escogido. Porque mejor es la sabiduría que las piedras preciosas; Y todas las cosas que se puede desear, no son de comparar con ella. El temor de Jehová es el principio de la sabiduría; Y la ciencia de los santos es inteligencia.

Esta tesis está dedicada con todo mi cariño a mi madre, que con su apoyo, comprensión y enseñanzas me ha formado un carácter solido para triunfar en todos los retos de mi vida.

También esta dedicada a *Carlos*, *Bugsy*, *Gamo*, *Miñan*, *Lopes* y *Opelo* por su apoyo incondicional durante toda mi formación académica y profesional.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Motivación

Los seres humanos somos capaces de reconocer rostros, identificar a una persona por su voz, diagnosticar enfermedades, decidir si un cliente puede recibir un crédito y reconocer códigos postales (manuscritos) en un sobre. Ahora bien, ¿cómo podríamos construir sistemas computacionales que pudieran desarrollar este tipo de comportamientos del ser humano? Si tomamos la habilidad de reconocer los rostros, el proceso se basa en cuatro fases: percepción, reconocimiento facial, identificación del sujeto familiar y recuperación del nombre.

En la fase de percepción se reconocen los rasgos básicos de una persona, como son nariz, boca y ojos para formar una percepción única del rostro. En la fase del reconocimiento facial la primera percepción se compara con caras memorizadas y se trata de encontrar una cara similar o familiar.

En la fase de identificación del sujeto familiar, otra parte del cerebro se activa y se confirma la relación cercana de una persona con otros elementos, como la voz, recuerdos y situaciones previamente almacenadas.

Finalmente, en la fase de recuperación de nombre, el cerebro puede confirmar la identidad de la persona. Del ejemplo anterior podemos ver que el proceso para reconocer rostros se basa en una técnica de clasificación con base a un conjunto de estímulos que llegan correctamente a las áreas visuales y la información almacenada en nuestra memoria de largo plazo.

En el aprendizaje maquinal y la estadística, la clasificación es el problema de identificar el conjunto de categorías (sub-poblaciones) a las que pueden pertenecer una nueva observación, sobre la base de un conjunto de datos de entrenamiento que contienen observaciones (o casos) con una pertenencia a un conjunto de categorías establecidas. Las observaciones individuales se analizan con un conjunto de propiedades cuantificables, conocidas como variables explicativas, características, etc. Estas propiedades pueden ser categóricas (e.g., "A", "B", "AB" u "O", para el tipo de sangre), ordinales (e.g., "grande" , "mediano" o "pequeño"), de valor entero (e.g., el número de apariciones de una palabra en un correo electrónico) o de valor real (por ejemplo, una medición de la presión arterial).

En aprendizaje maquinal existe un gran número de clasificadores que nos permiten realizar la clasificación de datos, por ejemplo, clasificadores bayesianos (Bayes Ingenuo[Elk97]), clasificadores basados en casos (k - vecinos más cercanos[FN75]), clasificadores del análisis de cluster (K - Medias[Mac67]), clasificadores de tipo árbol de decisión (*C4.5*[Qui93], *ID3*[Qui86], *ADtree*[FM99]), clasificadores basados en reglas de decisión (Tablas de Decisión[LL00]), redes Neuronales (Perceptrón Multicapa[Ros61]), clasificadores por hiperplanos (Máquinas de Soporte Vectorial[CV95]) y modelos ocultos de Markov[BP66], por mencionar sólo algunos ejemplos.

Cuando la cantidad de datos es menor a 200,000 registros, la clasificación de datos puede ser tratada por la mayoría de los clasificadores que mencionamos anteriormente, los cuales suelen tener buenos índices en las medidas de rendimiento. Sin embargo, en la actualidad la mayoría de las aplicaciones generan grandes cúmulos de datos debido al intercambio de multimedia, comunicación vía Internet, así como la gran capacidades de almacenamiento de las máquinas actuales las cuales superan fácilmente 200,000 registros. Esto nos trae como consecuencia que algunos algoritmos de clasificación no puedan manejar estas grandes cantidades de datos, ya que, por un lado, la complejidad temporal de los clasificadores aumenta por la cantidad de ejemplos dentro de los conjuntos de entrenamiento y prueba, y por el otro lado, la complejidad espacial también aumenta ya que los clasificadores tendrán que realizar mayor número de operaciones en las fases de entrenamiento y prueba.

La dimensionalidad de los datos también afecta el tiempo de ejecución de algunos clasificadores, por ejemplo, para el clasificador k-vecinos más cercanos con una dimensión de atributos mayor de a 10, el cálculo de medidas de similitud de cada uno de los ejemplos de prueba contra los ejemplos de entrenamiento suele ser muy demandante por el número de operaciones que se realizan. Aún si la base de datos sólo tuviera 20 mil registros para el conjunto de entrenamiento y 10 mil para el conjunto de prueba, en total se realizarían 2 mil millones de operaciones para obtener la clasificación final del conjunto de prueba.

Otro de los aspectos de la dimensionalidad es el número de clases presentes en los datos. En un escenario tradicional se realiza una clasificación con dos clases, sin embargo, existen bases de datos en las cuales se tiene más de dos clases teniendo de esta forma un escenario de una clasificación multiclase, el cual también representa un factor para el aumento de tiempo de ejecución de los clasificadores tradicionales.

Trabajos como los de Breiman [Bre96] y Schapire [Sch03], dieron origen a una nueva generación de clasificadores, en los cuales el objetivo principal radica en la combinación de varios modelos de un sólo tipo de clasificador como *Bagging* y *AdaBoost*, con el objetivo de aumentar los índices de medidas de rendimiento del modelo construido. Estos nuevos clasificadores llevaron el título de clasificadores basados en ensambles, los cuales también son conocidos como clasificadores híbridos, clasificadores basados en acoplamiento, sistemas híbridos, entre otros términos más, pero en este documento se va utilizar la primera denominación.

Estos clasificadores en un origen se idearon para que mejoren los índices de rendimiento, tales como exactitud, precisión y memoria, con respecto a los algoritmos de clasificación tradicionales. Los cuales, en efecto, presentaron una gran utilidad sobre grandes cantidades de datos reforzando el comportamiento conjunto de algunos clasificadores tradicionales. Es por ello que algunos trabajos se centraron en la utilidad de los clasificadores basados en ensambles, desarrollando dos modelos Generalización Apilada[Wol92], también llamado *Stacking* y Mezcla de Expertos[MU96].

Estos dos métodos utilizan el principio de no sólo combinar varios modelos de un sólo clasificador, si no que ahora la combinación se planteó sobre un conjunto de clasificadores de

base. Mediante esta combinación de los calificadores de base, estos ensambles presentaron un aumento en los índices de medidas de rendimiento, con respecto a ensambles que consideran solo varios modelos de un mismo clasificador, sin embargo, la complejidad temporal y espacial aumentó al tener varios modelos diferentes de clasificación.

Para los clasificadores de tipo generalización apilada, se implementa un criterio de votación por mayoría, en el que se suman los votos de cada clase presente en los datos. En el caso de los ensambles de tipo mezcla de expertos se combinan mediante un criterio en la votación ponderada, en el cual el problema radica en como asignarle el peso a cada clasificador de base. Esto normalmente se resuelve mediante el uso de una red neuronal de asignación, la cual después de un proceso de entrenamiento con un conjunto de datos, obtiene los pesos en base a la exactitud que tiene cada clasificador de base. Sin embargo, cualquier otro método de búsqueda informada o no informada se puede aplicar, por ejemplo: la búsqueda de coste uniforme, algoritmo $A*$, programación lineal entera y cualquier técnica heurísticas bio-inspiradas en la optimización.

Los ensambles de tipo mezcla de expertos son los menos trabajados en la actualidad, teniendo un conjunto pequeño de ejemplos en la literatura [MU96], [FP98], [MMMR11], [MMMM13], [MMMR14]. Esto se debe principalmente a la complejidad que conlleva su implementación, ya que además de considerar diferentes clasificadores de base, se tiene que implementar un método para asignar los pesos a cada clasificador, el cual nos permita obtener mejoras con respecto al criterio de votación por mayoría (peso igualitario) que utilizan la mayoría de ensambles.

Nos podemos preguntar, ¿qué ventajas nos ofrece implementar un ensamble de tipo mezcla de expertos?, como se vio anteriormente tenemos que implementar por separado un conjunto de clasificadores de base tomando como referencia aquellos mejores reportados en la literatura con respecto a su funcionamiento tradicional, como los que se mencionan en el trabajo de Witten[WFH11]. Además de esto, tenemos que desarrollar un método de búsqueda para la asignación de pesos que ofrezcan un beneficio en cuestión de medidas de rendimiento, los cuales en la mayoría de los casos solo son capaces de encontrar aproximaciones al óptimo

local de la solución final debido a un espacio de búsqueda amplio (número de combinaciones posibles).

Mediante la implementación y comparación de un ensamble de tipo mezcla de expertos, que se desarrolló en un proyecto de investigación previo [MMMR11], denominado Clasificador Híbrido con Ponderación Genética (*Hybrid classifier with genetic weighting - HCGW*), se obtuvo una mejoría estadísticamente significativa en los índices de medidas de rendimiento con respecto a un conjunto de clasificadores tradicionales y algunas mejoras similares al trabajo de Witten[Fly72]. También se comprobó que se obtiene una ventaja estadísticamente significativa con respecto a los métodos tradicionales mediante una prueba de tipo *t-test*. En este esquema, la estrategia de combinar varios modelos diferentes en términos de distribución de datos nos ofrece un espacio de decisión más amplio que el que nos ofrece un clasificador tradicional.

Sin embargo, la principal limitante del *HCGW* son los altos tiempos de ejecución para realizar la clasificación, por ejemplo, con bases de datos de más de un millón de registros se obtiene un tiempo de ejecución de más de 7 horas[MMMR11]. Para las pruebas se utilizó validación cruzada con 2 subgrupos, una base de datos con 1,025,010 registros, 10 atributos y 11 clases utilizando una versión secuencial de cada clasificador de base y del *HCGW*. Estos tiempos de ejecución son la consecuencia de la suma de los tiempos de ejecución de los clasificadores de base, el método para asignar los pesos a cada uno de estos y el módulo para combinar cada una de las clasificaciones individuales de los clasificadores de base.

Otra de las limitantes que tiene el *HGCW* es que todo el funcionamiento radica en combinar un conjunto de clasificadores tradicionales con esquemas secuenciales, los cuales no pueden ofrecer mayor aporte a las medidas de rendimiento ya que su algorítmica establece criterios de paro al manejar grandes cantidades de datos, que en la mayoría de los casos afecta directamente sobre estas medidas.

Considerando todo lo anterior, en este trabajo de investigación doctoral se presenta una forma de resolver la clasificación sobre grandes cantidades de datos, mediante el uso de un sistema paralelo de clasificación basado en un ensamble de tipo mezcla de expertos.

Con este sistema de clasificación paralelo se ofrece una forma más robusta de realizar la clasificación sobre grandes cantidades de datos, la cual se ajusta automáticamente a diferentes características en los datos.

Por una parte, al utilizar un ensamble de tipo mezcla de expertos, dado los trabajos previos, podemos obtener mejoras en los índices de rendimiento del clasificador, sin embargo, queda pendiente el problema de los altos tiempos de ejecución. Por otro lado, mediante la incorporación de cómputo paralelo se propone una solución a los problemas de los altos tiempos de ejecución y manejo de grandes cantidades de datos, ya que al aumentar el poder de cómputo mediante un sistema multicomputador se distribuyen las complejidades temporales y espaciales de un ensamble secuencial. En la siguiente sección explicaremos con más detalle la hipótesis que se plantea para el desarrollo de este proyecto de investigación doctoral.

1.2. Hipótesis planteada

Las hipótesis planteadas para este trabajo de investigación son las siguientes:

1. Existen trabajos previos como el de [Fly72] y el de [MMMR11] que corroboran que un ensamble presenta mejoras con respecto a clasificadores tradicionales, sin embargo, al implementar un esquema paralelo de clasificación es posible obtener mejoras en medidas de rendimiento y reducir los tiempos de ejecución de los esquemas tradicionales de construcción de ensambles secuenciales reportados en la literatura.

2. Normalmente se utiliza una red neuronal de agregación para combinar las clasificaciones individuales del ensamble de tipo mezcla de expertos, sin embargo, es posible obtener mejoras en el rendimiento del ensamble si se exploran otros métodos de búsqueda no informada e informada como los algoritmos genéticos, programación lineal entera, optimización multiobjetivo, por solo mencionar algunos de ellos.

3. Mediante la incorporación del cómputo paralelo sobre algunos algoritmos de clasificación, es posible obtener una reducción de menos de la mitad de los tiempos de ejecución del esquema secuencial.

4. Mediante un esquema de paralelismo global sobre un clasificador de tipo ensamble, esto es, cada componente del clasificador se encuentra modelado como un componente paralelo, es posible obtener mejoras en los índices de las medidas de rendimiento de algunos clasificadores de base, ya que, cada implementación paralela de algunos de estos componentes es una mejora con respecto al esquema secuencial.

5. Mediante la implementación de un sistema de clasificación paralelo, es posible que se pueda adaptar a conjuntos de datos diferentes, ya que se podría ajustar a diferentes tamaños, distribuciones, diferentes tipos y número de atributos y clases.

Hemos llegado a la parte final de este capítulo en el cual resta por presentar los objetivos que se plantean para darle solución a los principales factores de la problemática de clasificación sobre grandes cantidades de datos, en donde uno de los objetivos particulares es hacer uso del cómputo paralelo. Una vez que presentemos los objetivos, mostraremos la justificación de los mismos.

1.3. Objetivos

1.3.1. Objetivos Generales

Desarrollar un Sistema de Clasificación Paralelo basado en un Ensamble de tipo Mezcla de Expertos, *Parallel Classification System based on an Ensemble of Mixture of Experts - PCEM*, para evaluar y comparar su desempeño con respecto a métodos tradicionales y paralelos de clasificación, a fin de obtener mejores porcentajes de medidas de rendimiento y reducción en los tiempos de ejecución utilizando grandes cantidades de datos reales.

1.3.2. Objetivos Particulares

- Establecer una arquitectura paralela adecuada para la incorporación del *PCEM*, tomando en cuenta los principales sistemas y herramientas de cómputo paralelo reportados en la literatura.

- Implementar versiones paralelas de cada uno de los clasificadores de bases seleccionados para la construcción del *PCEM*.

- Utilizar un esquema paralelo de un algoritmo genético para encontrar las ponderaciones de cada uno de los esquemas paralelos de los clasificadores de base definidos en la estructura del PCEM.

- Implementar el *PCEM* para la clasificación sobre grandes cantidades de datos reales.

- Analizar el comportamiento de los resultados del *PCEM* con respecto a un conjunto de clasificadores actuales, para realizar la estimación de los principales aportes del proyecto de investigación doctoral.

1.4. Justificación

Como se mencionó anteriormente, la presencia de grandes cantidades de datos es uno de los factores de la problemática de clasificación. Este trabajo se enfoca de manera general a tres factores, que son, manejo de grandes cantidades de datos, altos tiempos de ejecución y bajos índices de medidas de rendimiento. Adicionalmente, se enfoca al problema de sobreajuste (también es frecuente emplear el término en inglés *overfitting*) de los clasificadores, el cual se refiere a que los clasificadores presentan un sobreentrenamiento a un tipo de datos en particular, lo cual produce una mala clasificación sobre datos con diferentes características de los que estuvieron presentes en la fase de entrenamiento.

Los clasificadores tradicionales tienen la característica de cubrir una cierta zona de la frontera de decisión de los datos. Al aumentar la complejidad de esta, los clasificadores suelen verse afectados en los índices de las medidas de rendimiento ya que solo tienen la posibilidad de cubrir ciertas regiones, teniendo así una situación del sobreajuste de las zonas de decisión de los datos.

Los clasificadores basados en ensambles aumentan la posibilidad de cubrir fronteras de decisión complejas mediante la combinación de diferentes modelos de un solo clasificador

o varios clasificadores diferentes, obteniendo mejoras de más del 5 % sobre los índices de las medidas de rendimiento comparándolos contra modelos de clasificación individual[Pol06], [Die00], [NYD09].

Las pruebas de clasificación deben realizarse con esquemas de tipo validación cruzada para evitar condiciones de sobreajuste en la construcción del clasificador. La validación cruzada consiste en utilizar diferentes conjuntos de entrenamiento y prueba de manera iterativa con el objetivo de cubrir la mayor parte de la distribución de los datos.

Algunos de los clasificadores tradicionales suelen tener limitantes en los tamaños de los datos que pueden manejar, normalmente se puede hablar que después de los 200 mil registros, la complejidad temporal y espacial aumenta teniendo tiempo de ejecución de más de 3 horas[MMMR11].

Una solución a los altos tiempos de ejecución de los clasificadores del aprendizaje maquinal es utilizar estrategias de mejora en optimización de código, como, por ejemplo; utilizar estructuras de datos dinámicas en vez de estructuras de datos estáticas y métodos recursivos en vez de iterativos[HSTZ17], que reducen tanto complejidades temporales como espaciales.

Las estrategias de optimizar el código mediante el uso de diferentes estructuras de datos o de reducción de la complejidad de un algoritmo (uso de recursividad) siguen siendo insuficientes en cuestiones lograr disminuciones significativas en los tiempos de ejecución de los algoritmos. Esto se presenta ya que al ser versiones secuenciales se tienen que seguir una serie de instrucciones de tipo *top-down* o de forma ordenada, esto es, hasta que no se termine el proceso A, no se comienza con el B aun en el caso de que sean independientes.

Es por ello que cuando la carga de procesamiento y el espacio de los datos alojados en la memoria RAM aumentan, una de las opciones es utilizar cómputo paralelo. Mediante el cómputo paralelo por un lado se mejora el procesamiento máximo de las diferentes maquinas o nodos y por otra parte se tiene una capacidad mayor de almacenamiento de memoria RAM que se encuentra distribuidas en el sistema de cómputo paralelo en el que se esté trabajando, normalmente llamados sistemas multicomputadora.

Además de las mejoras en cuestión de tiempos de ejecución, el cómputo paralelo ayuda a

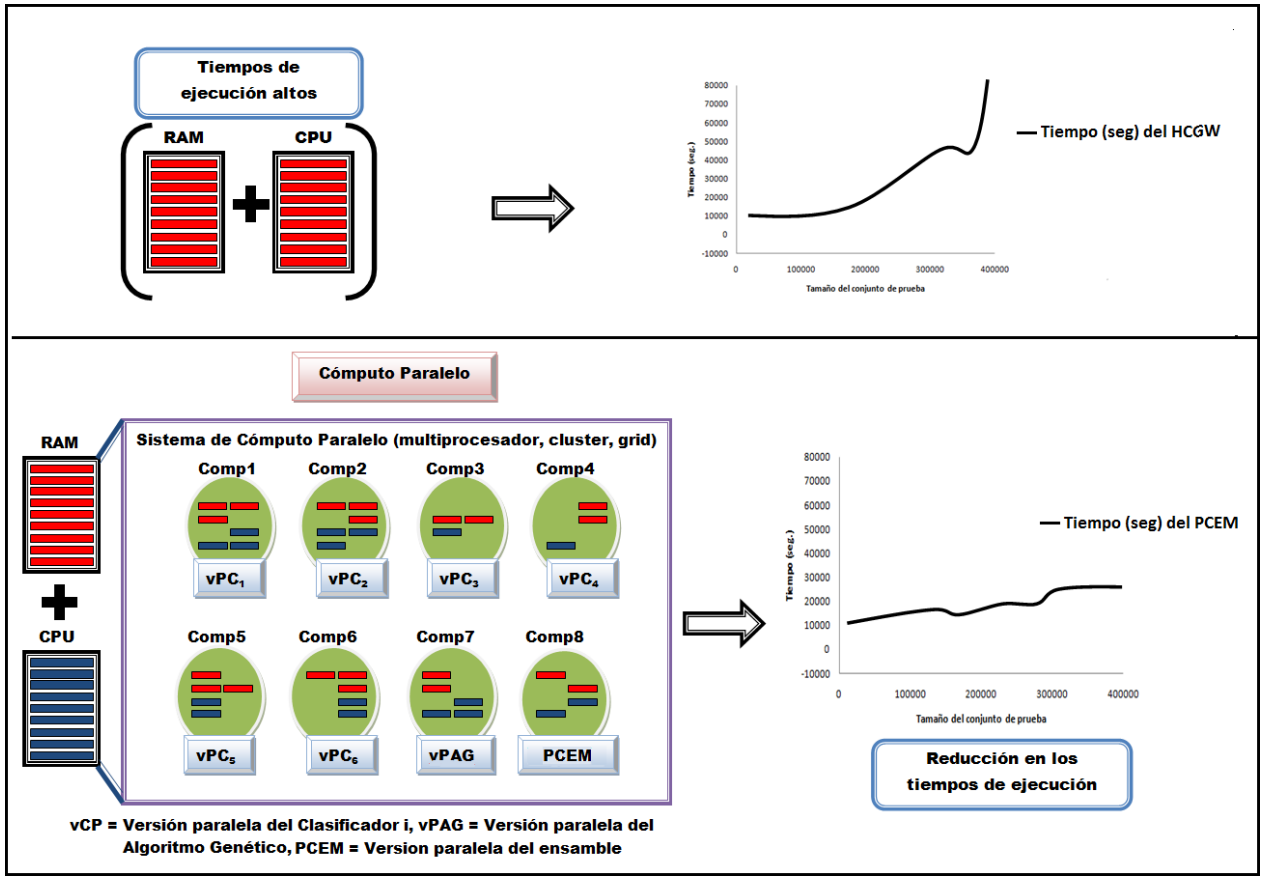
obtener mejoras en el rendimiento de los algoritmos, ya que algunos están limitados a tener un menor número de pasos iterativos o umbrales establecidos en los esquemas secuenciales. Esto también se presenta en algunos esquemas secuenciales de los clasificadores del aprendizaje maquinal lo que lleva a limitar los índices de medidas de rendimiento, es por ello que se al usar cómputo paralelo se pueden obtener mejoras con respecto a los esquemas tradicionales de ciertos clasificadores, sin embargo, en esquemas como Bayes Ingenuo esto no sucede ya que los clasificadores siempre obtienen los mismos índices de medidas de rendimiento para una base de datos en particular.

Al aplicar cómputo paralelo se pueden resolver diferentes problemas, como por ejemplo reducción de tiempos, consumo de Memoria RAM excesivo, manejo de gran cantidad de datos, división de problemas, uso de máximo poder de cómputo, distribución de procesamiento. Para esta propuesta se pretende solventar el problema del tiempo de ejecución, el cual como se revisó anteriormente, se debe al alto porcentaje de los recursos de cómputo (manejo de grandes cantidades de datos y número de operaciones). Para explicar este punto vamos a revisar el esquema de la Figura 1.1 en el cual se muestra que es lo qué se pretende realizar al utilizar cómputo paralelo.

En la Figura 1.1 podemos observar como pretendemos atacar el problema del alto tiempo de ejecución, en este caso se pretende usar un sistema de cómputo paralelo (cluster, grid, gpu's o un sistema miltiprocesador), en el cual radicaran las versiones paralelas de cada uno de los clasificadores de base, un esquema paralelo de un algoritmo genético para encontrar las ponderaciones de cada clasificador de base y un componente para realizar la clasificación ponderada de los datos.

Algunas de las razones más comunes del porque aplicar computo paralelo con respecto a versiones secuenciales son: aumentar la velocidad, procesar gran cantidad de datos, y resolver problemas en un tiempo razonable. Utilizando particionamiento y asignacion de tareas en diferentes nodos o procesadores se aprovecha el procesamiento de una manera efectiva y se puede verificar si estos modelos de algoritmos paralelos responden adecuadamente a las arquitecturas de sistemas de cómputo diseñados para soportar aplicaciones paralelas.

Figura 1.1: Comparación de tiempos de ejecución entre el esquema secuencia vs esquema paralelo del ensamble tipo mezcla de expertos.



Algunos medidas de rendimiento utilizadas para evaluar aplicaciones paralelas son[HJ11]:

- Tiempo de ejecución
- Memoria
- Número de procesadores implicados
- Costo: Relación entre tiempo de aplicación paralela y número de procesos
- *Speedup* ó Aceleración ó Rapidez ($S(n)$): Indicar el grado de ganancia de velocidad de una aplicación paralela.
- Eficiencia ($E(n)$): Mide la porción útil del trabajo total realizado por n procesadores.

- Escalabilidad: Como se comporta una aplicación paralela al aumentar el número de procesadores.
- Redundancia (*R(n)*): Mide el grado del incremento de la carga.
- Utilización (*U(n)*): Indica el grado de utilización de recursos durante un cálculo paralelo.
- Calidad del Paralelismo (*Q(n)*): Combina el efecto del speed-up, eficiencia y redundancia en una única expresión para indicar el mérito relativo de un cálculo paralelo sobre un sistema.

En la siguiente sección mostraremos como se encuentra organizado este documento, mencionando a grandes rasgos el contenido de cada uno de los capítulos.

1.5. Organización de la tesis

Esta tesis se encuentra organizada de la siguiente forma.

En el **Capítulo 2**, presentaremos algunos antecedentes, en particular áreas de investigación relacionadas con el desarrollo del *PCEM*.

En el **Capítulo 3** presentaremos una revisión del estado del arte, en el cual veremos los tipos de clasificadores basados en ensambles que están reportados en la literatura, también revisaremos esquemas secuenciales y paralelos de algunos clasificadores tradicionales. Dentro de este capítulo, también se hará un breve repaso sobre los algoritmos genéticos y cómputo paralelo, en donde para cada caso mencionaremos algunos conceptos básicos, así como un ejemplo de uso de ambos.

En el **Capítulo 4** describiremos a detalle cada uno de los componentes que conforman el *PCEM*. Por una parte, se describirá el funcionamiento de cada esquema paralelo de los clasificadores de base seleccionados, y por otra parte, describiremos el esquema paralelo utilizado para el Algoritmo Genético y finalmente revisaremos las etapas de funcionamiento del *PCEM*.

En el **Capítulo 5** revisaremos las pruebas y los resultados obtenidos por el *PCEM* al compararlo contra un conjunto de clasificadores tradicionales y paralelos. En este Capítulo mostraremos cuales fueron los datos seleccionados para dichas pruebas, describiendo los componentes de cada una de ellas, también revisaremos cuales fueron algunas de las pruebas estadísticas a las cuales fue sometido el *PCEM*, para obtener las conclusiones finales.

Finalmente, en el **Capítulo 6** revisaremos las conclusiones y trabajo a futuro que se obtienen de este trabajo.

Capítulo 2

Antecedentes

2.1. Introducción

En este capítulo se muestran cuatro áreas de investigación que se encuentran relacionadas con la implementación del PCEM (*Parallel Classification System based on an Ensemble of Mixture of Experts*), de las cuales mencionamos los aspectos más generales de cada una. Las áreas de investigación que examinaremos son: Aprendizaje Maquinal, Clasificación, Minería de Datos y la Extracción de Conocimiento en Bases de Datos.

Para el caso de aprendizaje maquinal, revisaremos algunos conceptos básicos, dando de esta manera un bosquejo general de las aplicaciones y tipos de aprendizaje maquinal reportados en la literatura. En particular veremos que, dentro del aprendizaje maquinal, se tienen tres opciones, el aprendizaje supervisado, no supervisado y por refuerzo, de los cuales una de las tareas que es de gran relevancia para este trabajo de investigación doctoral es la clasificación de datos.

Como vimos el segundo tema de investigación a revisar en este capítulo, es la tarea de clasificación de datos, en donde se presenta un bosquejo general, aplicaciones del mundo real y componentes del problema de clasificación de datos. Dentro de estos componentes, están los clasificadores del aprendizaje maquinal, de los cuales mostraremos algunos esquemas tradicionales, así como un tipo en particular, que son los clasificadores basados en ensambles, los cuales se utilizaron para la construcción del PCEM.

Una vez presentada esta revisión, describiremos la tercera área de investigación importante

para este trabajo que es la minería de datos. En minería de datos presentaremos la relación que existe entre aprendizaje maquinal y clasificación de datos, ya que veremos que una de las tareas más importantes en minería de datos es precisamente la de clasificación de datos.

Finalmente, en este capítulo revisaremos la última área de investigación relacionada con el desarrollo del PCEM, que es el proceso de extracción de conocimiento en bases de datos. Este proceso nos permitirá platear un problema relacionado con las áreas de investigación que revisaremos en este capítulo, llegando a la parte final del mismo, con el planteamiento de la problemática que demanda la clasificación de grandes cantidades de datos.

2.2. Aprendizaje Maquinal

La mayoría de las técnicas tradicionales dentro de Inteligencia Artificial [Mit97], tienen por objetivo desarrollar aplicaciones que dan una solución a problemas en los que se presenta un alto grado de complejidad, en los que a veces la solución óptima no es factible de encontrar. Es por ello que estas aplicaciones ofrecen soluciones parciales que suelen dar buenos resultados. Para construir dichas aplicaciones, los desarrolladores deberán incluir conocimiento del dominio del problema para facilitar su resolución, logrando con esto la reducción del costo computacional que implica el uso de técnicas convencionales.

Con frecuencia estas aplicaciones suelen ser programadas para dar solución a un problema en específico, sin embargo, al momento que se quiere generalizar el uso de estas para casos similares, se necesita volver a programarlas agregando nuevo conocimiento para solucionar estos nuevos problemas. Podemos darnos cuenta que una de las principales limitaciones de este tipo de aplicaciones radica en que no podrán resolver problemas para los que no hayan sido programadas. Es aquí donde el conocimiento que vayamos integrando aumenta el uso que le podemos dar a estas aplicaciones.

Una aplicación que sea capaz de adaptarse y poder integrar nuevo conocimiento, a medida que se usa, de manera automática, se considera más cercana a ser realmente una aplicación inteligente, ya que podrá resolver nuevos problemas que nosotros le proporcionemos sin esperar una reprogramación con conocimiento nuevo que le ayude a atacar estos problemas.

En del área de investigación llamada Aprendizaje Maquinal, dentro de la Inteligencia Artificial, podemos desarrollar aplicaciones en las que se tenga una adaptación al conocimiento de manera automática.

La idea dentro del aprendizaje maquinal es buscar métodos que logren aumentar las capacidades de las aplicaciones habituales (sistemas basados en el conocimiento, tratamiento del lenguaje natural, robótica, genética, etc.), de manera que puedan ser más flexibles y eficaces. Hay muchas aplicaciones que se le pueden dar al aprendizaje maquinal para la solución de problemas usando inteligencia artificial, a continuación, se muestran tres usos [Béj06]:

- **Tareas difíciles de programar:** Es muy frecuente que nos encontremos con tareas excesivamente complejas en las que construir un programa para resolverlas resulta muy difícil. Pongamos el ejemplo en el que se pretende desarrollar un sistema de visión que sea capaz de reconocer un conjunto de rostros, sería muy complicado desarrollar una aplicación con técnicas tradicionales que resuelva este reconocimiento. Mediante el aprendizaje maquinal podemos construir un modelo de clasificación a partir de un conjunto de ejemplos, para realizar la tarea del reconocimiento y de esta manera hacer la solución del problema más sencilla.

- **Aplicaciones auto adaptables:** Hay ciertas aplicaciones que tienen un mejor rendimiento si son capaces de adaptarse a las circunstancias e ir aprendiendo a lo largo de su ejecución. Podemos tener por ejemplo aplicaciones de interfaces que sean adaptables al uso continuo del usuario y poder ofrecer un mejor servicio. Un ejemplo de estas aplicaciones son las ayudas virtuales que ofrecen los aeropuertos de Europa, en particular en el Aeropuerto de Barajas, España. Esta aplicación consta de un holograma de una azafata virtual la cual apoya a los usuarios con las dudas que surjan acerca de sus vuelos, así como apoyarlos en la compra de boletos de avión mediante el trazo de trayectos en diferentes horarios y precios. Esta aplicación se retroalimenta a medida que dan el servicio, teniendo como consecuencia una aplicación más útil a los usuarios.

- **Minería de Datos/Descubrimiento de conocimiento:** El aprendizaje puede servir para ayudar a analizar información, extrayendo de manera automática conocimiento a partir de conjuntos de ejemplos (usualmente millones) y descubriendo patrones complejos. En minería de datos se utilizan algoritmos de clasificación del aprendizaje maquinal, que normalmente se les conoce como clasificadores, para realizar dicha extracción, la cual la describiremos en la sección 2.3.

El aprendizaje maquinal se divide en cuatro tipos de aprendizaje según [Mit97]:

- **Aprendizaje inductivo:** En este tipo de aprendizaje se crean modelos de conceptos o descriptores que posean características comunes de los ejemplos de entrenamiento.

- **Aprendizaje analítico o deductivo:** Es aplicada la deducción para obtener descriptores de un ejemplo, de un concepto y su descripción.

- **Aprendizaje genético:** Estos algoritmos son inspirados en la Teoría de la Evolución para encontrar descriptores generales de los ejemplos.

- **Aprendizaje conexionista:** Se desarrollan descriptores generales mediante el uso de las capacidades de adaptación de redes neuronales artificiales. Una red neuronal está compuesta de elementos simples interconectados que poseen algún estado. Tras un proceso de entrenamiento, el estado en el que quedan las neuronas de la red representa el concepto aprendido.

En particular en el aprendizaje inductivo podemos distinguir dos tipos el aprendizaje supervisado y el no supervisado. En el aprendizaje supervisado se tiene un conjunto de ejemplos, los cuales tienen asociados un atributo especial llamado la clase de cada ejemplo, por lo que tendremos entonces mecanismos de entrenamiento con base en este conjunto, para permitirnos distinguir las clases sobre ejemplos de prueba que proporcionemos.

Por otro lado, el aprendizaje no supervisado sólo se cuenta con un conjunto de ejemplos que no tiene asociado ninguna clase, por lo que se debe de encontrar una manera de

agruparlos. Para los métodos de aprendizaje no supervisado se utiliza el concepto de similaridad/disimilaridad de los ejemplos, para poder construir grupos en los que ejemplos similares estén juntos en un grupo y separados de otros ejemplos menos similares. En la siguiente sección mostraremos a detalle una de las principales tareas que se lleva a cabo mediante el aprendizaje supervisado, que es la clasificación de datos.

2.3. Clasificación de datos

Una de las tareas que podemos realizar mediante el aprendizaje supervisado es la clasificación de datos, con la cual podemos categorizar a un conjunto de ejemplos en base a datos de entrenamiento con clases implícitas para cada ejemplo. En el aprendizaje supervisado el proceso de clasificación se define de la siguiente forma:

- Entrada: Conjunto de ejemplos (m ejemplos) previamente clasificados (X_1, Y_1), $\cdots$, (X_m, Y_m) (Conjunto de entrenamiento).
- Salida: Un clasificador entrenado o configurado.

Como habíamos mencionado, uno de los usos del aprendizaje maquinal se da en la minería de datos, en la cual se encuentran una serie de tareas que nos permiten desarrollar dicha fase. En particular la clasificación de datos es una de las tareas más importantes debido al uso que tiene en aplicaciones del mundo real. Algunas aplicaciones en las que están presentes clasificadores del aprendizaje maquinal se enlistan a continuación[HORQFR04]:

1. **Financieras:** Detección de uso fraudulento de tarjetas de crédito, clasificación del gasto en tarjeta de crédito por grupos, análisis de riesgos en concesión de créditos, identificación de reglas de mercado a partir de datos históricos, así como la caracterización de posibles clientes morosos para implementar estrategias de cobro para cada tipo.
2. **Comercio:** Análisis de la cesta de la compra, evaluación de campañas publicitarias, segmentación de clientes, estimación de ventas de acciones (*stock sale*).

3. **Seguros:** Determinación de clientes potencialmente caros, clasificación de qué tipo de clientes contratan nuevas pólizas, identificación de patrones de comportamiento para clientes con riesgo, identificación de comportamiento fraudulento.

4. **Medicina:** Diagnóstico de enfermedades, detección de pacientes con riesgo de sufrir una patología concreta, clasificación de imágenes médicas.

5. **Bioinformática:** Búsqueda de genes (regiones codificantes del genoma), clasificación de la estructura secundaria de las proteínas, búsqueda de secuencias de ADN a partir de datos de microarreglos [MMMM13], los cuales contienen toda la información de los cromosomas.

6. **Otras áreas:** Aplicación de telecomunicaciones en la detección de fraudes. En correo electrónico y agendas personales, particularmente clasificación y distribución automática de correo y detección de correo spam. En Hacienda con la detección de riesgo a fraude fiscal. Una aplicación de clasificación la podemos encontrar en la Web, con el análisis del comportamiento de los usuarios y análisis de los archivos de bitácoras (*log*) de un servidor web. Finalmente, también podemos tener implícita la clasificación en los Deportes con la detección riesgo de lesiones a partir de datos médicos.

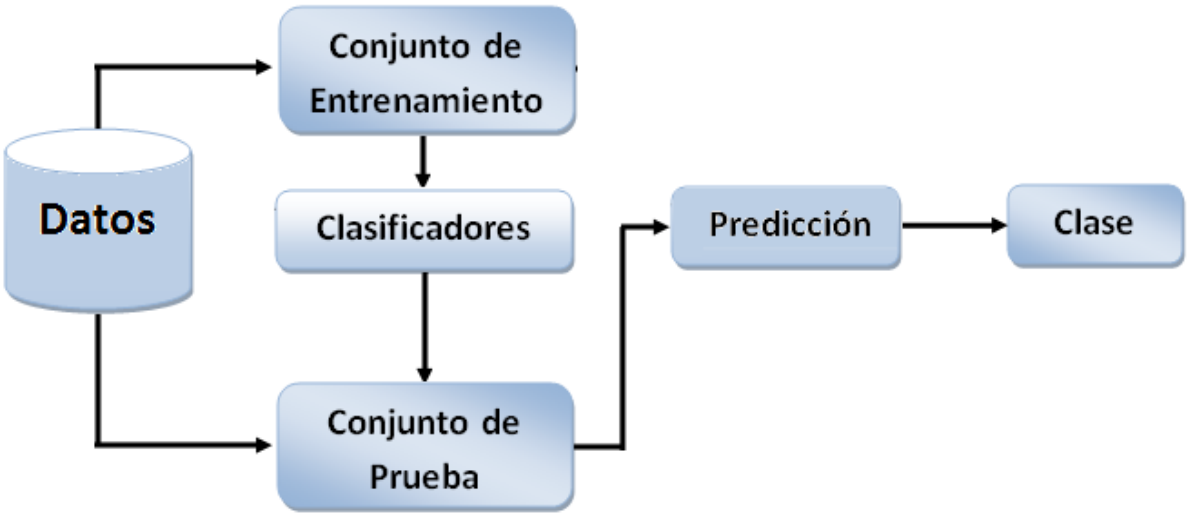
Podemos ver que la clasificación tiene muchas áreas en la que podemos aplicarla, para la cual en la mayoría de los casos se necesita el uso de clasificadores del aprendizaje maquinal. Es importante el desarrollo de clasificadores que sean capaces de auto adaptarse a condiciones diversas de clasificación de manera automática. Podemos darnos cuenta que en efecto estamos haciendo el uso del concepto de aprendizaje maquinal, aplicado a los clasificadores. ¿Y por qué de la necesidad de este tipo de clasificadores? Bueno podríamos resolver esta pregunta mencionando uno de los tantos problemas que se presentan en clasificación, mediante el uso del Teorema de *no-free-lunch* [WM97].

Este teorema aplicado a la clasificación nos dice que no hay ningún clasificador que de excelentes resultados para todos los datos que existen en el mundo, siempre podrá existir otro que sea mejor que el primero para algunos datos en particular. Y esto precisamente se

da porque hay clasificadores que se adaptan a ciertos tipos, tamaños y formas de datos, sin que existan métodos para que los clasificadores pudieran ajustarse a datos que no presenten estas características.

En la Figura 2.1 se muestra un esquema básico de un clasificador del aprendizaje maquinal, en el cual se muestra como a partir de un conjunto de datos se seleccionan dos conjuntos, uno de entrenamiento para aplicar una fase de entrenamiento a cada clasificador utilizado, para que de esta forma se pueda pasar a clasificar el conjunto de prueba del cual desconocemos la clase de cada ejemplo de prueba.

Figura 2.1: Esquema de un modelo de clasificación del aprendizaje supervisado.



A continuación, mostraremos algunos ejemplos de clasificadores del aprendizaje maquinal, en particular veremos una pequeña introducción de un tipo en especial de clasificadores, que son los clasificadores basados en ensambles, los cuales en pruebas recientes han demostrado una mejoría en los índices de medidas de rendimiento con respecto a esquemas tradicionales.

2.4. Clasificadores del aprendizaje maquinal

Existen una gran cantidad de algoritmos de clasificación que pretenden dar solución a problemas de clasificación, como el de obtener mejoras en los índices de rendimiento, bajos tiempos de ejecución y manejo de grandes cantidades de datos. Dentro del artículo titulado *Top 10 algorithms in data mining* de Wu et al. [WKRQ$^+$08], se menciona una revisión de los principales clasificadores utilizados para el año 2008, los cuales se enlistan en la Tabla 2.1.

Tabla 2.1: Los 10 de algoritmos de clasificación más utilizados según Wu et al., considerando los dos tipos de aprendizaje inductivo.

Caso	*Nombre*	*Tipo de aprendizaje*
1	*K*-Medias	Aprendizaje no supervisado
2	*k*-vecinos más cercanos	Aprendizaje supervisado
3	Bayes Ingenuo	Aprendizaje supervisado
4	AdaBoost	Aprendizaje supervisado
5	Máquinas de soporte vectorial (SVM)	Aprendizaje supervisado
6	Apriori	Aprendizaje supervisado
7	C4.5	Aprendizaje supervisado
8	CART	Aprendizaje supervisado
9	PageRank	Aprendizaje supervisado
10	Expectation−Maximization (EM)	Aprendizaje supervisado

Este trabajo es tomado como referencia e inclusive se han actualizado los clasificadores, tal es el caso de *The University of Vermont* que tiene una actualización del 2016 del *Top 18 de Clasificadores de Minería de Datos* en la página: *http://www.cs.uvm.edu/icdm/algorithms/CandidateList.shtml*.

Algunos de los clasificadores mostrados en la Tabla 2.1 han mostrado buenos resultados, al tratar de resolver alguno de los tres problemas anteriormente planteados. Sin embargo,

es poco común encontrar algoritmos que resuelvan todos los problemas, ya que siempre se presentan limitantes en alguno de ellos.

Se pueden tener clasificadores con altos índices en medidas de rendimiento, por ejemplo las Máquinas de Soporte Vectorial, pero por otro lado tener limitantes en cuestión de datos, debido a que este tipo de clasificadores consumen demasiado tiempo de ejecución por la complejidad del algoritmo, el cual requiere un orden computacional del $O(n^3)$[CB01].

Otro ejemplo lo podemos ver con clasificadores basados en análisis de cluster, como por ejemplo el algoritmo *K*-Medias [ZXMO06], este tipo de clasificador puede manejar grandes cantidades de datos, debido a la simplicidad del algoritmo, obteniendo tiempos de ejecución relativamente bajos. Sin embargo, en cuestión de medidas de rendimiento suelen tener porcentajes bajos, que en la mayoría de los casos es lo más importante.

2.5. Clasificación por ensamble

Dentro de la Tabla 2.1, podemos localizar el clasificador llamado *AdaBoost*, este clasificador pertenece a una nueva generación de clasificadores desarrollados a finales de los 90's, denominados Clasificadores basados en Ensambles (Ensamble). Cabe aclarar que existen muchos nombres para este tipo de algoritmos de clasificación, como Clasificadores Híbridos, Sistemas Expertos y Clasificadores basados en Acoplamiento, por lo que para este trabajo para mayor simplicidad nos referiremos a ellos como Ensambles. El objetivo de estos clasificadores es combinar varios modelos de clasificación, para obtener mejoras en las medidas de rendimiento al clasificar grandes cantidades de datos. En particular se pueden tener dos enfoques distintos de cómo construir este tipo de clasificadores, uno es considerar diferentes modelos de un clasificador como árboles de decisión y el otro es considerar diferentes algoritmos de clasificación o diferentes clasificadores.

La primera opción consiste en combinar diferentes modelos de un sólo tipo de clasificador, lo que representa la forma más usual de llevar a cabo la construcción de dichos clasificadores. Hastie et al. [Sch03], Hüllermeier et al. [BK99], Polat et al. [PG09], Koltchinskii [KPL01] y Houeland et al. [HA11], consideran la construcción de estos clasificadores usando diferentes

modelos de Árboles de Decisión. El considerar sólo árboles de decisión solía ser la forma más utilizada al construir un Ensamble de tipo *Boosting* y *Bagging*.

Sin embargo, en los últimos años se han reportado esquemas paralelos usando otro tipo de clasificadores, tal es el caso del Ensamble reportado por Kumar et. al [KMHH11], el cual propone un Ensamble utilizando diferentes modelos del clasificador K-medias. También cabe mencionar que en la actualidad se trabaja sobre otro tipo de ensambles que siguen este modelo de construcción, como los trabajos reportados por Melin et al. [MC14, SM14, GMVC14], en el cual se propone una forma de construir un ensamble con lógica difusa de tipo 1 y 2.

La segunda forma de cómo implementar un Ensamble, consiste en combinar diferentes esquemas de clasificación[WKRQ$^+$08] como los que vimos en la Tabla 2.1, en este caso cada uno tendrá una forma diferente de cómo llevar a cabo la tarea de clasificación. Dentro de la literatura se encuentran pocos trabajos en los cuales se implementa un Ensamble siguiendo esta forma de construcción.

Ya sea la primera o la segunda forma de cómo construir este tipo de clasificadores, se debe de seleccionar qué criterio vamos a utilizar para combinar cada uno de los modelos generados. Para ello existen dos formas, la primera es realizar un criterio de votación por mayoría, en el cual se asigna la clase de mayor número de votos a cada ejemplo de prueba. La segunda forma consiste en implementar un criterio de votación ponderado, en el cual cada uno de los modelos tiene un peso diferente de acuerdo a algún mecanismo seleccionado (redes neuronales, algoritmos genéticos y programación lineal), por lo tanto, al final se realizara una suma ponderada de votos para obtener la clasificación del conjunto de prueba.

2.6. Minería de Datos

Una de las áreas que ha tenido mayor avance en los últimos años es la de minería de datos [Béj06], esto se debe principalmente al incremento del tamaño de las Bases de Datos. Al ocurrir esto se tiene como resultado un aumento potencial del conocimiento implícito, por lo que al usar minería de datos podemos realizar la exploración y el análisis de estas bases de datos (BD). Obteniendo la identificación de patrones no triviales que nos permitan obtener

conocimiento novedoso y potencialmente útil.

La obtención de este conocimiento está relacionada con el tipo de modelos que se planten en la fase de la minería de datos, los cuales son divididos en las siguientes dos categorías[WFH11]:

- **Modelos descriptivos:** En estos modelos se identifican los patrones que explican o resumen los datos. Para este tipo de modelos normalmente se utilizan dos métodos para la identificación de estos patrones, reglas de asociación y agrupamiento. Por un lado, mediante las reglas de asociación se expresan patrones de comportamiento en los datos. Por el otro lado las reglas de agrupamiento nos permiten juntar casos homogéneos, utilizando una serie de características comunes referidas a la clase de los datos.

- **Modelos predictivos:** En este tipo de modelos se estiman valores de variables de interés (a predecir) a partir de valores de otras variables en el conjunto de entrenamiento (predictoras). Este tipo de modelos utiliza la Regresión cuando la variable a predecir es continua o Clasificación cuando la variable a predecir es discreta (nominal u ordinal).

Además de los dos grupos que se revisaron, la minería de datos se divide en dos categorías según la función que puede realizar en base a los dos tipos de aprendizaje que fueron revisados en la sección 2.2, las cuales son supervisado y no supervisado[HORQFR04], a continuación, se presenta la diferencia entre ambos:

- **Minería de datos supervisada:** El aprendizaje supervisado es el proceso en el que el aprendizaje es dirigido por un atributo u objetivo dependiente previamente conocido. Del aprendizaje supervisado generalmente resulta en modelos predictivos. Siendo este el contraste para el aprendizaje no supervisado, donde la meta es la detección de patrones. La construcción de un modelo supervisado involucra un conjunto de entrenamiento, con los datos previamente clasificados, de los cuales se obtienen clasificadores del aprendizaje supervisado. En el proceso de entrenamiento, el clasificador *aprende* la lógica para hacer una predicción. Mediante esta lógica se clasifican nuevos ejemplos del conjunto de prueba [WFH11].

- **Minería de datos no supervisada:** El aprendizaje no supervisado es no dirigido, en este caso no hay distinción entre atributos dependientes e independientes. Es decir, no hay un resultado previamente conocido o un conjunto de entrenamiento categorizado que guíe al algoritmo en la construcción del modelo. Por lo tanto, la minería de datos no supervisada puede ser usada para propósitos descriptivos. Aunque también puede ser usada para hacer predicciones[HORQFR04].

Otro de los aspectos importantes en la minería de datos es la identificación del tipo de BD que se vaya utilizar, ya que esto nos facilita la ejecución de fases previas en el proceso llamado Extracción de conocimiento en bases de datos [Hub92] (Knowledge Discovery in Databases - KDD). Veamos algunos ejemplos de tipos de bases de datos que existen en el mundo real:

- **BD relacionales:** La colección de relaciones se representa en unas tablas, que contienen un conjunto de atributos (variables, columnas, campos), las cuales son contenidas en tuplas, tales como casos, filas y registros. La representación final que se busca en este tipo de conjunto de datos es la presentación tabular con columnas en las que estén los posibles atributos con sus respectivos valores.

- **BD espaciales:** Algunos ejemplos de este tipo de datos son, datos geográficos, imágenes médicas, redes de transporte o tráfico, entre otras más.

- **BD documentales:** En este tipo de datos se tiene un conjunto de repositorios como documentos de texto, variables desde palabras hasta resúmenes. Este tipo de bases de datos constituyen una de las principales subcategorías dentro de las denominadas bases de datos relacionales. A diferencia de las bases de datos relacionales, estas bases de datos están diseñadas alrededor de una noción abstracta del repositorio de documentos.

- **BD multimedia:** Dentro de estas bases de datos se presentan repositorios con imágenes, audio y video. Las bases de datos multimedia están en un contexto que además de los datos se le incluye la nueva característica de la que se pueden tener variabilidad espacial y temporal.

- **La World Wide Web:** Este es uno de los repositorios de información más grande y diverso que se encuentra en la actualidad en el mundo real, con la cual podemos tener una posibilidad de exploración basta para la fase de minería de datos, como encontrar patrones en las páginas web, estudiar los hipervínculos y URLs, así como el análisis de la navegación de un conjunto de usuarios monitoreados.

La minería de datos se encuentra relacionada con otras áreas de investigación como:

- Estadística, que es considerada como la "Madre"de la minería de datos,
- Aprendizaje maquinal en el cual la computadora aprende a partir de ejemplos,
- Reconocimiento de patrones con las tareas de clustering y clasificación,
- Sistemas de toma de decisiones con la construcción de herramientas y sistemas que asisten al directivo de alguna organización,
- Visualización de datos en la cual se pretende descubrir, intuir o entender los datos,
- Recuperación de la información en datos textuales, búsquedas por Internet y bibliotecas digitales.

En particular para este trabajo de investigación doctoral, nos interesa la relación que existe entre la minería de datos con el aprendizaje maquinal. Haciendo uso de los tipos de aprendizaje que existen, en minería de datos se definen las siguientes metas, utilizadas en la extracción de conocimiento en grandes cantidades de datos:

1. Predicción: Descubrir la manera de cómo ciertos atributos, dentro de los datos, se comportarán en el futuro.
2. Identificación: Identificar la existencia de objetos, eventos y actividades dentro de los datos.
3. Agrupamiento: Nos permite maximizar las similaridades y minimizar las diferencias entre los objetos, mediante algún criterio de agrupamiento.

4. Asociación: Las reglas de asociación intentan descubrir cuáles son las conexiones que se pueden tener entre los objetos identificados.

5. Clasificación: Mediante esta tarea podemos separar los datos de acuerdo a las clases o etiquetas que sean asignadas a cada ejemplo en los datos.

Una de las aplicaciones que tiene la minería de datos es su incorporación al proceso KDD, en cual mediante un proceso ordenado, los datos se pueden convertir en información útil para la solución de una problemática establecida. En la siguiente sección describiremos a grandes rasgos este proceso, en el que veremos la utilidad que tiene la minería de datos.

2.7. Extracción de conocimiento en bases de datos

La extracción de información útil (conocimiento), sobre grandes cantidades de datos ha originado una nueva generación de Teorías de Cómputo y herramientas que nos permitan poder explorarlos, analizarlos y finalmente obtener dicho conocimiento. Algunas de estas teorías son aplicadas en el proceso KDD.

El incremento de los datos crudos[1] en los últimos diez años, ha tomado un papel muy importante en el origen de este proceso, ya que, a mayor cantidad de datos generados en el mundo real, mayor es el desconocimiento del conocimiento inmerso que nos podría ayudar a resolver una determinada problemática.

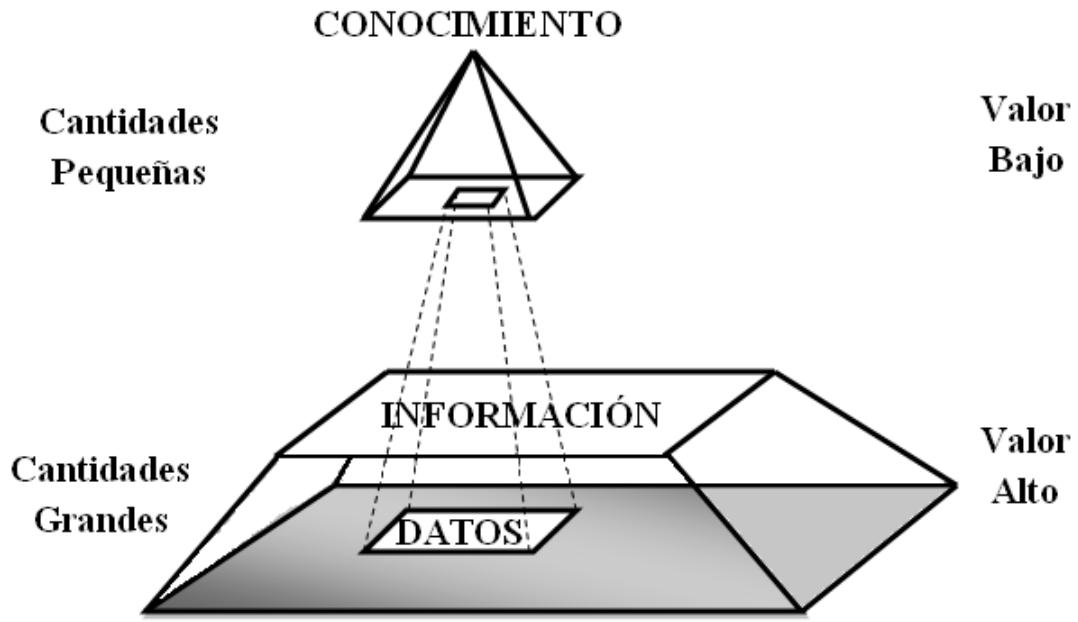
Cuando estos datos representan un significado especial para el conocimiento desconocido en el desarrollo de aplicaciones del mundo real, son consideradas información. En el momento en que los datos pasan a ser información, especialistas en el manejo de grandes cantidades de datos dan una interpretación a esta información, mediante la construcción de un modelo que represente un valor agregado a las organizaciones de negocios.

Es en este momento cuando se dice que se obtiene un conocimiento real de aquellas grandes cantidades de datos. En la Figura 2.2 se muestra un diagrama de la jerarquía que

[1]Este término se utiliza cuando llamamos a los materiales en su forma natural, por analogía con el petróleo crudo.

existe entre las principales partes que conforman un conjunto de datos (datos, información y conocimiento).

Figura 2.2: Esquema de la jerarquía del porcentaje de conocimiento que se tiene en los datos.



Los datos presentan un valor real cuando de ellos se obtiene un conocimiento, que puede ayudarnos a tomar mejores decisiones en una problemática determinada. El problema básico que aborda el proceso KDD es el de convertir los datos en información, para darles formas más compactas (por ejemplo, un informe corto), más abstractas (por ejemplo, una aproximación o un modelo descriptivo del proceso que generó los datos), y más útiles (por ejemplo, un modelo predictivo para estimar el valor de los casos futuros).

En el proceso KDD se definen una serie de metas que nos permita obtener el objetivo de transformar a los datos en información, las cuales se enlistan a continuación:

- Se debe de procesar de manera automática, grandes cantidades de datos crudos.
- Dentro de estas grandes cantidades de datos debemos identificar cuáles van a ser los patrones más significativos y relevantes.

- Una vez que se tiene identificados estos, se les da una interpretación para poderlos representar como conocimiento útil, para poder resolver la problemática que se plantea al inicio de cualquier proyecto que tenga implícito el uso de KDD.

Para poder llevar a cabo el proceso KDD, se debe de tener una guía de como poder realizar la obtención del conocimiento en los datos. El proceso KDD se realiza mediante una serie de fases, las cuales se muestran en el diagrama de la Figura 2.3:

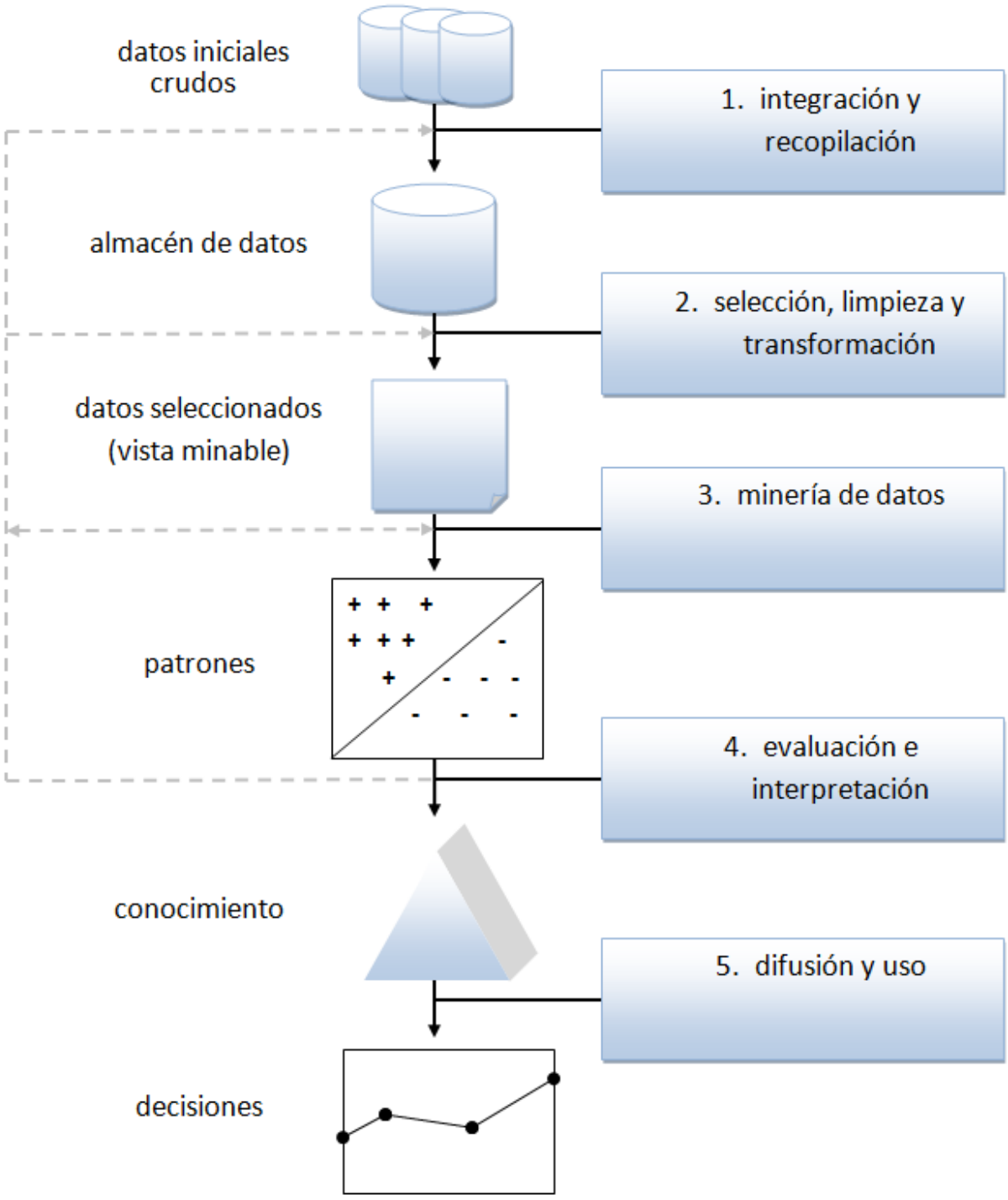
Mediante un ejemplo del mundo real mostraremos la utilidad de cada una de las fases del proceso KDD[Cor17]. Supongamos que un banco tiene la problemática de identificar fraudes sobre tarjetas de crédito, al cual se pretende darle solución mediante la aplicación del proceso KDD. En nuestro país este problema provoca pérdidas por más de 700 millones de pesos al año de acuerdo con datos del 2011 de la Asociación de Bancos de México[PMR14].

Al iniciar el proceso de KDD, los datos se encuentran dispersos y en el estado crudo como se había mencionado anteriormente. Diferentes repositorios de datos contienen información implícita sobre algún rubro, por ejemplo, información acerca de clientes que solicitan préstamos, cuentas de inversión, seguros de automóviles, aquellos clientes que realizan compras excesivas o muy diferente de sus transacciones habituales, transacciones por teléfono o Internet y operaciones autorizadas sin firma, por mencionar sólo algunas.

Empezaríamos con la primera fase que es la de integración y recopilación, sobre los diferentes repositorios con los que cuenta el banco. Cada repositorio de datos tiene información que nos puede ser útil para la solución de nuestra problemática planteada. Supongamos que para este caso nos interesan los datos del repositorio de tarjetas de crédito, transacciones por teléfono o Internet y operaciones autorizadas sin firma, por sólo mencionar algunos.

Una vez identificada la integración de múltiples datos que en la mayoría de los casos son de gran tamaño, se recomienda el almacenaje de este gran volumen de posible información en almacenes de datos (*data warehousing - DW o DWH*), con esto concluiríamos la fase de integración y recopilación del proceso KDD.

Figura 2.3: Fases del proceso KDD, e la cual se muestra la evolución de los datos crudos hasta llegar a ser patrones útiles en la problemática a resolver.



Ya realizamos la primera fase, los repositorios se analizan para identificar datos que generalicen las características esenciales en la problemática planteada. Es aquí donde en la mayoría de los casos la fase de selección, limpieza y transformación se utiliza para dar un mejor ordenamiento de los datos.

Supongamos que de los repositorios que integramos y recopilamos realizamos la selección de tres rubros en particular, aquellos datos que contengan compras excesivas, comportamientos diferentes a sus transacciones habituales de un conjunto de clientes y compras autorizadas sin firma.

Dentro de esta selección, procedemos a realizar la limpieza de algunos datos que podrían ser sobrados o que no tengan relevancia, por ejemplo, datos históricos que estén fuera de un periodo de tiempo determinado, compras autorizadas sin firma por montos menores a una cantidad establecida por el banco y aquellos datos sobre tarjetas con límites bajos de crédito.

Una vez que a los datos se les aplicó una limpieza y selección, los datos aún tienen que tener un formato adecuado para poder aplicar los métodos de minería de datos, es aquí donde se aplica la tarea de transformación. Dentro de las herramientas que nos permite aplicar minería de datos, se utiliza un formato especial para el manejo de los datos, por ejemplo, algunas manejan el formato relacional de SQL o DB2, mientras que otras tienen su propio formato, como los micro arreglos genéticos que manejan el formato ISCN (*International System for Human Cytogenetic Nomenclature*).

Esta es la razón por lo que se debe decidir primero qué herramientas se va utilizar en la fase de minería de datos, para que así sepamos de antemano qué formato deben tener los datos que ocupemos para aplicar el proceso de KDD. Una vez que se tiene establecido la herramienta que se va utilizar, se aplica la tarea de transformación para dejar en un formato necesario a los datos que serán utilizados. Para realizar esta tarea existe una serie de métodos definidos, como la discretización y la numerización.

La numerización es una técnica de la fase de Preprocesamiento, la cual corresponde al proceso inverso que se realiza en la discretización, la cual explicaremos más adelante. Un primer caso obvio donde la numerización es útil es cuando el método de minería de datos que

vamos a utilizar no admite datos nominales.

Para la mayoría de los casos se utiliza un tipo en especial de numerización, la cual es llamada *Numerización 1 a n*, con la cual se pueden hacer la creación de varias variables indicadoras, flag o dummy. Si tenemos por ejemplo el caso de una variable nominal x y además se cuenta con diferentes valores denotados por a_l, a_2, ... , a_n, se realizara la creación de n variables numéricas, tomando por ejemplo valores 0 o 1 dependiendo de si la variable nominal toma ese valor o no.

Se tienen otro tipo de casos en donde es posible identificar un cierto orden o magnitud que presentan los valores de cierto atributo nominal, es aquí donde se puede realizar otro tipo de numerización llamada *Numerización 1 a 1*. Si por ejemplo se nos presentan datos de géneros de música, tendríamos categorías del estilo alternativo, rock, pop, electrónica, clásica, podemos numerar los valores de 0 a 4, por ejemplo.

Se pueden presentar casos en los que interesa numerizar cuando se manejan atributos nominales, para el cual se puede discernir un orden que llevan los datos. Por ejemplo, supongamos que en una encuesta del gobierno se tienen recabados los niveles de estudios de un conjunto de individuos, de los cuales se tienen los siguientes valores posibles sin estudios, primaria, secundaria, preparatoria, universidad y posgrado. En este caso se puede observar que sería conveniente conservar la jerarquía de los grados académicos, en donde asignaríamos un 0 al valor sin estudios, 1 al valor primaria y así sucesivamente, hasta obtener una escala con todos los posibles valores, la cual podría ser 0, 1, 2, 3, 4, 5. En cualquiera de los casos que se presentaron, esto puede ser útil para obtener modelos más precisos y generales, agilizando de esta manera la implantación de los métodos de Minería de Datos que se elijan.

La discretización corresponde a una de las técnicas más utilizadas en la fase de transformación del proceso KDD, la cual consiste en realizar una transformación de un valor numérico en un valor nominal ordenado, esto se realiza cuando tenemos un error en la medida grande, esto es, existen ciertos umbrales significativos. El proceso de discretización se lleva a cabo cuando por ejemplo, existen diferencias en ciertas zonas del rango de valores sean más importantes que en otras o, dicho más técnicamente, cuando la interpretación de la medida no

sea lineal.

Otra de las razones de porque se lleva a cabo la técnica de discretización es cuando tenemos algunos atributos nominales y otros numéricos, y queremos que todos sean nominales para, en dado caso establecer algún tipo de reglas de asociación. En el caso en el que se desconoce el tipo de atributo que queremos discretizar o queremos realizar la discretización de muchos atributos, podemos utilizar de técnicas que nos indicaran principalmente dónde hay que separar los intervalos y cuántos hay que obtener, ya que existen infinitas posibilidades de cómo realizarlo.

Existe otra forma de discretizar los datos se basa en entropía, en la cual se utilizan la información existente de la clase que existe en los datos, encontrando así la entropía o contenido de información. De forma natural, se encuentra la mejor partición de tal forma que la mayoría de los valores en una división corresponden a la misma clase. En un método de discretización global, supervisado y estático.

Una vez que se aplica la primera y segunda fase del proceso KDD, se procede con la fase de minería de datos, en la cual se considera que el proceso realmente ha empezado, ya que la parte medular se encuentra en esta fase. Es aquí donde se aplican métodos de extracción de patrones sobre los datos que previamente se manipularon hasta llegar a esta fase.

Retomando el ejemplo del uso fraudulento de tarjetas de crédito, pensemos que podemos darle una solución parcial a la problemática planteada en este ejemplo, utilizamos la tarea de clasificación. Mediante la clasificación podemos categorizar aquellas compras en Internet o por teléfono que se salen del comportamiento habitual de un conjunto de clientes y compras excesivas autorizadas sin firmas, para simplificar el abanico de los posibles escenarios que se presentarían en el ejemplo planteado.

Mediante la clasificación podemos separar los escenarios de usos fraudulentos de tarjetas de crédito en diferentes categorías o clases, por ejemplo, uso fraudulento de tarjetas de crédito, tentativa de uso fraudulento y uso normal de tarjeta de crédito. Estos nos ayudarán a separar los datos en clases, las cuales tendrán ciertos valores de atributos específicos para un conjunto de clientes.

En este momento se dice que se tienen dos conjuntos de datos, el conjunto de entrenamiento que contiene un conjunto de registros o ejemplos de los datos, con la clase a la que pertenecen. También se tiene un conjunto de prueba, de los cuales desconocemos la posible clase que se asocie a ellos en un periodo de tiempo.

Finalmente tendríamos dos formas de aplicar la fase de difusión y uso al ejemplo que hemos planteado, por una parte, se le podría dar solución a un caso en el que un usuario en particular realice una denuncia de robo o clonación de su tarjeta de crédito. Por otra parte, el banco podría realizar un monitoreo periódico de un conjunto de clientes, para evitar la similitud de este tipo de comportamientos, en su cuenta de crédito asociada.

Abordaremos la primera utilidad de difusión y uso que se planteó anteriormente, en la que el banco puede ofrecer una investigación por una denuncia realizada por un cliente en particular. Aquí se revisarían los repositorios que se obtuvieron en las fases anteriores, mediante los cuales podríamos clasificar cuales operaciones bancarias dentro de un periodo de tiempo antes del reporte de robo se suscitaron, para darle al cliente un dictamen y su caso un rembolso de aquellas operaciones que fueron fraudulentas.

La segunda utilidad es más compleja, ya que supongamos que el banco quiere hacer un monitoreo en una delegación del Distrito Federal, en particular la Delegación Iztapalapa, con 1,815,786 habitantes, según el censo de población y vivienda realizado por el INEGI en el año 2010. Supongamos que el 30 % de la población total de esta delegación tiene una tarjeta de crédito con el banco en cuestión, esto sería más de 544,000 clientes. Ahora imaginemos que cada cliente realiza al mes 20 operaciones con su tarjeta de crédito, en total tendríamos más de 10 millones de posibles operaciones fraudulentas o no.

Podemos ver que ahora tenemos un problema de clasificación sobre grandes cantidades de datos, en el que posiblemente la respuesta que ofrezca un sector del banco, encargado del monitoreo de casos fraudulentos en tarjeta de crédito, sea en cuestión de minutos u horas. Mediante este pequeño problema que planteamos para describir el proceso KDD, visualizamos que dentro de la clasificación sobre grandes cantidades de datos está implícita una problemática, con diversos factores como la del tiempo de ejecución. En la siguiente sección

describiremos la problemática implícita al realizar clasificación sobre grandes cantidades de datos.

2.8. Problemática en clasificación sobre grandes cantidades de datos

Una de las principales tareas dentro de Minería de Datos es la de clasificación, la cual es usada para predecir las clases de cada uno de los ejemplos de los datos, esta tarea puede ser llevada a cabo mediante el uso de diversos tipos de clasificadores del aprendizaje maquinal [MC14, MMMR14, MMMR13, PG09, ST10, ZXMO06].

Cuando se realiza clasificación con un número de datos no mayor a cincuenta mil registros, se considera que es una tarea sencilla, en la cual el número de operaciones y los recursos de cómputo utilizados no presentan mayor problema. Cuando se presentan datos con un número mayor de registros, la complejidad de algunos clasificadores se incrementa gradualmente, teniendo como resultados altos tiempos de ejecución y bajos índices en las medidas de rendimiento.

En general cuando se realiza la clasificación de datos existe la presencia de tres elementos en particular, que representan los principales problemas al realizar clasificación sobre grandes cantidades de datos, los cuales se describirán en las siguientes secciones.

2.8.1. Manejo de grandes cantidades de datos

En la mayoría de los problemas reales de clasificación (empresas, desarrollo de juegos, proyectos de ciencias y tecnologías de la información, proyectos espaciales, etc.), se presentan grandes colecciones de datos. En los cuales se busca la extracción del conocimiento inmerso en estos. Cuando se tienen grandes cantidades de datos, solamente algunos algoritmos de clasificación son capaces de realizar la clasificación de manera directa.

Para mostrar un ejemplo de un repositorio con grandes cantidades de datos, presentaremos un estudio sobre Cáncer de Laringe realizando en el área de Oncología del Centro Médico

Nacional Siglo XXI, del Instituto Mexicano del Seguro Social (IMSS)[PBV+10].

Este repositorio consta de una BD en un formato especial llamado ISCN (An International System for Human Cytogenetic Nomenclature), el cual corresponde a toda la nomenclatura que se maneja en la área de investigación de Genética Humana, para el caso en específico de las BD de datos de biochips genéticos estas nomenclaturas se reducen a un subconjunto que contiene la información de posibles alteraciones genéticas asociadas con algún tipo de cáncer. Este repositorio contiene la información de las posibles alteraciones genéticas asociadas con algún tipo de cáncer, en particular con el Cáncer Cervicouterino.

Cabe destacar que esta BD representa un repositorio complejo, ya que solamente especialistas en el tema conocen esta nomenclatura, lo que hace que su interpretación y uso sea muy restringido por este tipo de especialistas ya que contiene información real a nivel cromosómico de pacientes con padecimientos de cáncer.

Esta BD representa un estudio realizado en un periodo de 10 años sobre las alteraciones genéticas de un conjunto de pacientes con Cáncer de Laringe. En el cual se tomaron muestras periódicas, esto es, cada mes en los biochips genéticos, se recopilaban las posibles alteraciones genéticas de un conjunto de pacientes, de entre los 25000 genes que tiene el ser humano.

Por medio de este estudio se han obtenido resultados muy importantes, en especial los encontrados por el Dr. Mauricio Salcedo Vargas et al.[PBV+10], con los cuales se han descubierto relaciones entre los patrones de los biochips genéticos que marca la posible predicción de cáncer cervicouterino. Teniendo como resultado un avance internacional en el ramo de la medicina, ya que con estas predicciones se puede ofrecer un tratamiento oportuno a pacientes con riesgo de desarrollar un tipo de cáncer.

Este análisis es muy valioso, ya que esta BD tiene un gran tamaño y existen muy pocas en el mundo, ya que cuenta con los registros de 50 pacientes en un periodo de monitoreo de 10 años, manejando tamaños de entre 300 mil a 7 millones de registros.

El problema que se plantea por los especialistas en el área de genética, consiste en determinar cual podría ser la condición de vida de un paciente con predisposición a padecer este tipo de cáncer. Podemos ver que tenemos un problema predictivo en el que se involucra por

una parte un gran conjunto de datos previamente categorizados y con características bien definidas, mientras que por otra parte se tienen nuevos pacientes que pueden presentar características similares a los pacientes del estudio realizado en los 10 años. Se puede pensar en crear un modelo predictivo que utilice la información previa y ofrezca una serie de patrones genéticos a los expertos para decidir qué tipo de expectativa de vida pueden tener los nuevos pacientes e inclusive que tipo de tratamiento es el más adecuado. Una forma de resolver este problema es mediante la tarea de clasificación, las cual, con base al conjunto de BD de biochips genéticos, se podría construir un clasificador capaz de asignar la clase adecuada a cada paciente nuevo monitoreado de acuerdo a su clase, que en este caso sería su expectativa de vida.

2.8.2. Altos tiempos de ejecución

Cuando se presentan grandes cantidades de datos, las operaciones de punto flotante realizadas para un conjunto de clasificadores suelen ser altas, para poner un ejemplo de este problema vamos a considerar un clasificador basado en casos llamado k-vecinos más cercanos[ST10].

El funcionamiento de este tipo de clasificadores se basa en la representación de los datos en un plano de n dimensiones. Dentro de este plano se encontraran todos los ejemplos de entrenamiento de los datos utilizada con sus clases correspondientes, de manera que al presentarse un ejemplo de prueba, se seleccionan los k-vecinos más cercanos a este nuevo ejemplo para determinar a qué clase pertenece. El concepto de similitud de un punto de prueba con respecto a los puntos de entrenamiento, se maneja mediante el uso de alguna distancia métrica, que en la mayoría de los casos es la distancia euclidiana[ST10].

El número de operaciones que se deben de realizar para este clasificador es proporcional a los tamaños de los conjuntos de entrenamiento y de prueba. Vamos a suponer que el tamaño del conjunto de entrenamiento es n y el tamaño del conjunto de prueba es m. Por cada uno de los datos de prueba se deben de calcular n distancias, los cuales son operaciones vectoriales que dependen de la estructura de base de datos que se esté utilizando, por lo tanto, en promedio se realizaran $m \times n$ operaciones, para todo el conjunto de entrenamiento.

Poniendo cifras a estos conjuntos pensando en el manejo de grandes cantidades de datos, por un lado, supongamos que tenemos un conjunto de prueba con un valor de $n = 100,000$. Por otro lado, supongamos que el conjunto de prueba tiene un valor de $m = 1,000,000$. En total el número de operaciones sería igual a $100,000,000,000$ lo que representa un alto gasto de los recursos de cómputo que impactan directamente en el tiempo de ejecución de este tipo de clasificador. Como pudimos ver en este ejemplo, el alto porcentaje de uso de recursos de cómputo (número de operaciones), impactan directamente en los altos tiempos de ejecución de algunos clasificadores, lo que representa el segundo problema al realizar la clasificación de datos.

2.8.3. Bajos índices en medidas de rendimiento

Adicionalmente a estos dos problemas, cuando se resuelve algún problema de clasificación de datos (sean grandes cantidades de datos o no), se debe buscar la forma con la que evaluamos dichos clasificadores. Esto lo hacemos mediante el uso de medidas de rendimiento usadas en minería de datos, tales como exactitud, precisión, mejora y memoria [MMMR13]. Las cuales nos permiten tener un grado de confiabilidad del correcto funcionamiento de los algoritmos de clasificación, qué en la mayoría de los casos, siempre se buscan porcentajes altos en estos índices. Por lo tanto, estos índices representan otro aspecto importante al que nos enfrentamos al realizar la tarea de clasificación.

Capítulo 3

Estado del Arte

3.1. Introducción

En este capítulo comenzaremos revisando algunos conceptos importantes de cómputo paralelo, así como su importancia en el uso de aplicaciones que demandan mayor poder de cómputo.

Continuaremos el capítulo con la revisión a detalle de los algoritmos clasificadores basados en ensambles que existen en la literatura. Para el algoritmo de Bagging y Boosting, presentaremos el esquema secuencial y paralelo en ambos casos. En esta revisión mostraremos los dos algoritmos de clasificadores basados en ensambles restantes, los cuales son Generalización Apilada y Mezcla de Expertos.

Para estos algoritmos en la literatura no se encuentran trabajos en los que se utilice el cómputo paralelo para obtener una ventaja con respecto al esquema secuencial. Por esto, en el caso de generalización apilada mostraremos una aplicación del mundo real en el cual se utiliza este método. En el caso de mezcla de expertos, revisaremos el funcionamiento de un algoritmo que desarrollamos en un proyecto de investigación previo, ya que en la literatura existen pocos trabajos en los que utilicen este algoritmo para la construcción de un ensamble.

También dentro de este capítulo veremos dos ejemplos de esquemas paralelos de clasificadores tradicionales del aprendizaje maquinal, los cuales representaron una porción de la revisión de los posibles clasificadores de base que seleccionamos para la construcción del PCEM (*Parallel Classification System based on an Ensemble of Mixture of Experts*).

Finalmente, se mostrarán los trabajos que fueron seleccionados para incorporarlos en la construcción del PCEM, describiendo de forma resumida su funcionamiento ya que los retomaremos en el siguiente capítulo, si fueron utilizados de manera directa o si realizamos modificaciones al respecto y también señalaremos aquellos clasificadores de los cuales no encontramos esquemas paralelos.

3.2. Cómputo paralelo

Conforme surgen nuevas aplicaciones el requerimiento de los recursos de los sistemas computacionales aumentan. Esto se debe a que la gran mayoría de aplicaciones ejecutan un gran número de iteraciones al momento de verificar los resultados o resolver problemas determinados. Estas aplicaciones por lo general tienen un gran número de operaciones de punto flotante que consumen los principales recursos de cómputo como son la memoria RAM y CPU. Una de las alternativas que tenemos para asignar una mayor cantidad de recursos de cómputo que demandan para nuestras aplicaciones es el cómputo paralelo. La principal ventaja de aplicar cómputo paralelo es que se pueden ejecutar varias tareas (programas) de manera simultánea para reducir los problemas de tiempos de ejecución, obteniendo soluciones que con un equipo de cómputo tradicional no se podrían encontrar en un tiempo razonable.

Retomando el ejemplo de la búsqueda de fraudes en tarjetas de crédito, en la parte final de la sección 1.4, planteamos el uso de clasificación de grandes cantidades de datos, sobre una porción de la población de la delegación Iztapalapa. Es aquí donde identificamos que podríamos tener más de 10 millones de posibles operaciones fraudulentas o no, en una sola delegación del Distrito Federal.

En su momento veíamos que este problema representaría una gran demanda de cómputo por la pronta respuesta que el banco exigiera para esta aplicación. Supongamos ahora que el problema va más allá de una sola delegación, en este caso ahora se pretende realizar el monitoreo de todo el Distrito Federal con sus 8.9 millones de habitantes. Siguiendo con el problema planteado, supongamos ahora que el 20 % del total de habitantes cuenta con una tarjeta de crédito del banco en cuestión, esto equivaldría a 1.9 millones de habitantes. Si para

cada habitante se realiza al mes 20 operaciones bancarias, tendríamos un repositorio de 35.9 millones de posibles operaciones, con las posibilidades de que algunas sean fraudulentas.

Después de un proceso de KDD, supongamos que el banco categoriza los datos en dos clases, aquellas operaciones que fueron fraudulentas y aquellas que no lo son en un determinado mes. Ahora pensemos que a partir de esta recopilación de datos el banco realiza la clasificación de las nuevas operaciones bancarias de un conjunto de clientes. Podemos ver que en este problema claramente se exige una gran demanda de los recursos de cómputo.

La ejecución de esta aplicación en una computadora comercial no sería suficiente para darle solución a está problemática, ya que podríamos tener desbordamiento de memoria RAM por el manejo de grandes cantidades de datos y un gran número de operaciones que propiciarían tener altos porcentajes de utilización del CPU. Una posible alternativa para darle solución a esta problemática sería aplicar cómputo paralelo, en el cual los datos serían distribuidos en un sistema de cómputo paralelo más robusto como un cluster o un grid reduciendo así los altos porcentajes de RAM necesaria. Mientras que, para la cuestión del CPU, mediante la incorporación de sistemas de cómputo paralelo, se tendrían varias computadoras ejecutándose de manera paralela para la distribución de las operaciones de la aplicación.

Vamos a revisar brevemente las principales arquitecturas de programación que se encuentran reportadas en la literatura, para las cuales nos vamos a valer de la clasificación de Flynn Michael[Fly72]. En esta clasificación se establece como se pueden agrupar el número de instrucciones así como el flujo de datos que es utilizado para el procesamiento de la información, teniendo las siguientes arquitecturas [Fly72]:

- SISD (Single Instruction and Single Data Stream). En esta arquitectura se sigue la forma de programación tradicional, en la cual una serie de instrucciones es ejecutada una a una sobre una serie de datos obtenidos de memoria.

- SIMD (Single Instruction and Multiple Data Streams). Significa que todas las unidades paralelas comparten la misma instrucción, pero la realizan en diferentes elementos de datos. Esta arquitectura se puede ejemplificar con un ejemplo del mundo real, en este caso pensemos en el deporte de remo de equipos el cual consiste en la propulsión de una

embarcación sobre el agua, con o sin timonel, mediante la fuerza muscular de varios remeros. En este caso hay varias entidades realizando una misma instrucción al mismo tiempo sobre diferentes datos que serían los remos que tendrían asignados a su cargo, esta instrucción se debe realizar de forma síncronizada ya que cualquier error en el ritmo establecido puede afectar el resultado final de la carrera de remos.

- MIMD (Multiple Instruction and Multiple Data Stream). En este caso múltiples instrucciones pueden ser ejecutadas en cada uno de los procesos presentes.

- MISD (Multiple Instruction and Single Data Stream). Este modelo es similar a MIMD, sin embargo, en este caso un conjunto de instrucciones puede ser ejecutada en un solo procesador.

El cómputo paralelo se puede aplicar en base a estas arquitecturas, ya que dependiendo de la aplicación que se esté modelando siempre se sigue el comportamiento establecido anteriormente, cualquiera que sea la arquitectura seleccionada se puede implementar sobre los dos sistemas de cómputo paralelo existentes que son: multiprocesadores y multicomputadores. Para sistemas de multiprocesadores se tienen varios procesadores ocupando el mismo espacio de memoria para la solución de un problema. En los sistemas multicomputadores se encuentran conjuntos de sistemas multiprocesadores que trabajan de forma conjunta para resolver algún problema.

De acuerdo con los dos sistemas de cómputo paralelo revisados en el párrafo anterior, existen dos modelos de programación, memoria compartida para el caso de sistemas multiprocesador y uso de paso de mensajes cuando estamos haciendo uso de sistemas multicomputador, sea cual sea el caso las herramientas de programación se clasifican dependiendo de estos dos modelos de programación, en la Tabla 3.1 se muestran algunos ejemplos de herramientas de programación [Fly72].

Dentro de este trabajo las dos herramientas de programación que utilizamos para la implementación del PCEM, fueron Pthreads Octave y MPI Octave. GNU Octave es un lenguaje de alto nivel destinado para el cálculo numérico el cual provee una interfaz sencilla, orientada

Tabla 3.1: Clasificación de herramientas de programación de acuerdo a su modelo de programación.

Modelo de Memoria Compartida	*Modelo de Paso de Mensajes*
Pthreads	PVM (Parallel Virtual Machine)
OpenMP (*Open Multi-Processing*)	MPI (Message Passing Interface)
Pthreads Octave	MPI Octave

a la línea de comandos (consola), que permite la resolución de problemas numéricos, además permite la ejecución de scripts y puede ser usado como lenguaje orientado al procesamiento por lotes.

Octave nació alrededor del año 1988, y fue concebido originalmente para ser usado en un curso de diseño de reactores químicos para los alumnos de Ingeniería Química de la Universidad de Texas y la Universidad de Wisconsin-Madison. Octave posee una gran cantidad de herramientas que permiten resolver problemas de algebra lineal, cálculo de raíces de ecuaciones no lineales, integración de funciones ordinarias, manipulación de polinomios, integración de ecuaciones diferenciales ordinarias y ecuaciones diferenciales algebraicas.

Sus funciones también se pueden extender mediante funciones definidas por el usuario escritas en el lenguaje propio de Octave o usando módulos dinámicamente cargados escritos en lenguajes como C, C ++, y Fortran entre otros.

Retomando algunos de las medidas de rendimiento paralelo que se presentaron en la sección 1.4, describiremos a más detalle cómo se formulan cada uno de estas[HJ11]:

- El costo de la aplicación paralela se formula de la siguiente forma:

$$Costo = \text{Tiempo} * \text{Número de procesadores } (c = t * n)$$

- El speed-up proporciona la mejora en la velocidad de ejecución de una tarea ejecutada en dos arquitecturas similares con diferentes recursos. Sea $O(n)$ el número total de operaciones elementales realizadas por un sistema con n procesadores, y $T(n)$ el tiempo de ejecución de la aplicación paralela. En general, $T(n) < O(n)$ si los n procesadores

realizan más de una operación por unidad de tiempo, donde $n \geq 2$. Supongamos que $T(1) = O(1)$ en un sistema mono-procesador, lo cual da como resultado que las instrucciones por ciclo sea igual a 1 (*IPC = 1*). El factor de mejora del rendimiento, speed-up, se define como:

$$S(n) = T(1)/T(n)$$

- La eficiencia de una aplicación paralela con n procesadores se define como:

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

La eficiencia más baja $E(n) \rightarrow 0$ corresponde al caso en que todo el programa se ejecuta en un único procesador de forma serie. La eficiencia máxima $E(n) = 1$, se obtiene cuando todos los procesadores están utilizándose al máximo durante todo el periodo de ejecución.

- Un sistema se dice que es escalable para un determinado rango de procesadores $[1, \ldots n]$, si la eficiencia $E(n)$ del sistema se mantiene constante y en todo momento por encima de un factor 0.5. Normalmente todos los sistemas tienen un determinado número de procesadores a partir del cual la eficiencia empieza a disminuir de forma más o menos significativa. Un sistema es más escalable que otro si este número de procesadores, a partir del cual la eficiencia disminuye, es menor que el otro.

- La redundancia en un medida de rendimiento paralela que se define como la relación $O(n)/O(1)$, que son respectivamente el número total de operaciones ejecutadas por el sistema con n procesadores dividido por el número de operaciones ejecutadas cuando utilizamos un solo procesador en la ejecución.

$$R(n) = O(n)/O(1)$$

- La utilización indica el porcentaje de recursos de cómputo (procesadores, memoria, etc.) que se utilizan durante la ejecución de un programa paralelo y se formula de la siguiente forma:

$$U(n) = R(n)E(n) = O(n)/nT(n)$$

- La calidad del paralelismo es directamente proporcional al speed-up y la eficiencia, inversamente proporcional a la redundancia, teniendo la siguiente formula:

$$Q(n) = \frac{S(n) \times E(n)}{R(n)}$$

Los resultados de algunas de estas medidas de rendimiento para evaluar una aplicación paralela con respecto a una secuencial serán analizados en el capítulo 5, ahora revisaremos la teoría que encontramos en la literatura referente a la construcción y característica de un ensamble.

3.3. Bosquejo inicial de un ensamble

En algunas áreas de investigación como la médica, financiara y social, la toma de una segunda opinión es mejor que la de una sola, mejor una tercera y mejor todavía la toma de varias opiniones a la vez. Esto se encuentra de manera intrínseca en las experiencias de la vida cotidiana, por ejemplo, pedir opiniones diferentes antes de someterse a una cirugía mayor, leer las reseñas de usuarios antes de comprar un producto determinado, o realizar una reservación en algún hotel con base a los comentarios de huéspedes pasados.

En cada caso, el objetivo principal es mejorar nuestra confianza de que estamos tomando una decisión correcta, combinando varias opciones mediante algún proceso de pensamiento hasta llegar a una decisión final. ¿Por qué no, entonces, seguir el mismo proceso, y pedir la opinión de otros expertos en el análisis de datos y automatización de aplicaciones para hacer toma de decisiones?

Con este razonamiento surgen un tipo especial de clasificadores del aprendizaje maquinal llamados clasificadores basados en ensambles, los cuales para más simplicidad vamos a referirnos simplemente como ensambles. En los ensambles se plantea como hipótesis inicial, el uso de un conjunto de expertos para la construcción de un clasificador que puedan aportarnos

beneficios en los principales índices de rendimiento, con respecto a los esquemas tradicionales de clasificación.

Los términos de expertos, clasificador y la hipótesis se relacionan el uno con el otro de manera intrínseca [Qui93]. El objetivo de un experto es tomar una decisión a partir de la elección de una opción de un conjunto previamente definido de opciones. Este proceso se puede visualizar como un problema de clasificación, en este caso el experto (clasificador) hace una hipótesis inicial sobre la clasificación de una instancia en los datos, utilizando una de las categorías predefinidas, las cuales representan diferentes decisiones. La decisión se basa en la formación previa del clasificador utilizando un conjunto de datos de entrenamiento, para que de esta manera las decisiones correctas sean conocidas a priori. Existen varias razones teóricas y prácticas del porque usar ensambles, a continuación, veremos algunas de ellas.

Grandes cantidades de datos: En algunas aplicaciones, como las médicas, se generan grandes cantidades de datos, por ejemplo, en estudios sobre marcadores genéticos para la determinación del origen de una enfermedad se generan repositorios de datos con más de 50 millones de registros. La noción de entrenar un clasificador con esta gran cantidad de datos puede no ser la más óptima. En cambio, se puede pensar en la división de estos datos en conjuntos más pequeños, los cuales a su vez serían utilizados por un conjunto de clasificadores mediante un criterio de combinación establecido para generar un mayor beneficio.

Tamaño de datos pequeño: Este problema es exactamente opuesto al anterior ya que para este caso se cuenta con una cantidad de datos pequeña. Uno de los factores de suma importancia en un algoritmo de clasificación es el conjunto de entrenamiento, mediante el cual el clasificador aprende con éxito la distribución de los datos. Ante la falta de una información adecuada en escasos datos, se exploran las opciones de considerar técnicas de remuestreo para la generación de subconjuntos aleatorios superpuestos de los datos disponibles. Estos datos nuevos, se pueden utilizar para entrenar diferentes clasificadores creando ensambles que han demostrado ser muy eficaces para este tamaño de datos [MMMM13].

Razones estadísticas: Como lo habíamos presentado en la sección 1.4, uno de los principales problemas de cualquier clasificador del aprendizaje maquinal es el sobreajuste, el cual

sucede al momento de entrenar un clasificador mediante un determinado conjunto de datos [B.09]. Este clasificador aprenderá o definirá un espacio de búsqueda de acuerdo con los ejemplos del conjunto de entrenamiento.

En el problema del sobreajuste existen dos posibles escenarios, por un lado, si los datos de entrenamiento presentan una distribución suficientemente similar a los datos de prueba, el clasificador no tendrá problemas al momento de clasificar los datos de prueba. Por otro lado, puede existir la posibilidad de que un clasificador aprenda bien los datos de entrenamiento, los cuales puedan representar una pequeña porción del espacio total de búsqueda de los datos de prueba. Si se presentan datos que no fueron utilizados en la fase de entrenamiento, lo más probable es que el clasificador los clasifique de manera errónea.

Es aquí donde un conjunto de clasificadores con conjuntos de entrenamiento diferentes, puede presentar diferentes rendimientos al problema del sobreajuste. Inclusive si el conjunto de clasificadores pudiera generar diferentes zonas del espacio de búsqueda del conjunto de prueba, mediante la combinación promedio de las salidas de varios clasificadores, se podría reducir el riesgo de efectuar una clasificación errónea.

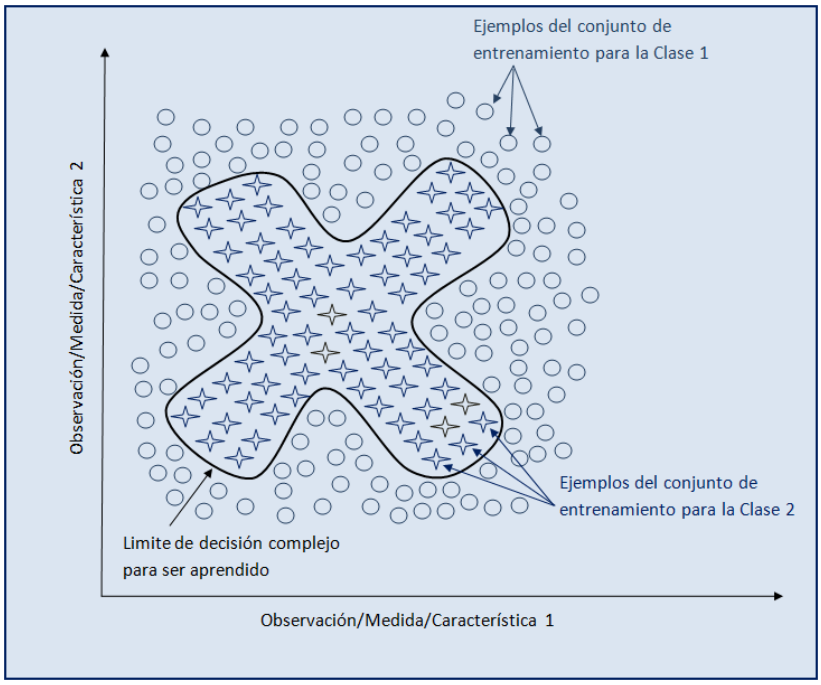
El promedio de las salidas puede o no tener mejores índices de medidas de funcionamiento al efectuar la clasificación en el ensamble, pero sin duda en algunos casos puede evitar el riesgo de tener una selección errónea en un cierto porcentaje de los datos. Esto es precisamente la razón por la que consultamos las opiniones de otros expertos: consultar varios médicos que están de acuerdo en algún diagnóstico, o varios huéspedes para saber si seleccionamos o no un hotel, nos reduce el riesgo de seguir el consejo de solamente un médico (o solamente un huésped) cuya experiencia específica puede ser significativamente diferentes a los de los demás.

Divide y vencerás: Independientemente de la cantidad de datos, algunos problemas son difíciles de resolver para un clasificador tradicional [WKRQ$^+$08]. Para ser más específico, el límite de decisión que separa los datos en las diferentes clases puede ser muy complejo, o estar fuera del espacio de búsqueda de un clasificador elegido. Para ejemplificar este caso, en la Figura 3.3 vamos a presentar un ejemplo, en el cual se consideran un conjunto de datos

representados en un plano bidimensional, con dos clases presentes en los datos, teniendo un límite o frontera de decisión complejo.

En la Figura 3.3 se puede observar que, para un clasificador de tipo lineal, tratar de clasificar los datos en las dos clases presentes, representaría una tarea difícil, por la complejidad del límite no lineal. Sin embargo, mediante una combinación apropiada de varios clasificadores lineales en un ensamble, se podría aprender este límite no lineal.

Figura 3.1: Frontera de decisión compleja entre datos pertenecientes a dos tipos de clases

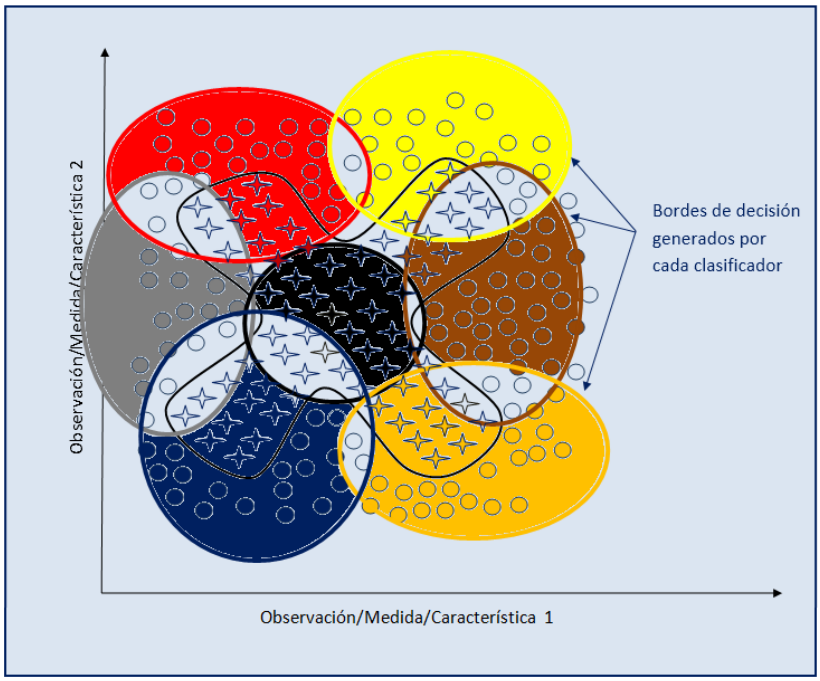


Ahora supongamos que podemos construir nuestro ensamble, con un conjunto de clasificadores que generan espacios de búsqueda elipsoidales. En la Figura 3.3, se muestra como los diferentes clasificadores generan diferentes espacios de búsqueda, en los cuales se puede observar que se tienen un número de ejemplos con las clases 0 y X. Mediante un criterio de combinación por mayoría de votos, cada clasificador podría separar a nuevos ejemplos a partir de las fronteras construidas.

En cierto sentido, la clasificación del ensamble sigue un enfoque divide y vencerás, dividiendo el espacio de datos en partes más pequeñas y más fácil de particionar, donde cada

clasificador aprende sólo una de las particiones más simples. La clasificación de un límite de decisión complejo puede ser aproximada por una combinación apropiada de diferentes clasificadores. Después de estas razones teóricas y prácticas del porque es importante el uso de los ensambles, vamos a describir cada uno de los modelos de ensamble que existen en la literatura.

Figura 3.2: Ensamble de clasificadores que abarca el espacio de decisión compleja de mejor forma que en el esquema de un solo clasificador



3.4. Bagging Secuencial

El algoritmo de Agregación Autosuficiente (***B**ootstrap **agg**regat**ing*** - *Bagging*) fue introducido por Breiman [Bre96] en la década de los 90, integrando el concepto de agregación reforzada. La implementación de este modelo es uno de los más sencillos, el ensamble consiste en escoger varios modelos de un clasificador, los cuales son combinados por un criterio de votación por mayoría. En un inicio, para este tipo de ensambles se seleccionaban diferentes

árboles de decisión, mediante la selección de diferentes conjuntos de entrenamiento generados de manera aleatoria. Los árboles de decisión cómo es sabido, dependiendo del algoritmo de construcción presentan diferentes estructuras y caminos para realizar la clasificación.

Bagging es una buena opción cuando los datos disponibles son de tamaño limitado [Bre96], en cuyo caso para asegurarse de que hay suficientes muestras de entrenamiento en cada subconjunto, se extraen porciones relativamente grandes (75 % al 100 %) del conjunto de entrenamiento para cada subconjunto.

Esto hace que los subconjuntos individuales de entrenamiento se superpongan significativamente, con muchos de los mismos casos que aparecen en la mayoría de los subconjuntos, y en algunos casos que aparezcan varias veces en un determinado subconjunto.

Con el fin de garantizar la diversidad en este escenario se utilizan clasificadores que pueden cubrir diferentes límites de decisión a partir de la selección de diferentes conjuntos de datos de entrenamiento. Como se ha mencionado anteriormente redes neuronales y árboles de decisión son buenas opciones para la construcción de bagging, sin embargo en la actualidad otros clasificadores como lógica difusa [MC14] y K-Medias [KMHH11], son utilizados como clasificadores de base, en el Algoritmo 1 se muestra el pseudocódigo para el algoritmo de bagging.

Entrada:

- Conjunto de entrenamiento S con los clases reales $w \in \Omega = [w_1, \ldots, w_n]$;
- Aprendices inestable) del algoritmo;
- Valor entero T, el cual especifica el número de iteraciones;
- Porcentaje (o fracción) F para crear el conjunto de entrenamiento autosuficiente (*bootstrapped*).

para t = *1 hasta* T **hacer**

- Tomar una réplica autosuficiente S_t mediante una selección al azar de F, la cual representa un porcentaje de S.
- Ocupar aprendices inestables con S_t y aprobar la hipótesis (clasificador de base) h_t.
- Adicionar h_t para el ensamble E.

fin

Fase de prueba: Combinación por mayoría de votos - Dada una instancia **x** sin clase

- Evaluar el ensamble $E = h_1, \ldots, h_T$ en **x**.
- Permitir $v_{t,j} = \left\{ \begin{array}{c} 1, \text{si } h_t \text{ escoge la clase } w_j \\ 0, \text{en otro caso} \end{array} \right\}$

  De esta manera se le otorga un voto para la clase w_j por el clasificador de base h_t.
- Se obtiene el total de votos recibidos para cada clase

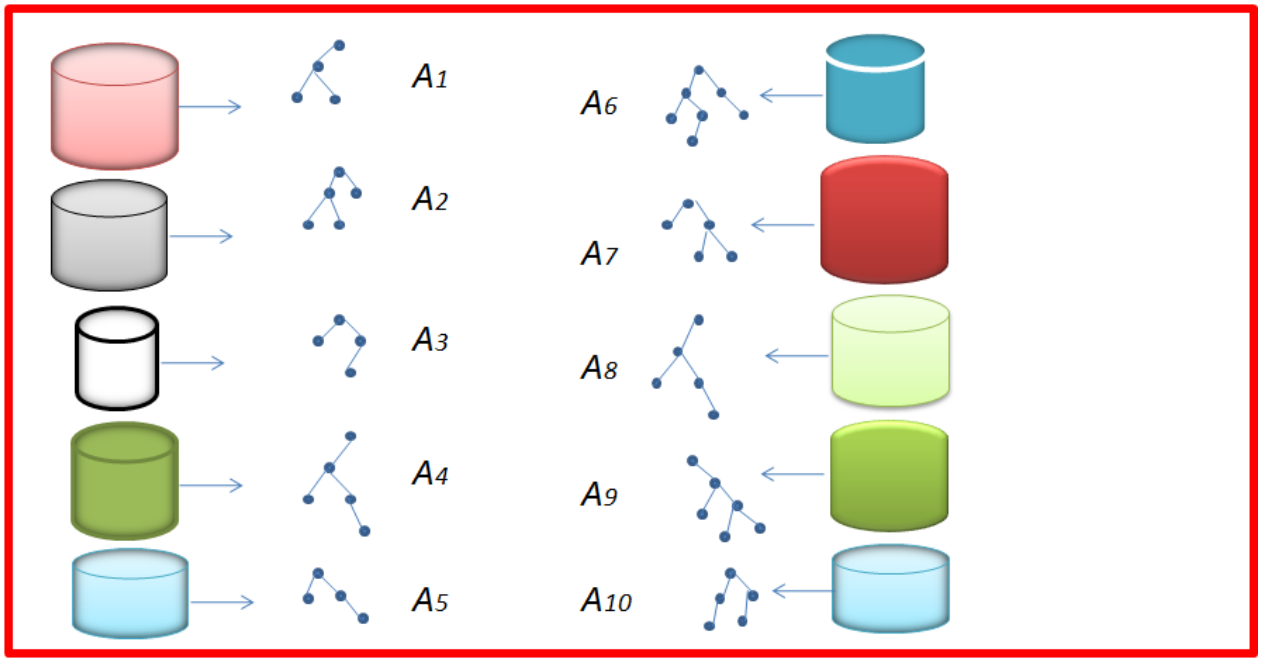
$$V_j = \sum_{t=1}^{T} v_{t,j}, j = 1, \ldots, C$$

- Escoger la clase que haya recibido la mayoría de votos para obtener la clasificación final.

Algoritmo 1: Clasificador de tipo ensamble llamado Bagging

Vamos a describir brevemente cada uno de los pasos que se realizan en este ensamble [Bre96], lo cual nos permitirá posteriormente presentar un esquema paralelo del mismo. Como primer paso se procede a dividir el conjunto de entrenamiento en subconjuntos para generar diferentes modelos de clasificación de un clasificador, en este ejemplo vamos a suponer que generamos diferentes modelos de un árbol de decisión, esta primera etapa se muestra en la Figura 3.3.

Figura 3.3: Generación de diferentes árboles de decisión.

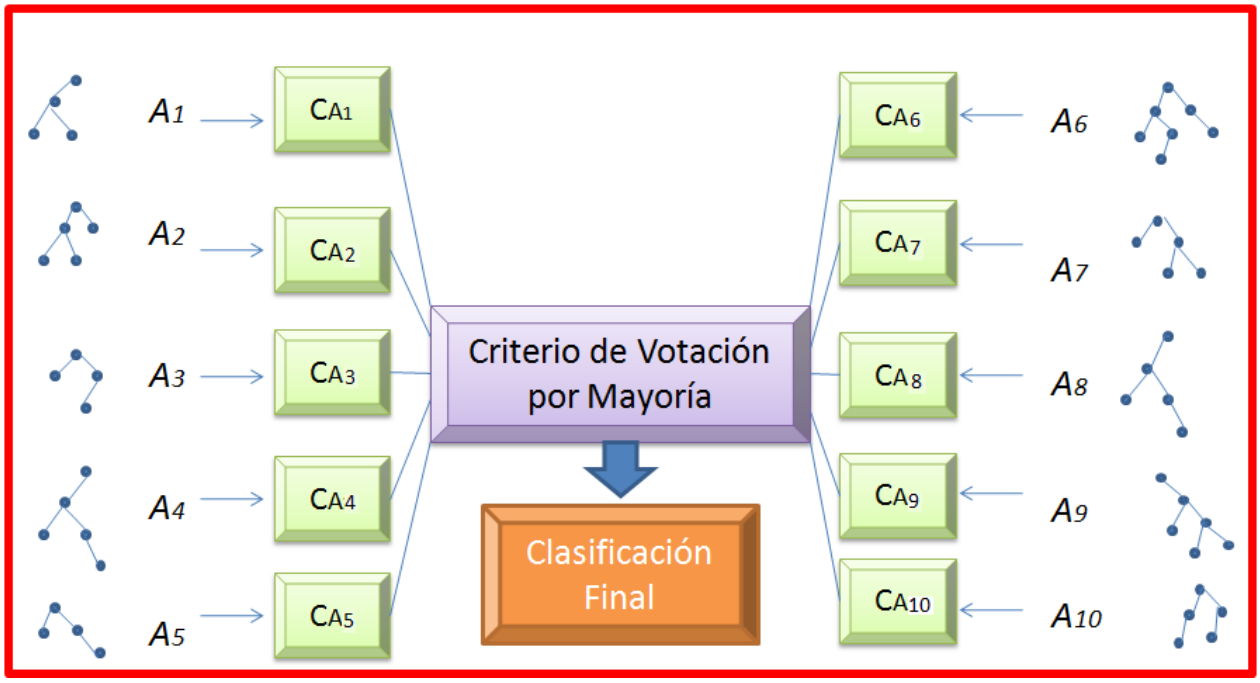


Como pudimos observar en la Figura 3.3, se generan n árboles de decisión por cada uno de los subconjuntos del conjunto de entrenamientos, ejecutados en un sólo proceso ya que se está revisando la versión secuencial. Posteriormente una vez que se tienen los diferentes modelos de clasificación, cada uno clasifica el conjunto de prueba por separado para obtener un conjunto de clasificaciones individuales.

Una vez que se obtiene la clasificación individual de la BD, son combinadas mediante el uso de un criterio de votación por mayorías para obtener las clases finales de la BD, este último paso lo podemos ver en la Figura 3.4. Ya que revisamos el esquema secuencial del

algoritmo Bagging, en la siguiente sección describiremos un esquema paralelo encontrado en la revisión del estado del arte.

Figura 3.4: Implementación del Criterio de Votación por Mayoría.



3.5. Bagging paralelo

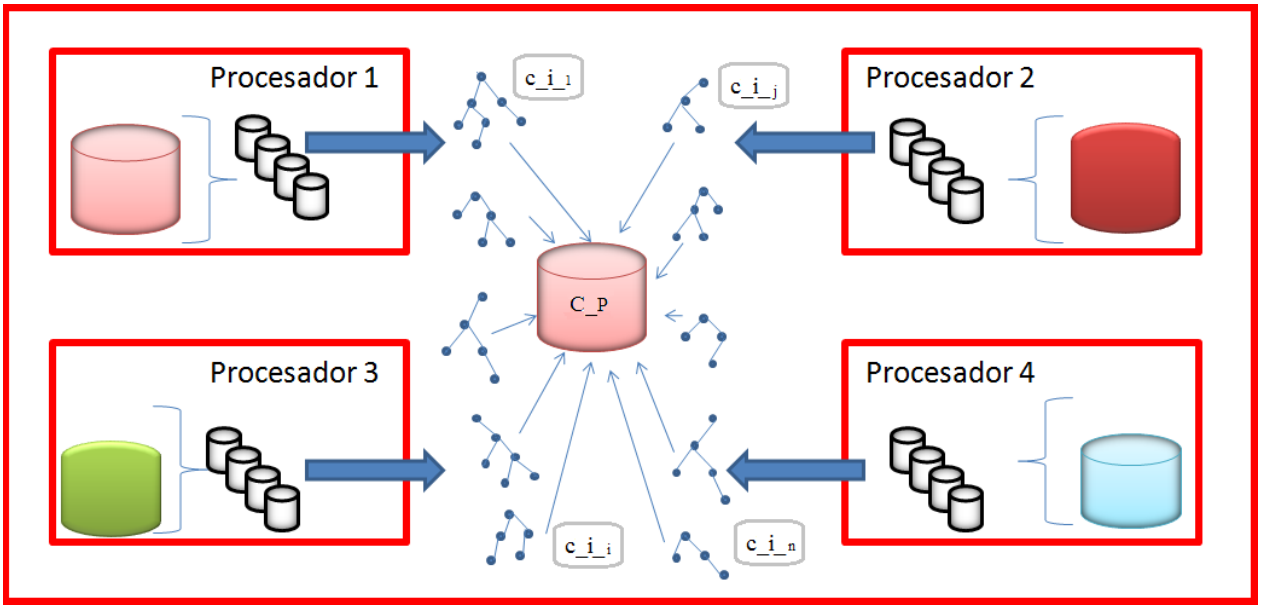
Para el esquema de funcionamiento paralelo de *Bagging* [YS01], partimos primero en la distribución de los n subconjuntos, obtenidos mediante una división del conjunto de entrenamiento. Estos subconjuntos para este esquema paralelo se distribuyen en un número determinado de procesadores, utilizando en cada uno de los procesadores un conjunto determinado de procesos.

Esto se debe a que cada proceso puede generar de manera individual un árbol de decisión diferente, siguiendo la estrategia de divide y vencerás, el cual fue mencionado en la sección 2.2. Para este ejemplo utilizaremos el caso en el que los subconjuntos se distribuyen en un conjunto de 4 procesadores el cual simula el caso en el que estemos ejecutando dicho esquema

paralelo en un sistema multiprocesador.

Ya que se tiene distribuidos los n subconjuntos, el siguiente paso en este esquema paralelo, consiste en generar un árbol de decisión por cada uno de los procesos lanzados. Una vez que se generan los árboles de decisión de cada proceso, cada árbol de manera paralela encuentra las clasificaciones individuales del conjunto de prueba, esta segunda etapa del esquema paralelo que estamos presentando del ensamble *Bagging*, se puede observar en la Figura 3.5.

Figura 3.5: Clasificación individual de los aprendices inestables.

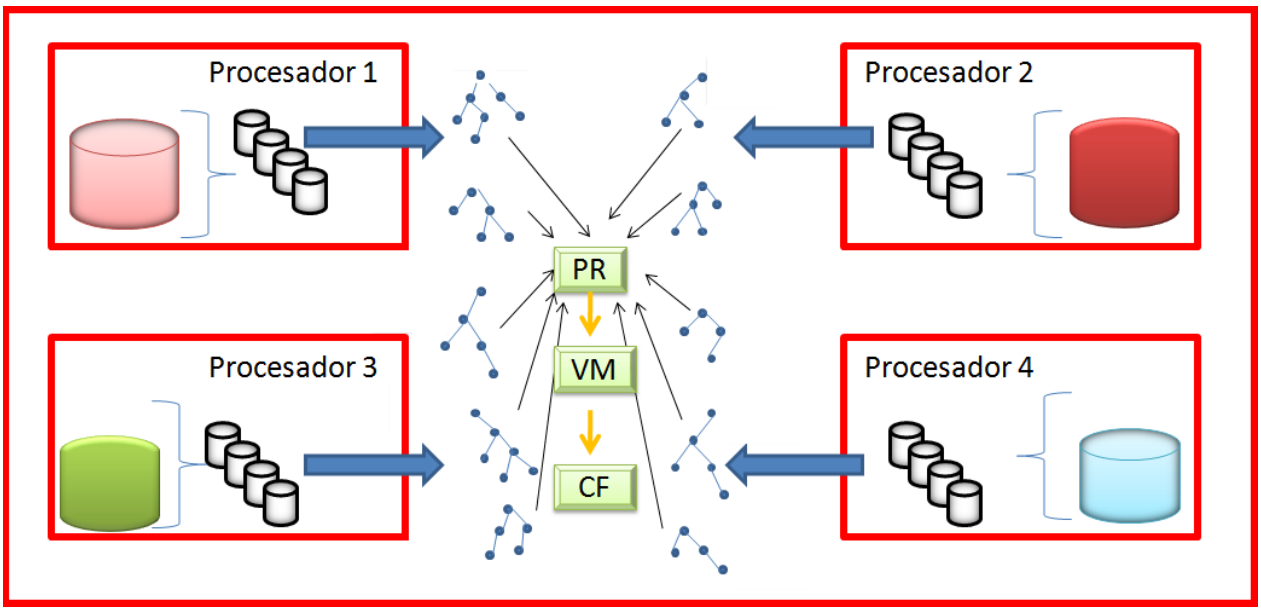


En la Figura 3.5, podemos observar que cada uno de los aprendices inestables generados por los diferentes procesos de cada procesador, realizan una clasificación individual (c_{i_k} $\forall$ $k = 1, \ldots, n$) del conjunto de prueba (C_P). Mediante este esquema de implementación se reducen los tiempos de ejecución en la generación de diferentes modelos de los árboles de decisión, sobre todo cuando el algoritmo de generación tiene una complejidad alta, además de esto se pueden generar un número mayor de árboles para tener una mayor cobertura en la distribución de los datos que se pretendan clasificar.

Finalmente, un procesador recolecta (**PR**) las clasificaciones individuales de cada uno de los árboles generados para aplicar el criterio de votación por mayoría (**VM**) para obtener la

clasificación final (**CF**) del conjunto de prueba, este último paso del esquema paralelo que presentamos del ensamble de tipo *Bagging* lo podemos ver el Figura 3.6.

Figura 3.6: Clasificación final del conjunto de prueba.



3.6. Boosting secuencial

En la década de los años 90, este sistema basado en acoplamientos fue desarrollado por estudiantes de Shapire [Sch03], los cuales probaron que si se seleccionan un conjunto de aprendices débiles, entrenándolos con diferentes conjuntos de entrenamiento y combinándolos con un criterio de combinación de clases adecuado, se pueden generar aprendices fuertes, con lo cual se obtuvo el algoritmo impulsando (*Boosting*), considerado junto con el ensamble de tipo *Bagging*, dos de los algoritmos seminales [ET93] de los clasificadores basados en ensambles.

Entrada:

- Conjunto de entrenamiento: $\mathcal{D} = \{(x_i, y_i), i = 1, \ldots, N\}$, donde $x_i \in \mathbf{R}^d$ y $y_i \in \{-1, +1\}$.
- Número de ejemplos muestreados por cada iteración: m
- Aprendices débiles: $\mathcal{L}$ que automáticamente aprende un clasificador binario $h(x) : \mathbf{R}^d \mapsto \{-1, +1\}$ de un conjunto de ejemplos de entrenamiento.
- Número de iteraciones: T

Salida:

- El clasificador final: $H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

Algoritmo

Inicilizar las distribuciones $D_0(i) = 1/N, i = 1, \ldots, N$

para $t = 1$ *hasta* T **hacer**

- Muestra m con ejemplos de remplazo $\mathcal{D}$ según la distribución $D_{t-1}(i)$.
- Entrenar un clasificador binario $h_t(x)$ utilizando los ejemplos muestreados
- Calcular la tasa de error $\varepsilon_t = \sum_{i=1}^{N} D_{t-1}(i) I(h_t(x_i) \neq y_i)$ donde $I(z)$ se le da la salida 1 cuando z es verdadera y cero en caso contrario.
- Salir del ciclo Si $\varepsilon > 0.5$.
- Calcular el peso α_t como $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
- Actualizar la distribución como $D_t(i) = \frac{1}{Z_t} D_{t-1}(i) \exp(\alpha_t I(y_i \neq h_t(x_i))$ donde $Z_t = \sum_{i=1}^{N} D_{t-1}(i) \exp(\alpha_t I(y_i \neq h_t(x_i))$.

fin

Construir el clasificador final $H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$.

Algoritmo 2: Clasificador de tipo ensamble llamado AdaBoost

Al igual que en el *bagging*, *boosting* también crea un conjunto de clasificadores por remuestreo de los datos, los cuales se combinan por un criterio de mayoría de votos. Sin embargo, las similitudes entre ambos métodos son solamente las anteriormente mencionadas, ya que en boosting, el remuestreo se orienta estratégicamente para proporcionar datos de entrenamiento más informativos para cada generación de los aprendices inestables.

En esencia, boosting crea tres aprendices inestables: el primer clasificador C_1 se entrena con un subconjunto aleatorio de los datos de entrenamiento disponibles. El subconjunto de datos de entrenamiento que se elige para el segundo clasificador C_2, se escoge del entrenamiento que se llevó a cabo por su predecesor C_1. Esto es, C_2 se entrena con un conjunto de entrenamiento que en la mitad tiene los ejemplos correctamente clasificados por C_1, y la otra mitad lo componen los ejemplos que fueron clasificados de manera incorrecta. El tercero aprendices inestables C_3, es entrenado con casos en los que C_1 y C_2 están en discrepancia. Finalmente, los tres clasificadores generados se combinan a través de un criterio de votación por mayoría de tres vías. En el Algoritmo 2 se muestra el pseudocódigo de bossting.

Schapire [Sch03] demostró que el error de bagging, constituido por los tres clasificadores, está acotado superiormente ya que es menor que el error del mejor clasificador del ensamble, siempre que cada clasificador tiene un porcentaje de error menor al 50 %. Para un problema de dos clases, un porcentaje de error de 50 % es lo menos que se puede esperar para el clasificador fuerte, generado por los tres aprendices inestables. Sin embargo, para la creación de un clasificador fuerte se puede crear mediante el uso de un número determinado de iteraciones, para que se reduzca el error esperado en el ensamble de tipo boosting.

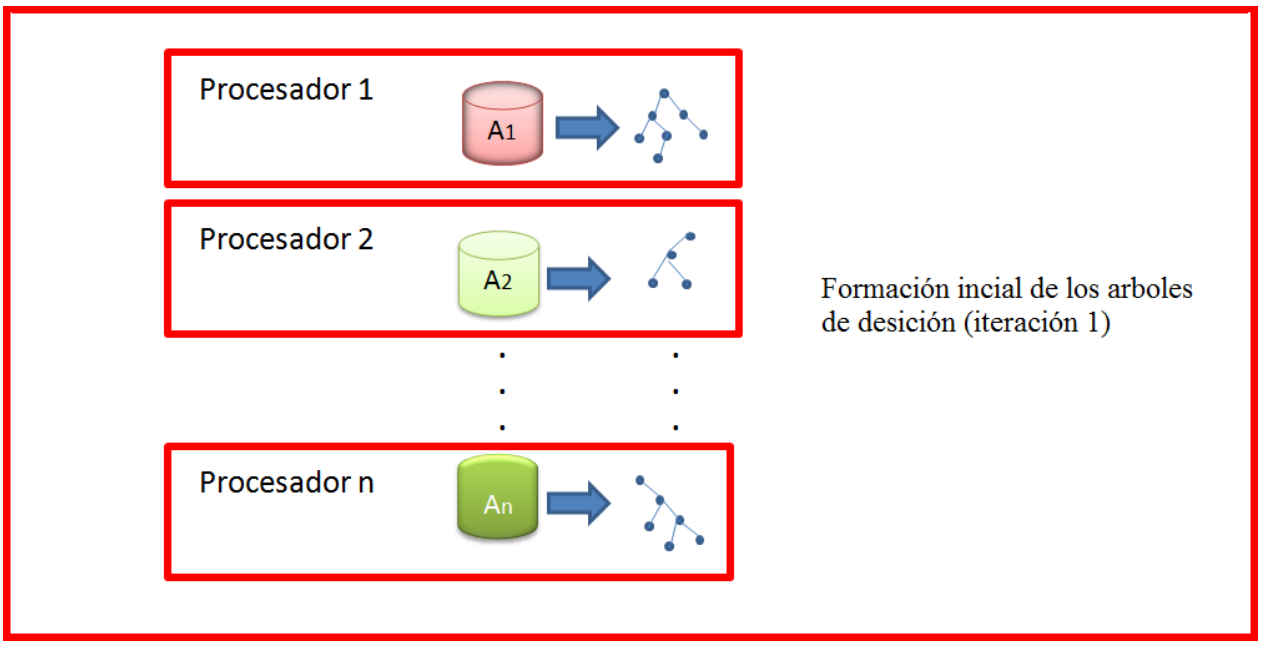
3.7. Boosting paralelo

Para el desarrollo de este esquema paralelo, lo primero que plantearon los autores en [YS01] fue la necesidad de realizar un número de iteraciones mayor para tener un entrenamiento más óptimo de los aprendices débiles, tal y como se mencionó en el final de la sección anterior. Este enfoque se debe a que no es suficiente la construcción de un aprendiz débil mediante el uso de un sólo conjunto de entrenamiento, es por esto que para este esquema

paralelo [YS01], los autores proponen el uso de computo paralelo en la construcción de un conjunto de aprendices débiles a partir de una serie de iteraciones, en las cuales la información sobre los ejemplos que difícilmente y fácilmente fueron clasificados, son recopilados por cada clasificador generado en el ensamble.

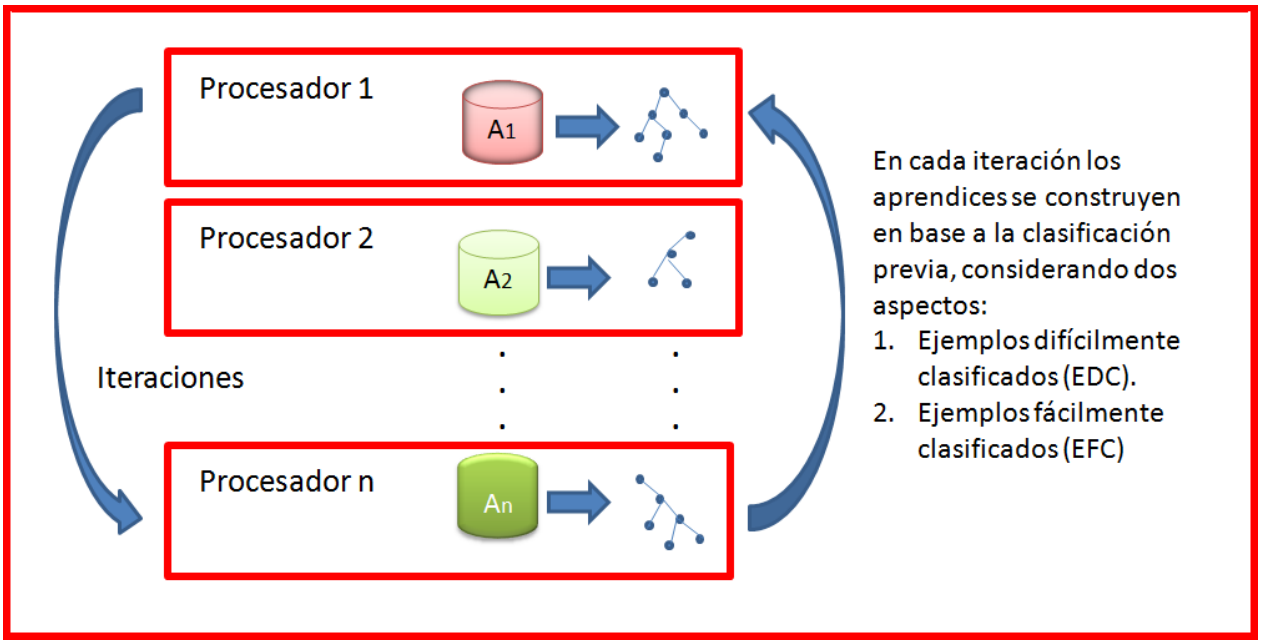
En la versión paralela de Boosting [YS01] que mostraremos, la primera fase consta del mismo procedimiento que en el esquema secuencial, con la diferencia de que la generación de cada uno de los árboles corresponde directamente al número de procesos ejecutados en un sistema multiprocesador. Es por esto que para el esquema paralelo no se generan solamente tres aprendices inestables, como en el caso secuencial, sino que el número va de acuerdo con el número de procesos lanzados al inicio de la ejecución de este esquema paralelo. Para ejemplificar esta primera fase pensemos en un sistema multiprocesador, el cual consta de n procesadores, y siguiendo un esquema equitativo, en cada procesador radica un proceso para generar un árbol de decisión, esta primera fase se muestra en la Figura 3.7.

Figura 3.7: Formación inicial de los aprendices débiles



Una vez que se genera la primera iteración los aprendices débiles generan un conjunto de ejemplos que son difíciles y fáciles de clasificar (EDC y EFC), estos ejemplos son utilizados en las iteraciones subsecuentes, formando una estructura de anillo. En el cual los EDC_{C_1} y EFC_{C_1} del primer clasificador C_1, son utilizados por el segundo clasificador C_2 y así sucesivamente hasta llegar al último clasificador generado C_n. Para el caso del primer clasificador C_1, se utilizan los EDC_{C_n} y EFC_{C_n}, del último clasificador C_n, es por esto que este esquema sigue un modelo de anillo. Esta segunda etapa de funcionamiento del esquema paralelo del ensamble de tipo *Boosting* se muestra en la Figura 3.8.

Figura 3.8: Incorporación de un número N de iteraciones

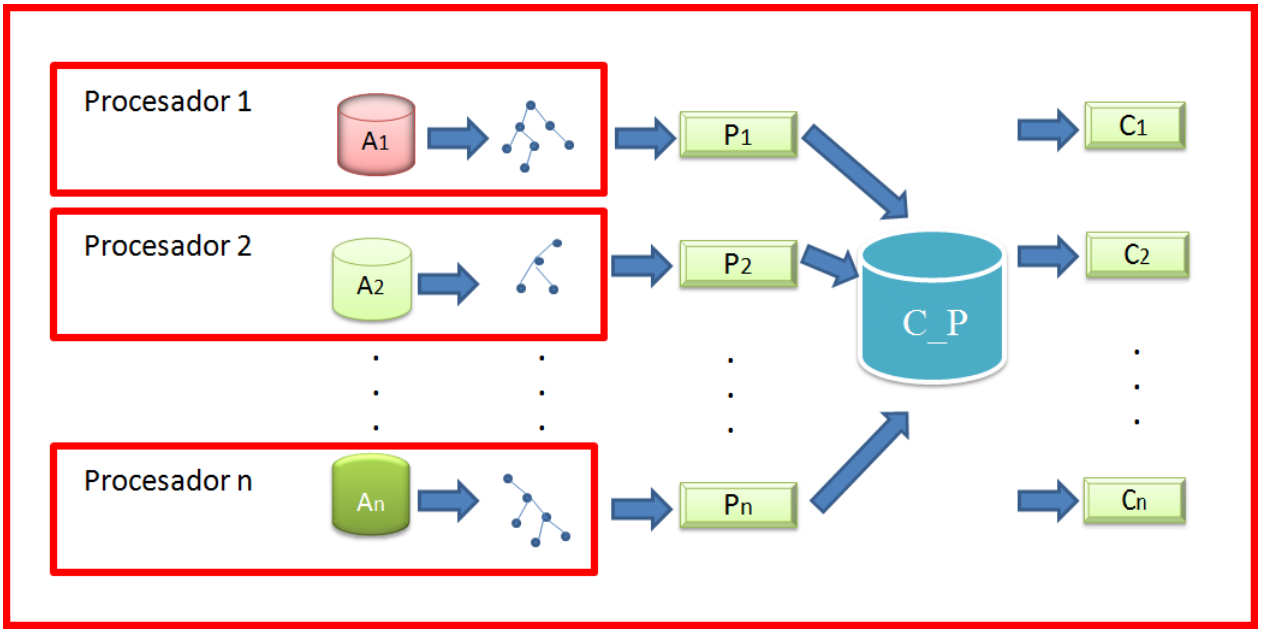


Otra de las aportaciones de este esquema paralelo, es la incorporación de un método de combinación de las clasificaciones individuales, diferente del que se utiliza en el esquema secuencial. Para este caso se utiliza un criterio de votación ponderado, como el que se presentó en el algoritmo AdaBoost [FS97], desarrollado por Freund and Schapire en el año 1997.

Para este esquema paralelo, una vez terminada la fase de entrenamiento, cada uno de los clasificadores generados reciben una ponderación, de acuerdo con las tasas de error que

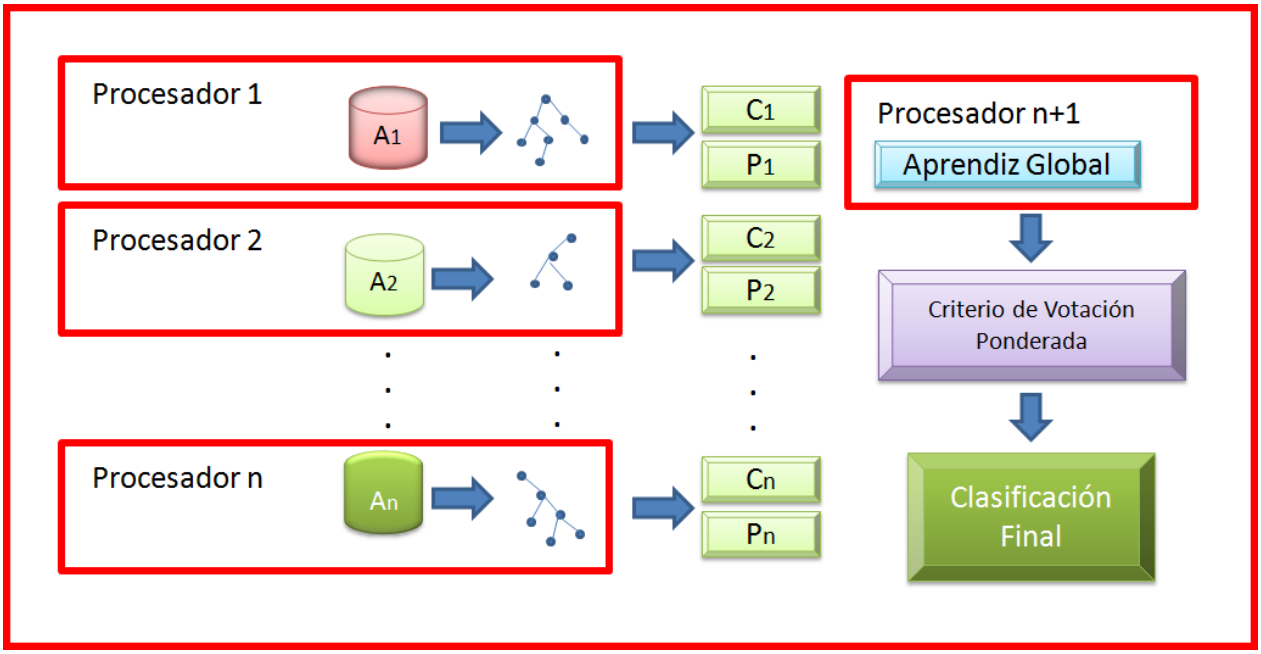
se obtuvieron previamente, de esta manera además de tener clasificadores con un mejor entrenamiento, aquellos que tienen los mejores índices en medidas de funcionamiento tienen un peso mayor sobre el resto de clasificadores generados. De manera paralela cada clasificador $C_i \ \forall \ i = 1, \ldots, n$, obtienen las clasificaciones individuales del conjunto de prueba C_P, los cuales se obtienen en base a la regresión de mínimos cuadrados parciales. Estos dos procesos se pueden visualizar en el esquema de la Figura 3.9.

Figura 3.9: Ponderación de los clasificadores y obtención de las clasificaciones individuales



Una vez que se tienen las clasificaciones individuales y las ponderaciones de cada clasificador, otro proceso seleccionado como el coordinador de este esquema paralelo, recopila esta información para aplicar el criterio de votación ponderado, en el cual se realiza una suma ponderada de votos por cada clase presente, asignando la clase de mayoría de votos a cada ejemplo, este proceso final lo podemos ver en la Figura 3.10.

Figura 3.10: Criterio de votación ponderada



3.8. Generalización apilada

En ciertos casos se puede tener una alta probabilidad de tener ejemplos mal clasificados debido a que algunos datos están muy cerca de la frontera de decisión, y por lo tanto a menudo caen en el lado equivocado de la frontera aproximada por un clasificador. Por otro lado, en ciertos casos se puede tener una alta probabilidad de ser clasificado correctamente, debido a que los datos se ubican principalmente lejos del límite de decisión y en el lado correcto de la zona de clasificación generada por un clasificador dado. Surge una pregunta natural: ¿Podemos aprender de ciertos clasificadores que constantemente clasifiquen correctamente o que constantemente clasifiquen erróneamente algunos casos? ¿O más específicamente, dado un ensamble de clasificadores que operan en un conjunto de datos extraídos de una distribución desconocida y fija, podemos comparar las salidas de estos clasificadores a sus clases verdaderas?

El ensamble de tipo generalización apilada o también llamado *Stacking*, fue introducido por Wolpert [Wol92] en el año de 1992, y su funcionamiento se basa en construir un ensamble con diferentes clasificadores, algo totalmente diferente con respecto a *Bagging* y *Boosting*, los cuales generan el ensamble a partir de un sólo clasificador. En generalización apilada de Wolpert, un conjunto de clasificadores se genera primero, cuyas salidas se utilizan como entradas de un segundo nivel, llamado el meta-clasificador, para aprender de la comparación entre las salidas del ensamble y las clases correctas del conjunto de prueba [Wol92].

En la Figura 3.11 ilustra el funcionamiento del ensamble de tipo generalización apilada, en el cual se observa que se tiene un conjunto de clasificadores $C_1, \ldots, C_T$, están entrenados utilizando parámetros de entrenamiento denotados por $\theta_1, \ldots, \theta_T$ (en este tipo de ensambles se escogen diferentes clasificadores de base, conjuntos de entrenamiento, etc.), los cuales generan un conjunto de salidas (clasificaciones individuales), denotadas por $h_1, \ldots, h_T$.

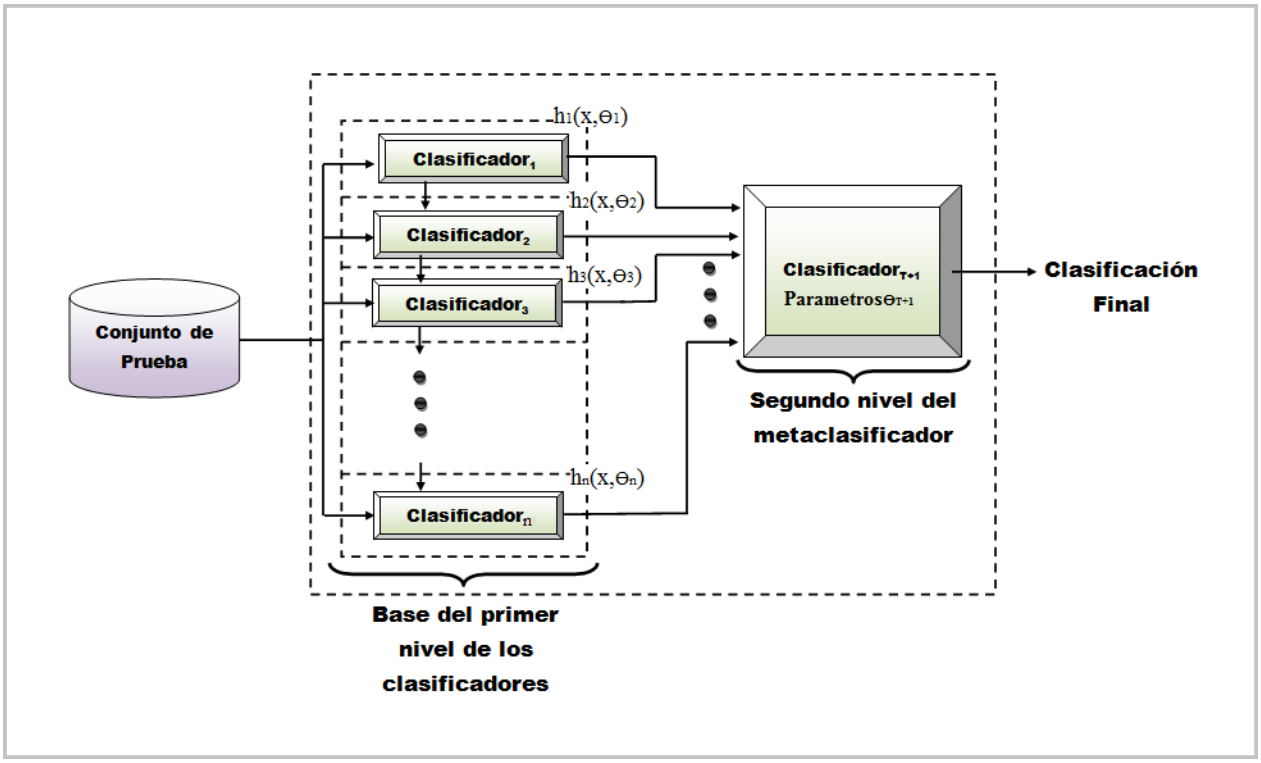
Como lo habíamos mencionado al inicio de este capítulo, para este ensamble, en la literatura no encontramos trabajos en los que se proponga un esquema paralelo para solventar algunos de los problemas implícitos dentro del esquema.

Un ejemplo de estos problemas es el tiempo de ejecución que se presenta al utilizar un conjunto de clasificadores de diferentes tipos. Dado que no se encontraron trabajos en la literatura para solventar las principales limitantes de este ensamble, en la siguiente sección mencionaremos una aplicación del mundo real en el cual se utiliza este tipo de ensamble.

3.9. Aplicación de generalización apilada

Una de las aplicaciones que tiene la tarea de clasificación está en el área de medicina, en particular vamos a presentar en este capítulo una aplicación de generalización apilada en el estudio de las interfaces computarizadas del cerebro [SZL08] (*Brain Computer Interface - BCI*). En esta aplicación se presenta una sencilla pero efectiva aproximación de un ensamble para la clasificación de señales de electroencefalograma (*electroencephalogram signal classification - EEG*), llamado Ensamble de Selección de Electrodos Aleatorios (*random electrode selection ensemble - RESE*).

Figura 3.11: Funcionamiento de Generalización Apilada



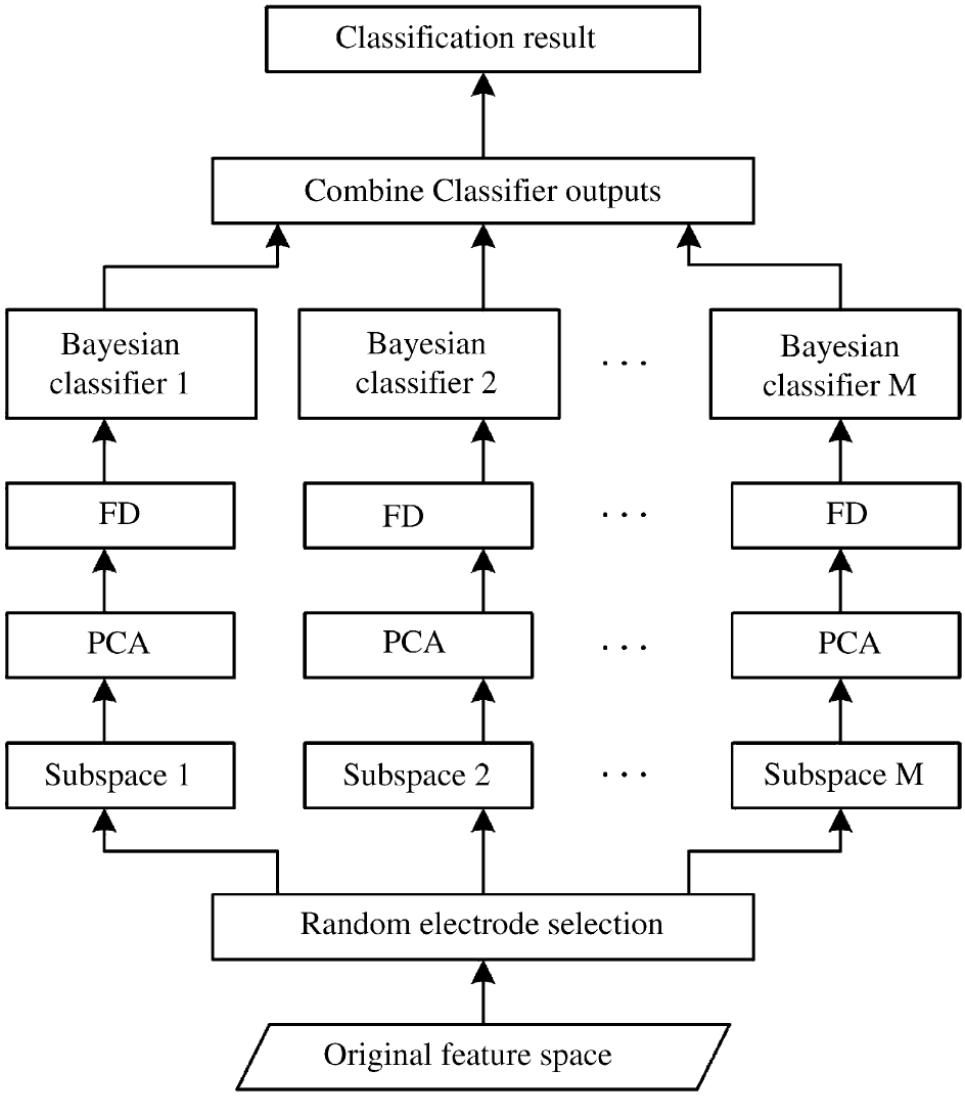
A través de una selección al azar de registros de electrodos, los autores proponen una respuesta a los antecedentes fisiológicos en las actividades mentales de un conjunto de usuarios monitoreados, mediante el uso de múltiples clasificadores individuales. En un subespació característico determinado por un par de electrodos seleccionados al azar, el análisis de componentes principales (*principal component analysis - PCA*) se utiliza de manera inicial para llevar a cabo la reducción dimensional del espacio de búsqueda.

El discriminante de Fisher (*Fisher discriminant - FD*), es un enfoque de extracción de características muy conocidos para describir conjuntos de datos multivariantes en un espacio derivado de FD. El cual es utilizado para la búsqueda de un conjunto de vectores de proyección para maximizar la relación entre y dentro de la matriz de clase de dispersión [SZL08], lo cual representa un papel importante en la investigación de BCI.

Los autores proponen la construcción de un ensamble de tipo generalización apilada,

usando diferentes modelos de un clasificador bayesiano de tipo mezcla gaussiana (*Gaussian mixture model - GMM*). En la Figura 3.12, se muestra la arquitectura del ensamble de tipo generalización apilada propuesto por Shiliang Sun, et al. [SZL08, Wol92].

Figura 3.12: Ensamble de Selección de Electrodos Aleatorios - RESE



Podemos ver en la Figura 3.12 que el ensamble inicia en la parte inferior con un espacio de características originales, las cuales pasan a través de un filtro para generar los conjuntos de cada clasificador de base seleccionados. Primero se realiza una selección aleatoria de electrodos, de los cuales se seleccionan los subespacios de búsqueda denotados por $Subspace_i \; \forall \; i = 1, \ldots, n$. A estos subespacios de búsqueda se les aplica el análisis de componentes principales (PCA) y la discriminante de Fisher [Fis36] (FD), para que se generen los clasificadores de base de tipo mezcla gaussiana ($Bayesian \; classifier_j \; \forall \; j = 1, \ldots, n$).

La eficiencia de RESE [SZL08], radica en la selección eficiente de los subconjuntos de búsqueda seleccionados para el clasificador de base en el ensamble, en el cual se tienen las fases de selección, limpieza y transformación del proceso KDD, el cual revisamos en la sección 2.6 de este trabajo.

Finalmente, cada uno de los clasificadores de base son combinados mediante un criterio de votación por mayoría. La relevancia de presentar este ejemplo es para diferenciar los dos tipos de ensambles que existen. Por un lado, existen los ensambles monoclasificadores que consideran diferentes modelos de un clasificador, de este tipo de ensambles se tienen a *Bagging* y *Boosting* los cuales fueron presentados en las secciones 3.4, 3.5, 3.6 y 3.7. Por otra parte, se tienen los ensambles multiclasificadores, en los cuales se consideran diferentes tipos de clasificadores, dentro de esta categoría de ensambles se encuentran generalización apilada o *Stacking* y un caso particular de este último llamado mezcla de expertos o también llamado por su nombre en inglés, *Mixture of Experts*.

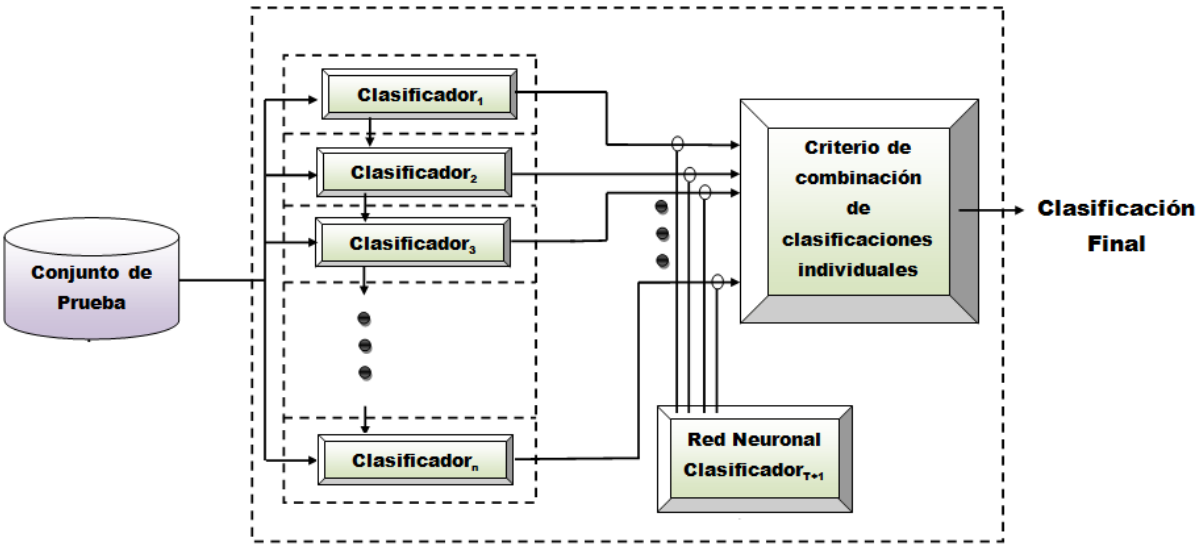
A continuación, se revisa este último modelo de construcción de un ensamble multiclasificador, posteriormente se mostrará un algoritmo que desarrollamos en un proyecto de investigación previo, que representa el esquema secuencial del PCEM.

3.10. Mezcla de expertos

El algoritmo de mezcla de expertos [Pol06] es un caso particular de generalización apilada, con la diferencia sobre la combinación de las clasificaciones individuales de cada clasificador de base, lo cual se realiza mediante un criterio de votación ponderada y no por mayoría o

voto duro con en el caso de generación apilada[MMMR14, FP98]. En mezcla de expertos se tienen dos componentes, el primero es un conjunto de clasificadores de base $C_1, \ldots, C_T$ que constituyen el ensamble, el segundo componente lo constituye un método para la asignación de los pesos, C_{T+1}, a cada clasificador de base, así como la aplicación posterior del criterio de votación ponderada, en la Figura 3.13 se muestra el esquema de funcionamiento de este ensamble.

Figura 3.13: Esquema de funcionamiento de Mezcla de expertos



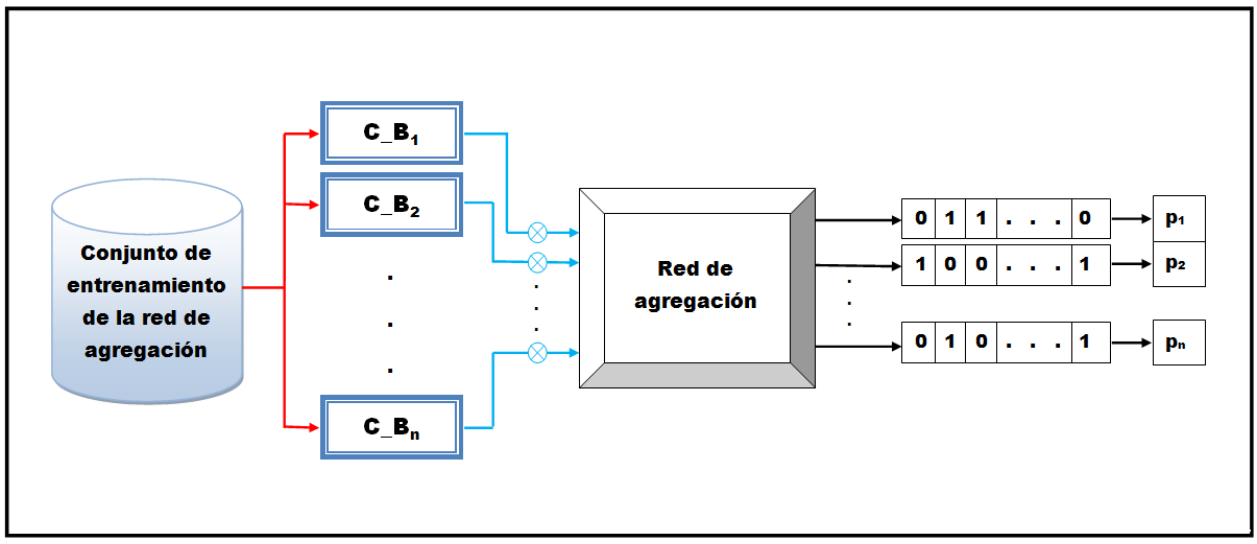
Si volvemos a la sección 3.3, una de las razones que se establecían para el uso de un ensamble es pensar en la estrategia de divide y vencerás, en la cual se vio que a medida que se agregaban más clasificadores de base, las zonas de búsqueda generada por cada uno abarcarían de mejor forma la distribución general de los datos.

En la Figura 3.13, se puede observar que se utiliza un criterio de votación ponderado para la combinación de las clasificaciones individuales, lo que corresponde al segundo componente del ensamble de tipo mezcla de expertos, para la asignación de pesos a cada clasificador de

base se utiliza una red de agregación (*gating network.*) [Ros58]. Mediante esta red se resuelve el problema de la búsqueda de los pesos para cada clasificador de base seleccionados en el ensamble, y su funcionamiento se muestra en la Figura 3.14.

En la Figura 3.14, se puede observar que para el entrenamiento de la red de agregación se selecciona un conjunto de entrenamiento, el cual es utilizado por cada uno de los clasificadores de base ($CB_i \ \forall \ i = 1, \ldots, n$) seleccionados en el ensamble. Estos a su vez obtienen una clasificación individual, las cuales son las entradas de la red de agregación. Esta red de agregación compara las clasificaciones individuales, con las clases reales de cada ejemplo, asignando 1 en caso de clasificar correctamente y 0 en caso contrario.

Figura 3.14: Funcionamiento de la red de agregación



Al finalizar la comparación de todos los ejemplos de entrenamiento, se contabilizan los aciertos y desaciertos, formando un vector por cada ejemplo de entrenamiento. Una vez que se tienen todos los vectores, se calcula la probabilidad de clasificar correctamente cada ejemplo ($p_j \ \forall \ j = 1, \ldots, n$). Esto es, si de un total de ejemplos de entrenamiento n, la probabilidad p_z de tener un acierto es igual a $\#$ *de aciertos*$/n$.

Una vez calculadas las probabilidades de cada clasificador de base, las cuales representan los pesos que tienen cada uno de éstos, posteriormente se aplica el criterio de votación

ponderado para obtener la clasificación final del conjunto de prueba. En la siguiente sección mostraremos un ensamble de tipo mezcla de expertos que desarrollamos en un proyecto de investigación previo.

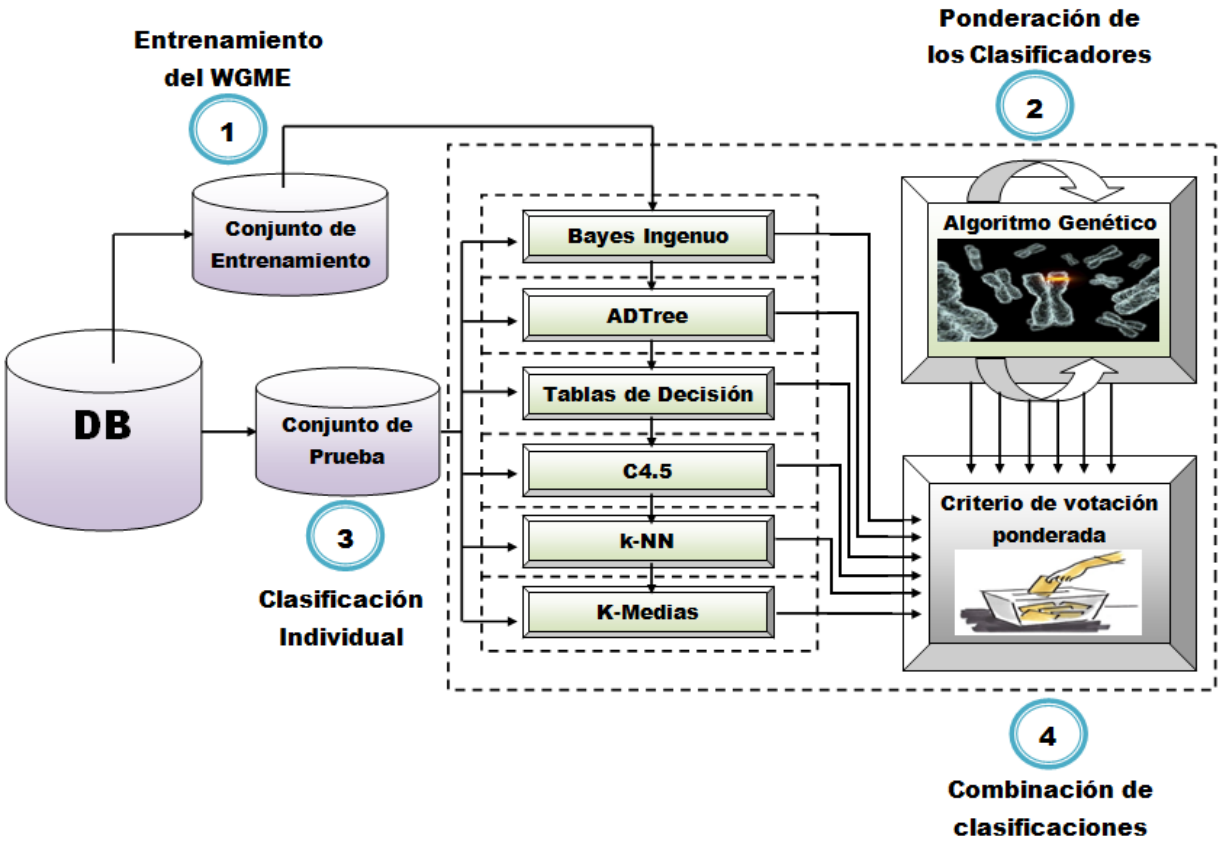
3.11. Clasificador Híbrido con Ponderación Genética

El Clasificador Híbrido con Ponderación Genética (*Hybrid classifier with genetic weighting - HCGW*) [MMMR11], considera un grupo de varios tipos de clasificadores de base a diferencia con trabajos como los de [BK99, Sch03, KPL01, ET93, Bre96, FS97], en los que consideran solamente diferentes modelos de árboles de decisión. Además, se utiliza un algoritmo genético para encontrar la mejor ponderación de cada clasificador considerado, obteniendo mejoras en los índices de rendimiento, mediante la combinación de las clasificaciones individuales de cada uno de ellos. Finalmente, el criterio de combinación de las clasificaciones individuales, es un criterio de votación ponderado, en la Figura 3.11, se muestra el funcionamiento del HCGW.

En las pruebas realizadas con el HCGW, se obtuvieron mejoras en el rendimiento con respecto a un conjunto de clasificadores tradicionales. Para una de ellas que es la exactitud, la cual nos dice el porcentaje de los ejemplos que fueron clasificados correctamente, se obtuvo una mejora promedio del 5.36 % mayor que el que se obtuvo con un conjunto de clasificadores tradicionales (Bayes Ingenuo, Tablas de Decisión, Stacking, ADTree, AdaBoost, C4.5, entre otros).

En cuestión de tiempos de ejecución, el HCGW fue uno de los clasificadores que consumen mayor tiempo, debido a que el tiempo es igual a la suma de los tiempos individuales de cada clasificador de base utilizados, además del tiempo del algoritmo genético que se encuentra implícito como otro elemento dentro del clasificador. Al manejar grandes cantidades de datos y tener gran número de operaciones para cada clasificador, se tiene como resultado un porcentaje muy alto de uso de memoria RAM y CPU, lo que representa un factor dentro de la problemática de clasificación de grandes cantidades de datos revisados en la sección 2.8. A continuación mostraremos dos esquemas paralelos de clasificadores tradicionales de apren-

Figura 3.15: Esquema de funcionamiento del HCGW.



dizaje maquinal, los cuales fueron considerados como posibles clasificadores de base para el PCEM.

3.12. Clasificadores Tradicionales Paralelos

Una vez que se revisaron cada uno de los modelos que existen de los ensambles, ahora revisaremos dos versiones paralelas de clasificadores tradicionales que existen en la literatura, en específico mencionaremos clasificadores basados en árboles de decisión y basados en casos, los cuales están implícitos en el PCEM.

3.12.1. Esquema paralelo de árboles de decisión

Para esta sección veremos las versiones paralelas que se encontraron dentro de la mayoría de los artículos encontrados en la literatura[AK99, JKK06, BCRV00], de las cuales presentamos una idea general, de las cuales se describirá cada una de sus fases de su funcionamiento.

Para este tipo de clasificadores su construcción presenta diferencias con respecto a los algoritmos tradicionales tales como ID3 [Qui86] y C4.5 [Qui93], ya que la idea de implementar un clasificador paralelo es reducir los recursos tales como CPU, memoria RAM y Disco Duro para lograr reducciones en tiempos de ejecución, por esto la mayoría de los artículos que fueron consultados plantean la construcción de árboles de decisión binarios.

Los árboles de decisión binarios [PG09] producen solamente dos hojas por cada uno de los niveles del árbol que se generen, esto representa una manera más ágil de construir los árboles, ya que en esquemas tradicionales los árboles son expandidos de acuerdo con el número total de valores que tenga un atributo en específico, lo cual cuando se tienen BD grandes repercute en el tiempo de ejecución del clasificador. Esto es uno de los factores que se toma en cuenta para el desarrollo de esquemas paralelos de árboles de decisión.

Siguiendo con el desarrollo de árboles de decisión binarios, si se tienen atributos continuos (A), tenemos divisiones de la forma:

$A \rightarrow Valor(A) < x$, donde x es un valor dentro del dominio de A

Si tenemos la presencia de atributos categóricos (B) tendríamos divisiones de la forma:

$B \rightarrow Valor(B) \in X$, donde $X \subset B$

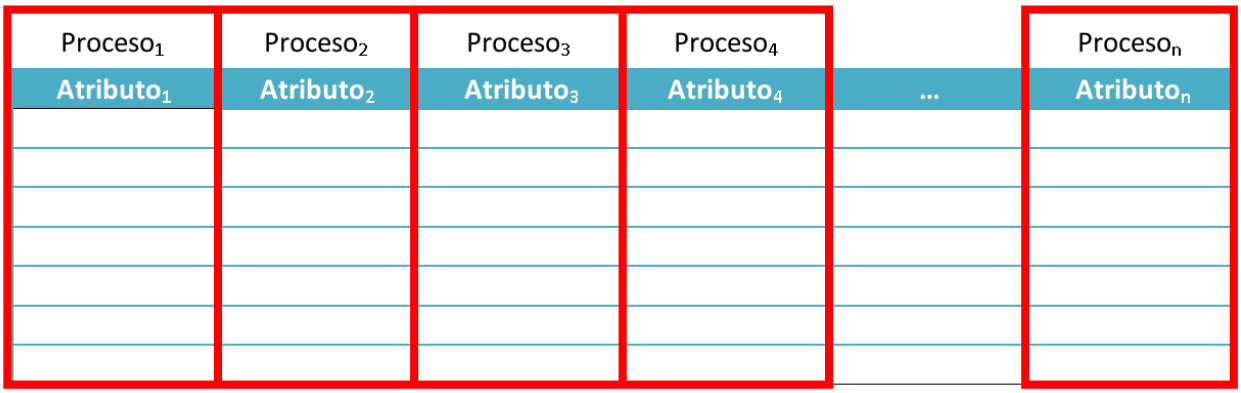
Ya sea cualquier tipo de atributo que presente la BD, se debe de tener una forma de cómo seleccionar un criterio para escoger al mejor atributo a ser dividido y posteriormente aplicar un criterio de división. Para realizar este proceso se selecciona un conjunto de medidas de cantidad de información que se ocupan en los esquemas tradicionales, tales como la entropía, la ganancia de información y la cantidad de información [AK99, JKK06, BCRV00]. Estas medidas nos servirán para seleccionar que atributo es el mejor candidato a ser divido.

El problema de encontrar al mejor candidato para ser dividido sobre BD grandes, afecta directamente el tiempo de ejecución, ya que se realizan una gran cantidad de operaciones

por cada atributo de la BD, lo cual tiene relación con el número de valores que tenga cada atributo. Es mediante este razonamiento donde surge la primera aproximación de llevar a cabo la paralelización de un árbol de decisión.

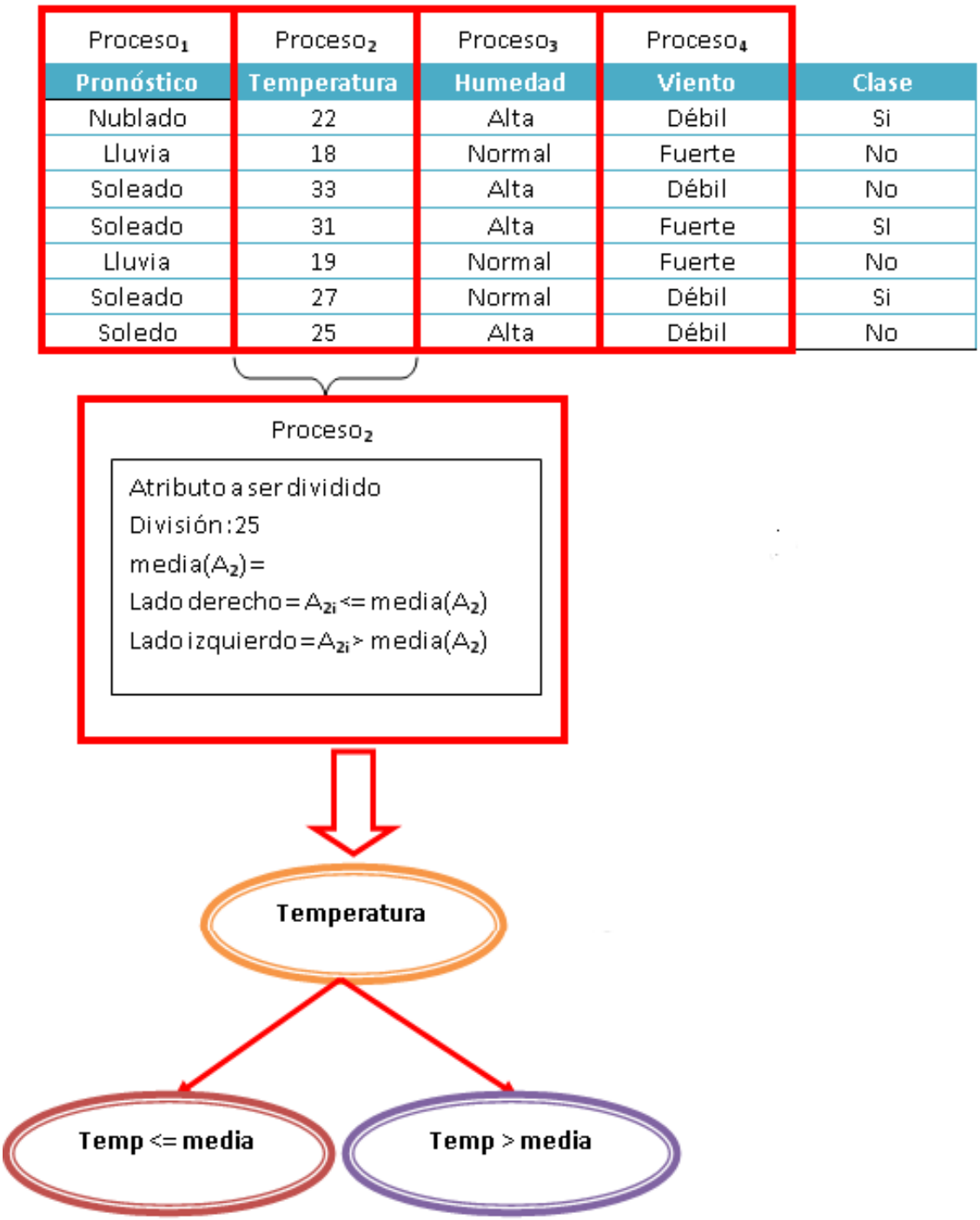
Para este caso se plantea un esquema en el que la BD se ve como una matriz, donde cada columna de la matriz representa un atributo, por lo que a cada atributo se le asigna un proceso para calcular las medidas de información cada vez que se desarrolle parte del árbol, por esto la relación entre los procesos y los atributos es 1 a 1. El esquema de la asignación de procesos se puede ver en la Figura 3.16.

Figura 3.16: Asignación de procesos en versiones paralelas de árboles de decisión.



Ya que se tienen asignados los procesos a cada atributo, se procederá a calcular las medidas de información de cada uno de ellos de manera paralela para elegir al mejor candidato. A medida que se va construyendo el árbol de decisión, se va teniendo un atributo menos por cada nivel del árbol desarrollado, es aquí cuando los procesos van finalizando su tarea mediante el envío de un mensaje a un proceso *n+1* (coordinador) que estará encargado de construir el árbol de decisión, aplicando un criterio de división de acuerdo al tipo de atributo que se esté manejando y de volver a llamar a los procesos que aún continúan encargados de realizar los cálculos sobre el atributo asignado. Este es solo una generalización de los muchos esquemas paralelos para la construcción de árboles de decisión[JKK06, AK99], el cual lo podemos ver ejemplificado en el esquema de la Figura 3.17.

Figura 3.17: Esquema de construcción paralelo de árboles de decisión por división de atributos.



De esta manera es como se puede paralelizar un árbol de decisión [JKK06, AK99], aunque dentro de la literatura se encuentran otras formas de cómo realizar la paralelización de un árbol de decisión, la gran mayoría tiene el mismo funcionamiento del esquema que revisamos anteriormente. En la siguiente subsección veremos cómo se puede paralelizar un clasificador basado en casos llamado k-vecinos más cercanos.

3.12.2. Esquema paralelo de k-vecinos más cercanos

El funcionamiento de este tipo de clasificadores se basa en la representación de los datos en un plano de n dimensiones [FN75], dentro de este plano se encontrarán todos los ejemplos de entrenamiento de la BD utilizada con sus clases correspondientes, de manera que al presentarse un ejemplo nuevo estos podrán predecir a que clase pertenece manejando alguna distancia con respecto a los puntos del conjunto de entrenamiento. A continuación, describiremos cuál es el funcionamiento secuencial de este algoritmo para entender en donde se presenta la paralelización una vez que veamos el esquema paralelo de este clasificador.

Funcionamiento el método de los k-vecinos más cercanos

Teniendo ya definida la distancia euclidiana si el atributo es numérico se puede describir los siguientes pasos para poder desarrollar el método de los k-vecinos más cercanos [Yia93]:

1. Dado un punto x de prueba.

2. Encontrar los k vecinos más cercanos sobre los datos de entrenamiento $x_1, x_2, \ldots, x_n$ a x dada el tipo de distancia utilizada, en este caso con la que más se trabaja es la distancia euclidiana definida como:

$$d(x, x_i) = \sqrt{(((x_{11} - x_{21})^2 + (x_{11} - x_{21})^2))} \forall i = 1, 2, \ldots, n$$

3. En donde cada punto esta denotado por la pareja de coordenadas en n dimensiones de la siguiente forma:

$$(x_{11}, x_{21}), (x_{12}, x_{22}), \ldots, (x_{1k}, x_{2k})$$

4. Dado los k vecinos más cercanos al punto k se hace una votación y se le asigna al punto x la clase que tenga más votos con respecto al número k de vecinos más cercanos que se tenga.

Ejemplo de uso del método de los k-vecinos más cercanos

A continuación, se mostrará un ejemplo del uso de este método. Se tiene el siguiente conjunto de coordenadas en 2 dimensiones:

Con la clase •:

$$x_1 = (1, 0), x_2 = (2, 1), x_3 = (1, 3), x_4 = (-2, 3),$$
$$x_5 = (-1, -2), x_6 = (4, 2), x_7 = (-1, -1)$$

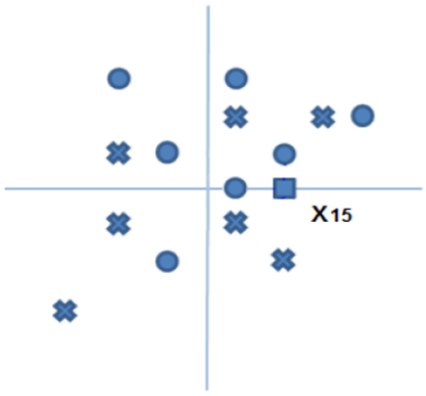
Con la clase **x**:

$$x_8 = (1, 2), x_9 = (-2, 1), x_{10} = (1, -1), x_{11} = (2, -2),$$
$$x_{12} = (3, 2), x_{13} = (-3, -3), x_{14} = (-2, -1)$$

La representación de los puntos de cada una de las clases se puede ver en la Figura 3.18. Una vez que tenemos estos puntos de entrenamiento el punto de prueba que se quiere clasificar es el $x_{15} = (2, 0)$:

Se propone el número de vecinos más cercanos $k = 3$, el cual, en la mayoría de los casos es preferible escoger un número impar, para evitar posibles empates, siempre y cuando se trate de 2 clases. Se puede notar, que para $k = 1$ se tiene el caso del vecino más cercano.

Figura 3.18: Representación de los puntos de entrenamiento.



Se desea encontrar la clase correspondiente al punto x_{15}, por lo que se localizaran cuáles son sus 3 vecinos más cercanos y después se determinará a que clase pertenece viendo que clase tiene más representantes. Para el punto x_{15} después de varios cálculos obtenemos que sus tres vecinos más cercanos son x_1, x_2 y x_{10} esto lo obtenemos seleccionando las 3 menores distancias euclidiana al nodo x_{15} de la siguiente forma:

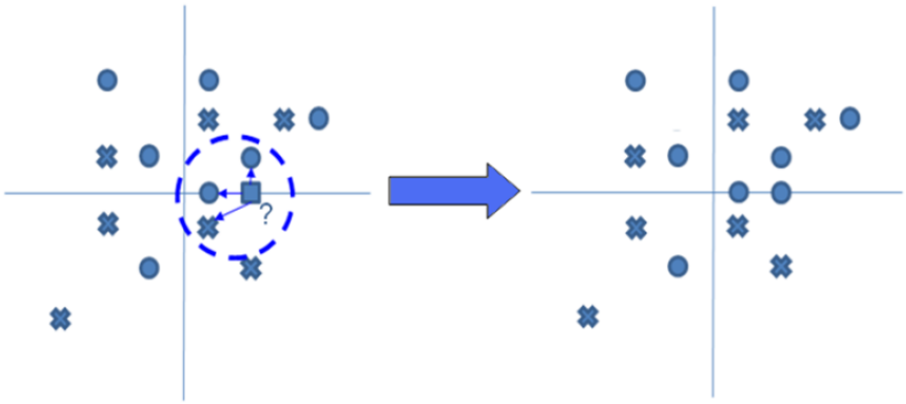
$$d(x_{15}, x_1) = \sqrt{((2-1)^2 + (0-0)^2)} = \sqrt{(1+0)} = \sqrt{1} = 1$$
$$d(x_{15}, x_2) = \sqrt{((2-2)^2 + (0-1)^2)} = \sqrt{(0+1)} = \sqrt{1} = 1$$
$$d(x_{15}, x_{10}) = \sqrt{((2-1)^2 + (0-(-1))^2)} = \sqrt{(1+1)} = \sqrt{2} = 1.41$$

En el plano de dos dimensiones este punto se localiza como se muestra en la Figura 3.19 con sus vecinos más cercanos. Como hay 2 vecinos con la clase •y solo 1 con la clase x, la clase a la que pertenece este nuevo punto x_{15} es •

Versión paralela de k-vecinos más cercanos

Se puede observar en el ejemplo de la sección anterior que el proceso puede resultar sencillo cuando se utilizan pocos datos de entrenamiento con dimensiones pequeñas, pero este tipo de algoritmos son utilizados para grandes cantidades de datos por los buenos índices en las medidas de rendimiento que suelen obtener.

Figura 3.19: Localización de los vecinos y la clasificación del nuevo punto.



Cuando contamos con grandes cantidades de datos el cálculo de las distancias de los k vecinos más cercanos hacia nuevos ejemplos por clasificar, resulta muy complicado [ABH07]. Como pudimos observar en el ejemplo anterior, contamos con los siguientes factores:

1. Por un lado, tendremos un número fijo de vecinos denotado por k, los cuales nos permitirán realizar la clasificación de cada ejemplo del conjunto de prueba.

2. Tenemos también un número de atributos para la BD que estemos utilizando que denotáramos por la variable m, el cual nos dará las dimensiones en las que los atributos serán representados.

3. Finalmente, cada conjunto de prueba contara con un número de ejemplos denotados por la variable n.

Los dos últimos factores nos dan una idea de que tan complicado puede ser el cálculo de las distancias, como pudimos observar en el ejemplo anterior, por cada uno de los ejemplos que se quieran clasificar se debe de calcular las medidas de todos los puntos de prueba que se tienen para poder decidir cuáles son los k vecinos más cercanos, por una parte la variable n nos dirá cuántas distancias tenemos que calcular, mientras que la variable m será el indicador de que tan difícil será el cálculo de cada distancia, ya que representa las dimensiones en la

que representaremos los datos. Podemos ver que, a partir de estos dos factores, la idea de paralelizar este tipo de clasificadores resulta necesaria a medida que se tiene una BD de gran tamaño.

La idea de esta paralelización[ABH07] consiste en dividir los datos de entrenamiento entre un número x de procesos, esto hará que cada proceso se encargue de realizar el cálculo de las distancias para los datos que se le haya asignado. Este esquema reducirá el tiempo de ejecución, ya que ahora se distribuirá el conjunto de prueba en un número mayor de procesos y cada uno de ellos trabajará de forma independiente para obtener los vecinos más cercanos a cada ejemplo de prueba.

Esto sin embargo presenta un problema, ya que ahora no solamente tendremos que encontrar la mejor manera de cómo dividir los datos de prueba, sino que también debemos de establecer cuantos candidatos a ser vecinos más cercanos (vpi) tenemos que utilizar por cada proceso lanzado y cuantos vecinos (vcf) serán los que ocuparemos para realizar la clasificación final del conjunto de prueba.

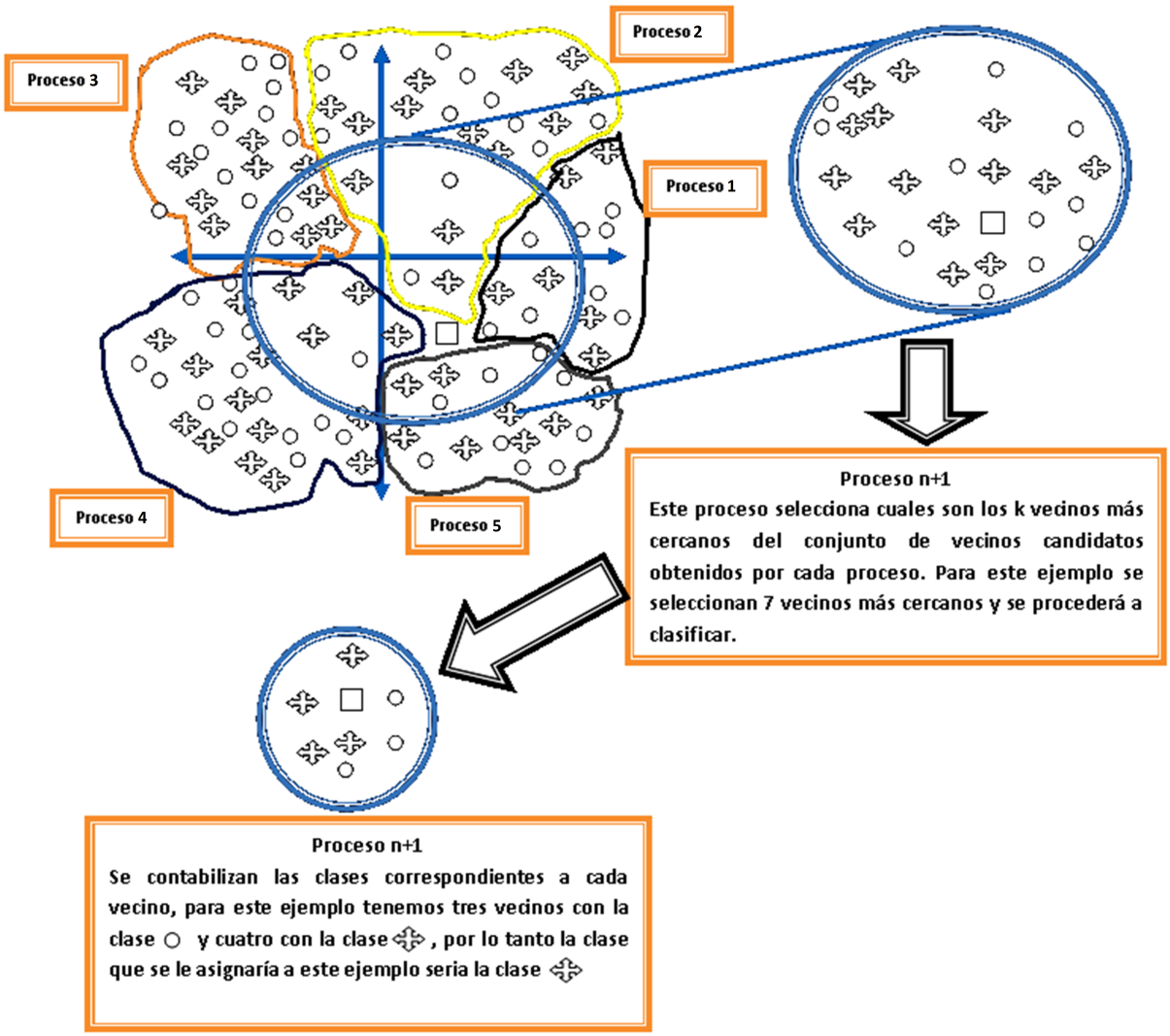
Una forma de cómo podemos realizar esto es primeramente plantear un número aleatorio de procesos lanzados, en donde para cada proceso se fijaría un número impar de vecinos vpi.

El número de vecinos vcf por otra parte sería también un número impar, con esto lo que podemos evitar son empates a la hora de clasificar los datos, sin embargo, esto es solamente una de las muchas formas en las que podemos escoger esta granularidad de este esquema paralelo.

Una vez que cada uno de los procesos calculan las distancias para genera los vecinos vpi, un proceso recolecta cada uno de estos vecinos que obtuvieron cada proceso, por lo que contara con un conjunto de aquellos vecinos candidatos a ser los vecinos más cercanos.

Con este conjunto, el proceso recolector selecciona los vecinos vcf para finalmente obtener la clasificación final del conjunto de prueba, en el esquema que presentamos en la Figura 3.20 podemos ver el funcionamiento de este esquema de cómputo paralelo.

Figura 3.20: Ejemplo de uso del esquema paralelo de los k-vecinos más cercanos.



3.13. Algoritmos de clasificación relacionados con el PCEM

En esta sección se mencionarán algunos trabajos relacionados, no en su totalidad ya que en la literatura no encontramos trabajos exactamente iguales al PCEM, pero que plantean

escenarios de clasificación mediante el uso de un ensamble.

3.13.1. A Parallel Mixture of SVMs for Very Large Scale Problems

Este trabajo fue presentado por Collobert et al. se considera como un trabajo seminal en el estudio referente a la mezcla de expertos de tipo máquina de soporte vectorial (*SVM*), el cual se encuentra en más de 350 trabajos similares.

En este caso los autores proponen un ensamble de tipo experto, con la aclaración de que solo utilizar varios modelos de SVM, los cuales se combinan mediante un criterio de votación ponderado. Se maneja una salida de la mezcla para un vector de entrada x se calcula en base a la Formula 3.1.

$$f(x) = h\left(\sum_{m=1}^{M} w_m(x) s_m(x)\right) \tag{3.1}$$

Donde m es el número de expertos en la mezcla, $s_m(x)$ es la salida del m^{th} experto dado entrada x, $w_m(x)$ es el peso para el experto m^{th} dado por un módulo .ªgregación"teniendo también x como entrada, y h es una función de transferencia que de tipo tangente hiperbólica para tareas de clasificación. En este ensamble cada experto es un SVM, y se utiliza una red neuronal de agregación como la que se revisó en la sección 3.10. Esta red de agregación se entrena para minimizar el costo de la función de evaluación representada en la Ecuación 3.2:

$$C = \sum_{i=1}^{N} (f(x_i)?y_i)^2 \tag{3.2}$$

Para entrenar este modelo, los autores mencionan el siguiente un algoritmo:

1. Se divide el conjunto de entrenamiento en M subconjuntos aleatorios de tamaño cerca de N/M, donde N es el número de expertos en el ensamble.

2. Entrenar a cada experto por separado utilizando alguno de estos subconjuntos.

3. No realizar ninguna clasificación con los expertos hasta entrenar la red de agregación, para minimizar la Ecuación 3.2 en todo el conjunto de entrenamiento.

4. Reconstruir los subconjuntos M: para cada ejemplo (x_i, y_i),
   - Ordenar a los expertos en orden descendente según los valores $w_m(x_i)$,
   - Asignar el ejemplo al primer experto de la lista que tenga menos que $(N/M + c)$ ejemplos[1], a fin de garantizar un equilibrio entre los expertos.
5. Si no se cumple un criterio de terminación (como un número dado de iteraciones o una tasa de error establecido), se va al paso 2.

Los autores marcan la paralización de este ensamble en el paso 2 ya que cada experto puede ser entrenado por separado en un nodo diferente, teniendo un sistema multiprocesador.

Discusión de este ensamble:

- Podemos ver que a pesar de que el nombre hace referencia a una mezcla de expertos, la mezcla solo se realiza con un solo clasificador que en este caso son las SVM.
- En el caso de la combinación de las clasificaciones de cada uno de los expertos, se considera un criterio de votación ponderado, utilizando una red de agregación similar a la que presentamos en la sección 3.10, con el aporte de que se realiza una minimización de la función de evaluación establecida.
- Para el PCEM fue de ayuda este trabajo ya que establece la necesidad de generar subconjuntos de los datos de entrenamiento del tamaño N/M, sin embargo, a diferencia de lo que proponen los autores en este trabajo se consideran subconjuntos que respetan la distribución original de la base de datos y además implementamos un método de validación cruzada para realizar la fase de entrenamiento y prueba del PCEM.
- A pesar de que los autores muestran resultados positivos con respecto a la *UCI Forest dataset*[2], los datos que se pueden manejar son de 500 mil registros ya que al combinar varios modelos de uno de los clasificadores con mayor complejidad el tipo de ejecución

[1] donde c es una pequeña constante positiva. En los experimentos, $c = 1$

[2] La base de datos Forest se encuentra en la siguiente página del UCI: ftp://ftp.ics.uci.edu/pub/rnachine-learning-databases/covtype/covtype.info

del ensamble es igual a la suma de cada componente, tal y como lo revisamos en el capítulo 1 de este documento.

- En el PCEM por la razón anterior descartamos a las SVM, ya que incrementaban los tiempos de ejecución y los beneficios en cuestión de las medidas de rendimiento no eran estadísticamente significativas, estas pruebas las revisaremos con más detalle en la sección 4.3.

3.13.2. Scalable and Parallel Boosting with MapReduce

Uno de los clasificadores basados en ensambles es *Boosting* y en particular el algoritmo *AdaBoost* con trabajos que van desde el 2012 hasta el 2017. En este caso se describirá a grandes rasgos el trabajo de Palit, et al.[PR12], el cual presenta una mejoría al algoritmo *AdaBoost* mediante la incorporación de la herramienta de programación llamada *MapReduce*.

Map-Reduce es un nuevo paradigma de programación distribuida para el entorno de cloud computing introducido por Dean et al[DG08]. El modelo es capaz de procesar paralelamente grandes conjuntos de datos distribuidos a través de muchos nodos. La meta principal es simplificar el procesamiento de datos de gran tamaño mediante clúster económicos de una forma fácil para los usuarios, obteniendo balance de carga y tolerancia a fallos. *MapReduce* tiene dos funciones primitivas: que son la función *Map* y la función *Reduce*, las cuales se definen por el usuario para cumplir con los requisitos especificados en el problema.

Para el trabajo que proponen Palit, et al., las funciones *Map* y *Reduce* realizan las siguientes operaciones:

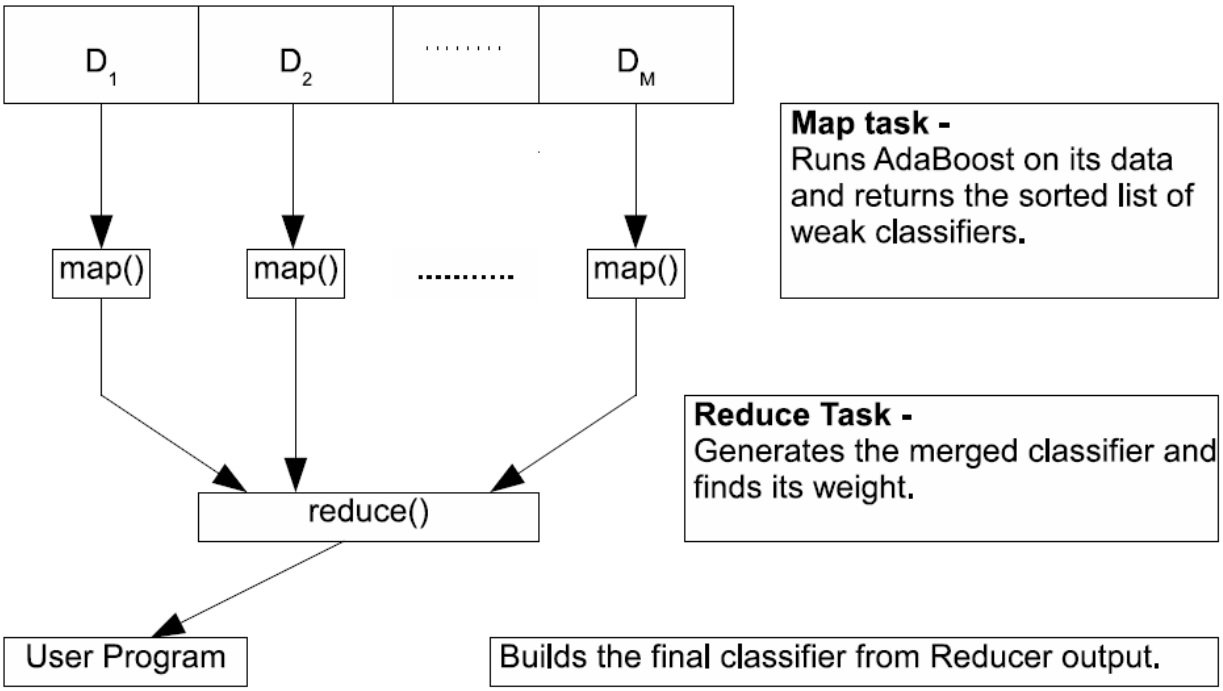
- Para la función *Map*, dado una seria de subconjuntos $\mathcal{D} = \{(x_i, y_i), i = 1, \ldots, N\}$, cada uno de los clasificadores son entrenados mediante el principio del reforzamiento entre cada uno de los *week classifiers* o clasificadores débiles.

- La función *Reduce*, tiene la función de construir el ensamble, tomando como entrada cada uno de los *week classifiers*. Es aquí donde se le asignan los pesos a cada uno de

ellos y donde se aplica el criterio de votación ponderado para encontrar la clasificación final del conjunto de prueba.

Siguiendo las dos funciones que anteriormente presentamos, se respeta el principio de programación de *MapReduce* ya que por un lado se mapea o se representa en un todo las clasificaciones individuales de los *week classifiers* y posteriormente se reduce este mapeo a una clasificación final combinada mediante el criterio de votación ponderada. En la Figura 3.21 se puede corroborar las dos funciones que los autores presentan para esta implementación paralela de ensamble de tipo *Boosting*

Figura 3.21: Esquema del funcionamiento del Algoritmo AdaBoost mediante la incorporación de *Map-Reduce*.



Discusión de este ensamble:

- Este trabajo que presentamos es de gran relevancia ya que es una de las corrientes más actuales en cuestión de implementación no solo de los clasificadores basados en ensam-

bles sino de cualquier clasificador tradicional. Esto se debe al nuevo paradigma que presenta la herramienta *Map-Reduce*, la cual para este tipo de algoritmos es muy intuitiva, ya que siempre se quiere, por un lado, mapear el entrenamiento de un clasificador y por otra parte reducir la clasificación final mediante el mapeo previo.

- Se presenta este trabajo ya que, en la actualidad se sigue generando investigación referente al ensamble AdaBoost, como por ejemplo los trabajos de Haixiang et al.[HYY+16], de Lee et al.[LJL17] y el trabajo de Luo et al.[LYC+16] por solo mencionar algunos.

- Claramente este tipo de implementaciones es un candidato ideal para comparar los resultados que presentamos en este trabajo, sin embargo, es muy difícil obtener una versión de este tipo de implementaciones ya que tienen requerimientos especiales en cuestiones de recursos de cómputo.

- Por el otro lado uno de los trabajos a futuro que se propone de este trabajo es migrar nuestra implementación a un esquema *Map-Reduce* ya que es una herramienta de programación paralela y distribuida muy novedosa en investigaciones referentes a la creación de nuevos modelos de clasificación paralela y sobre todo en la generación de ensambles que no solo consideren varios modelos de un solo clasificador como es el caso del trabajo de Palit, et al.

- Cabe destacar que es importante este trabajo y el que revisamos en la sección 3.13.1 ya que ambos consideran un criterio de votación ponderado, el cual minimiza la tasa de error del ensamble. Dadas estas ventajas en el PCEM proponemos de la misma forma que ambos trabajos que presentamos en esta sección, un esquema de ponderación de los clasificadores mediante un algoritmo genético, el cual describiremos con más detalle en la sección 4.4.3.

- Una cosa importante que propone Palit, et al., es el uso de bases de datos con alto grado de dimensionalidad, en este caso los autores utilizan una base de datos sintética con 200 mil instancias y 20 atributos, lo que nos una dimensión de datos igual a 4 millones. Estas

bases de datos son una buena opción para probar clasificadores que pretenden realizar clasificación sobre grandes cantidades de datos, en este trabajo uno de los objetivos de desarrollar el PCEM fue precisamente realizar este tipo de clasificaciones, es por ello que en los experimentos proponemos el uso de una base de datos con este tipo de características.

3.13.3. Esquemas paralelos de clasificadores tradicionales

Durante la revisión del estado del arte, se seleccionaron los principales esquemas paralelos de clasificadores de tradicionales que fueron seleccionados como base teórica de la implementación que realizamos de cada uno de ellos. Cabe mencionar que para el caso del Bayes Ingenuo y Tablas de Decisiones se tomó la decisión de implementarlos con un esquema de nuestra propia autoría los cuales serán revisados de forma individual en las secciones 4.3.4 y 4.3.5. Como resultado final de esta revisión se mencionarán los trabajos que seleccionamos de la literatura, sobre esquemas paralelos de clasificadores tradicionales, los cuales se muestran en la Tabla 3.2.

Tabla 3.2: Revisión de la literatura sobre los esquemas paralelos de los clasificadores de base.

Autor(es)	*Nombre del artículo*	*Descripción*	*Año de Publicación*
[ZXMO06] Zhang, Y., Xiong, Z., Mao, J., Ou,L.	*The study of parallel k-means algorithm.*	En este artículo los autores manejan una forma de como paralelizar el clasificador k-medias, mediante la distribución de operaciones sobre la formación de nuevos clusters por cada iteración realizada. Se decide utilizar este esquema paralelo dada la simplicidad del algoritmo que proponen los autores.	2006
[ABH07] G. Aparício, I. Blanquer, and V. Hernández.	*A parallel implementation of the k-nearest neighbours classifier in three levels: Threads, MPI processes and the Grid.*	Los autores plantean un esquema paralelo del clasificador k - vecinos más cercanos basado en la partición del conjunto de entrenamiento sobre un determinado número de procesos (MPI, Grid) e hilos (Threads). Se seleccionó este esquema por la similitud de la herramienta de programación paralela que utilizan los . autores para desarrollar el esquema paralelo del k-NN, que es la misma que se utiliza en el PCEM, la cual es MPI.	2006
[AK99] Anurag, S.and Eui-Hong H. and Vipin K.	*Parallel Formulations of Decision Tree Classification Algorithms in three levels: Threads, MPI, processes and the Grid.*	Los autores plantean un esquema paralelo del clasificador C4.5 mediante el enfoque de dividir los atributos sobre un determinado número de procesos, obteniendo una reducción gradual en el tiempo de ejecución de la fase de entrenamiento. Del mismo modo que en k-NN la herramienta paralela utilizada, MPI fue una de las razones por las que seleccionamos este trabajo, además de que la forma en como se plantea la paralelización es la más utilizada en la mayoría de los trabajos relacionados.	1999
[MMMR13] Moreno-Montiel, B. and MacKinney-Romero, R.	*ParalTabs: A Parallel Scheme of Decision Tables Construction*	Los autores plantean un esquema paralelo del clasificador del clasificador Tablas de Decisión Tablas de Decisión basado en la estrategia de divide y vencerás. En esta estrategia los datos de entrenamiento se dividen para reducir la complejidad de la construcción de la tabla de decisión, considerando un mayor número de combinaciones que en el esquema tradicional.	2013
[MMMR14] Moreno-Montiel, B. and MacKinney-Romero, R.	*Parallel Formulations of Naive Bayes Classifier*	Se propone un esquema paralelo del clasificador Bayes Ingenuo mediante la construcción de las tablas de probabilidades condicionales y a priori, divididas en un número determinado de procesos. Cabe mencionar que para los últimos dos clasificadores mostrados en esta tabla, por un lado, para el caso de tablas de decisión no se encontraron trabajos similares en los que se planteara la parelización de este clasificador, es por ello que se decidió proponer una versión propia. Por otro lado, para el clasificador Bayes Ingenuo, se decidió implementar una versión propia ya que el algoritmo es simple por lo cual no se necesito revisar algún trabajo de la literatura.	2013

Capítulo 4

Sistema de Clasificación Paralelo basado en Ensamble de tipo Mezcla de Expertos

4.1. Introducción

En este capítulo explicaremos a detalle funcionamiento de cada uno de los componentes del Sistema de Clasificación Paralelo basado en Ensamble de tipo Mezcla de Expertos (Parallel Classification System based on an Ensemble of Mixture of Experts - PCEM) que proponemos en este trabajo. El PCEM tienen implícito un conjunto de componentes los cuales son: esquemas paralelos de los clasificadores base, esquema paralelo del algoritmo genético de ponderación, módulo para aplicar el criterio de combinación de las clasificaciones individuales por medio del voto ponderado.

Dentro de la arquitectura paralela veremos la incorporación del cómputo paralelo que nos permitió tener una reducción de los tiempos de ejecución en el manejo de grandes cantidades de datos. Mediante el cómputo paralelo podemos resolver muchos problemas en los que se busque la reducción en los tiempos de ejecución, ahorro de memoria, manejo de grandes cantidades de datos, uso máximo del poder de cómputo y distribución de procesamiento.

Para la estructura del ensamble como pudimos observar en el Capítulo 3, existen diferentes modelos de clasificación basados en ensamble, que son Bagging [Bre96], Boosting [Sch03],

Generalización Apilada [Wol92] y Mezcla de Expertos [MU96]. En nuestro caso escogimos una estructura de tipo mezcla de expertos debido a que se tienen mejores índices de rendimiento, lo que es uno de los factores que revisamos en la sección 2.7.3.

Mediante el uso de un ensamble de tipo mezcla de expertos, el funcionamiento del PCEM radica en el uso de diversos clasificadores de base que se combinan mediante un criterio de votación ponderado. Es por ello que en este capítulo se revisaran a detalle cada uno de los clasificadores de base que seleccionamos para el funcionamiento del PCEM. Cabe destacar que dentro de la literatura existen un gran número de clasificadores de base que pudieron formar parte del PCEM, sin embargo mediante el trabajo de Quinlan, et al. [WKRQ$^+$08] reducimos esta lista a los 10 clasificadores más utilizados en minería de datos, utilizando la actualización del 2016 de *The University of Vermont*.

Dentro del enfoque general de un ensamble de tipo mezcla de expertos podemos identificar el componente de combinación de las clasificaciones individuales por el criterio de votación ponderado. Para este criterio a cada uno de los clasificadores de base se les asigna un peso de acuerdo a un entrenamiento previo con una porción de los datos de prueba. Mediante este entrenamiento se puede conocer cuáles son los mejores clasificadores de base para una base de datos en particular. Este punto representa un aspecto importante en el funcionamiento del ensamble de tipo de mezcla de expertos, ya que con una correcta selección de estos pesos se puede obtener una mejoría de rendimiento.

Para el PCEM esta asignación de pesos la realizamos mediante el esquema paralelo de un algoritmo genético, el cual nos permite asignar un peso específico a cada clasificador de base mediante una prueba previa, teniendo de esta manera un ajuste con cada uno de los conjuntos de datos con los que se pruebe el funcionamiento del PCEM. En este capítulo revisaremos tanto el criterio de votación ponderado como el esquema paralelo que se utilizó para la implementación de este componente del PCEM. Finalizaremos este capítulo con la revisión de la comunicación de cada uno de los componentes que nos permite el correcto funcionamiento del PCEM.

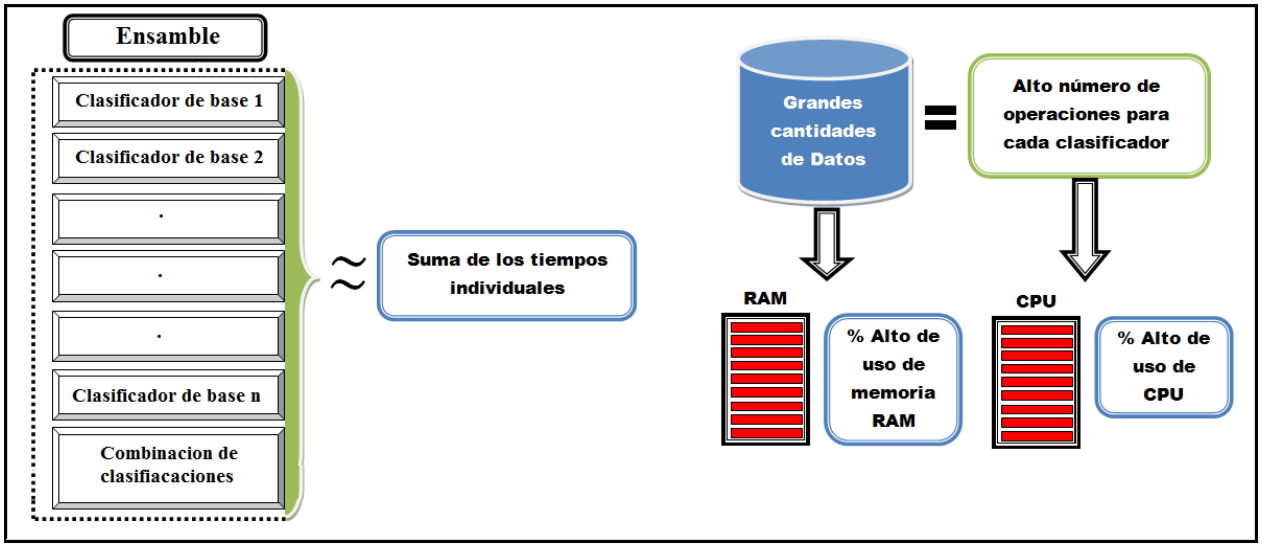
4.2. Arquitectura paralela del PCME

El problema de generalización se presenta en la mayoría de los clasificadores tradicionales del aprendizaje maquinal, el cual se debe a un mal o un sobre entrenamiento. Una de las hipótesis que planteamos en el desarrollo de este proyecto de investigación doctoral, fue tratar de darle solución a este problema mediante el desarrollo de un clasificador basado en ensamble (ensamble) de tipo mezcla de expertos. Mediante el desarrollo de este ensamble se utiliza la estrategia de divide y vencerás, revisada en la sección 3.3, para poder tener cubierto un mayor espacio de búsqueda de los datos de prueba.

Con la incorporación de un conjunto de clasificadores de base en el ensamble podemos evitar el problema de la generalización, ya que se tienen diferentes modelos de clasificación que se pueden ajustar a conjuntos de datos diferentes, sin embargo, con esto nos enfrentamos a nuevos problemas en esta estrategia de clasificación. Como vimos en la sección 1.1, cuando se incorpora un conjunto de clasificadores de base con el fin de obtener mejores índices de rendimiento al llevar a cabo la clasificación sobre grandes datos, tenemos el problema de altos tiempos de ejecución. Esto claramente se puede deducir ya a que mayor número de datos, la complejidad para algunos clasificadores se incrementa teniendo un gran número de operaciones de punto flotante, las cuales son las que consumen más tiempo.

Mediante este pequeño repaso de los problemas implícitos en la clasificación de grandes cantidades de datos, podemos darnos cuenta que la incorporación de computo paralelo nos ofrece una buena alternativa para darle solución a la problemática en su conjunto. Ya vimos que al tratar de solucionar el problema de generalización nos enfrentamos al problema de altos tiempos de ejecución, retomemos el razonamiento que hacíamos en la sección 1.4 para identificar los dos factores particulares que dan pie a este problema, el cual se muestra en la Figura 4.1.

Figura 4.1: Problema de altos tiempos de ejecución en un ensamble



En la Figura 4.1, podemos ver que tenemos un ensamble de tipo mezcla de expertos con un conjunto de clasificadores de bases y un módulo para la combinación de las clasificaciones individuales. Siguiendo el esquema de la Figura 1, tenemos identificados dos factores que nos dan como resultado los altos tiempos de ejecución en los ensambles, por un lado las grandes cantidades de datos generados por el ensamble por un lado afecta en el consumo alto de memoria RAM de un sistema multiprocesador tradicional. Por el otro lado tenemos identificado un alto número de operaciones por cada clasificador de base y el componente de combinación de clasificaciones, los cuales repercuten en tener un alto porcentaje de uso de CPU.

Una vez identificados estos factores particulares, mediante la incorporación del cómputo paralelo podemos darles solución a ambos. Para el factor de manejo de grandes cantidades de datos, mediante el uso de un sistema de multicomputadores, se tiene un mayor poder de computo, en particular una mayor cantidad de memoria RAM en conjunto, la cual se encuentra distribuida en los diferentes sistemas mulprocesador que conforman el cluster con un ancho de banda alto (Gigabits Ethernet). Si bien una computadora comercial en la actualidad puede tener más de 64 GB en RAM y más de 12 procesadores, el precio que se debe pagar

por este tipo de equipos es muy elevado, es por esto que se decidió utilizar una estructura de tipo cluster ya que se tienen PC cada vez más potentes, son escalables, son baratos y son fáciles de actualizar.

Para el factor del alto número de operaciones por cada clasificador y el componente de combinación de clasificaciones, se ofrece una solución de manera individual. Como vimos anteriormente al manejo de grandes cantidades de datos la complejidad de los clasificadores de base aumenta, es por ello que para esta arquitectura paralela se desarrolló una versión paralela para cada uno de los clasificadores de base para obtener una reducción de la complejidad de los esquemas secuenciales.

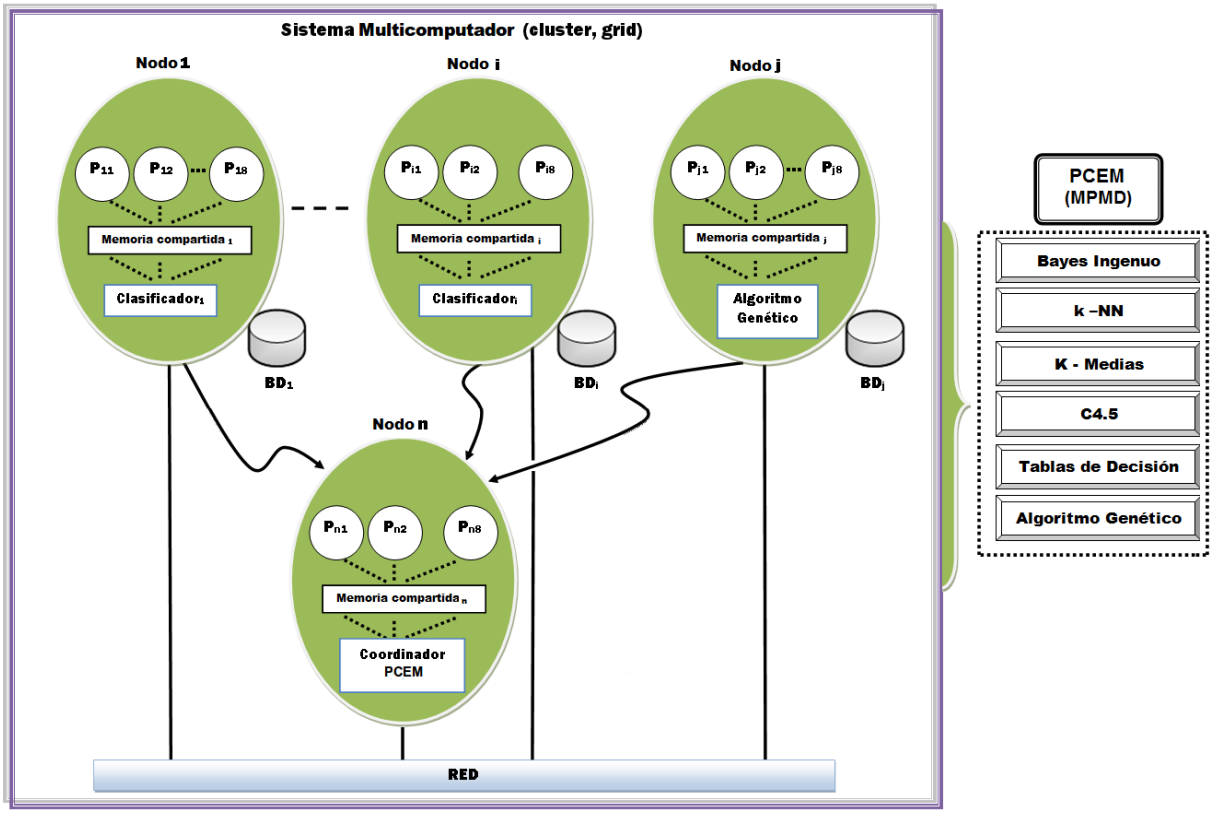
Para el componente de combinación de clasificaciones individuales, en el PCEM se utilizó un criterio de votación ponderada mediante el uso de un algoritmo genético, en este caso se implantaron dos componentes en la arquitectura paralela del PCEM. Un esquema paralelo del algoritmo genético para la asignación de los pesos a cada clasificador de base, y un coordinador general para la recopilación, monitoreo e interpretación de los resultados del PCEM entre cada uno de los miembros que lo conforman, esto es, existe una comunicación del coordinador con el algoritmo genético y a su vez una comunicación entre el coordinador y cada versión paralela de los clasificadores de base. Esta arquitectura paralela del PCEM se muestra en la Figura 4.2.

Podemos observar en la Figura 4.2, que el PCEM se ejecuta en un sistema de multicomputadores, él cual para referirnos a el vamos a ocupar su nombre real, que es el cluster Pacífico, el cual consta de 7 nodos (sistemas multiprocesador) con 8 procesadores y 8 GB en memoria RAM por cada nodo.

Dentro de la Figura 4.2, podemos identificar cada una de las versiones paralelas de los clasificadores de base denotados por $Clasificador_i \ \forall \ i = 1, \ldots, n$ y la versión paralela del algoritmo genético. En este caso n es igual a cinco ya que para esta versión del PCEM desarrollamos cinco versiones paralelas de clasificadores tradicionales, los cuales explicaremos más adelante.

Otro de los componentes que podemos visualizar es el esquema paralelo del algoritmo

Figura 4.2: Arquitectura paralela del PCEM



genético para la ponderación de cada clasificador de base. Finalmente, en el esquema podemos observar que tenemos un coordinador del PCEM que recopilara las clasificaciones individuales de cada clasificador de base, así como la ponderación asignada a cada uno de estos mediante el uso del algoritmo genético. Este coordinador realizara la clasificación final aplicando el criterio de votación ponderado, para obtener la clasificación final del conjunto de prueba.

Podemos observar que, para los clasificadores de base, el algoritmo genético y el coordinador, su funcionamiento sigue el mismo patrón, se tiene un conjunto de procesos que se ejecutan de manera paralela en cada nodo los cuales a su vez tienen asignada una memoria compartida. La comunicación de los procesos se realiza mediante el uso de mensajes para tener un monitoreo de la ejecución local de cada componente del PCEM. Finalmente, la

arquitectura de programación que se utiliza en el PCEM es una arquitectura MPMD, ya que se realizan un conjunto de instrucciones por cada componente del PCEM para diferentes conjuntos de datos.

En la siguiente sección describiremos la estructura del ensamble que se seleccionó para el desarrollo del PCEM, así como el criterio de selección de cada componente del ensamble.

4.3. Estructura del ensamble

Para el desarrollo del PCEM se eligió una estructura de tipo mezcla de expertos, en la cual combinamos un conjunto de clasificadores de base mediante el uso de un criterio de votación ponderada. Para la selección del conjunto de clasificadores de base, establecimos los siguientes criterios:

1. La implementación de cada esquema paralelo para los clasificadores de base considerados tenía que ser simple, esto debido a que uno de los objetivos de cualquier ensamble es la construcción de un aprendiz o clasificador fuerte a partir de un conjunto de aprendices débiles.

2. Seleccionar algunos esquemas paralelos de clasificadores reportados en la literatura para tener una base de comparación del correcto funcionamiento del PCEM.

3. Los esquemas paralelos de los clasificadores de base seleccionados debían poder manejar grandes volúmenes de datos.

4. Finalmente, mediante la selección de esquemas paralelos para los clasificadores de base se pretende obtener una sustancial reducción de tiempos de ejecución por cada clasificador de base presente en el ensamble.

Cabe mencionar que dentro de la revisión de los posibles clasificadores de base existe un gran número de opciones. Por ello, para reducir las opciones que tenemos reportadas en la literatura nos apoyados del trabajo de Quinlan, et al. [WKRQ$^+$08], en el cual se revisa una

lista de los clasificadores más utilizados en minería de datos. Algunos de los clasificadores presentados en este trabajo cumplieron con el criterio establecido para la construcción del ensamble en el PCEM los cuales se enlistan en la Tabla 4.1.

Tabla 4.1: Lista de clasificadores seleccionados.

Caso	*Nombre*	*Tipo de aprendizaje*
1	Bayes Ingenuo	Aprendizaje supervisado
2	k-Vecinos más Cercanos	Aprendizaje Supervisado
3	Decision Tables	Aprendizaje Supervisado
4	C4.5	Aprendizaje Supervisado
5	K-Medias	Aprendizaje No supervisado

En la Tabla 4.1 podemos observar que tenemos cuatro clasificadores del aprendizaje supervisado (Bayes Ingenuo, k-Vecinos más Cercanos, Decision Tables, C4.5) y un clasificador del aprendizaje no supervisado (K-Medias). En el caso de K-Medias se puede utilizar como un clasificador del aprendizaje supervisado, ya que para este caso se puede fijar el número de clusters igual al número de clases presenten en los datos a ser clasificados.

Decidimos utilizar este número de esquemas paralelos debido a que, por una parte, con un número menor no obtuvimos mejoras en los índices de medidas de rendimiento. Por otra parte, con un número mayor de clasificadores el tiempo de ejecución se incrementaba y la mejora en los índices de medidas información no resulto estadísticamente significativo. Para constatar esto realizamos una prueba con el PCEM incorporando una serie de versiones paralelas de los clasificadores *ID3 - Iterative Dichotomiser 3*, Clasificador con probabilidades condicionales y riesgo condicional, Clasificador con probabilidades condicionales de distribución gaussiana, *ADTree - Alternating Decision Tree* y el clasificador *Apriori*.

Los resultados que se presentan son sobre la base de datos *Bank Marketing*[1], la cual

[1]La base de datos Bank Marketing se encuentra en la siguiente página del UCI: ftp://ftp.ics.uci.edu/pub/rnachine-learning-databases/covtype/covtype.info

representa una base de datos con una alta dimensionalidad en el número de atributos (más de 45 mil), realizando las pruebas mediante validación cruzada, estos resultados los podemos ver en las gráficas de las Figura 4.3 y Figura 4.4.

Figura 4.3: Tiempos de ejecución con diferente número de clasificadores

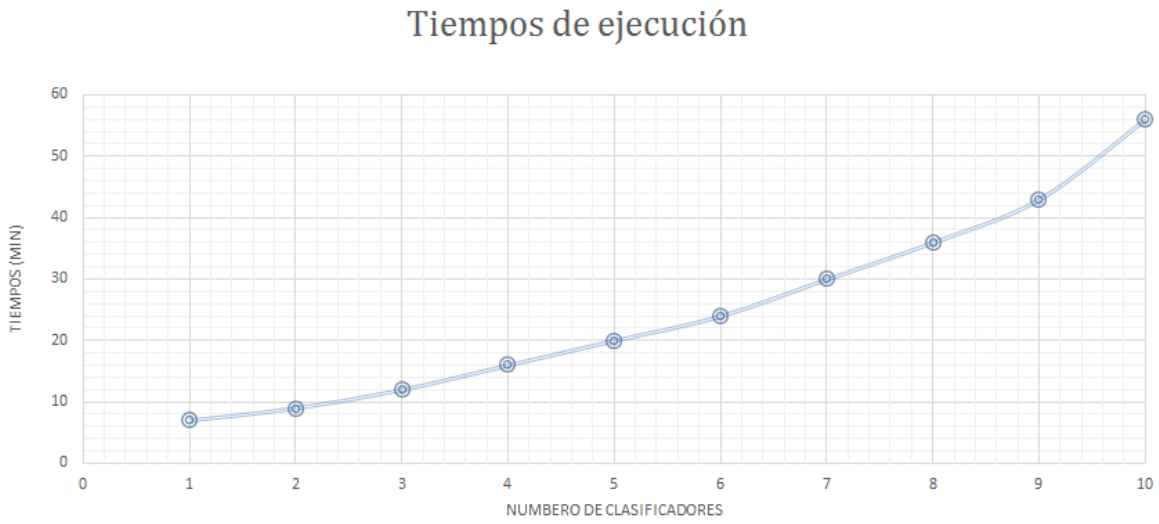
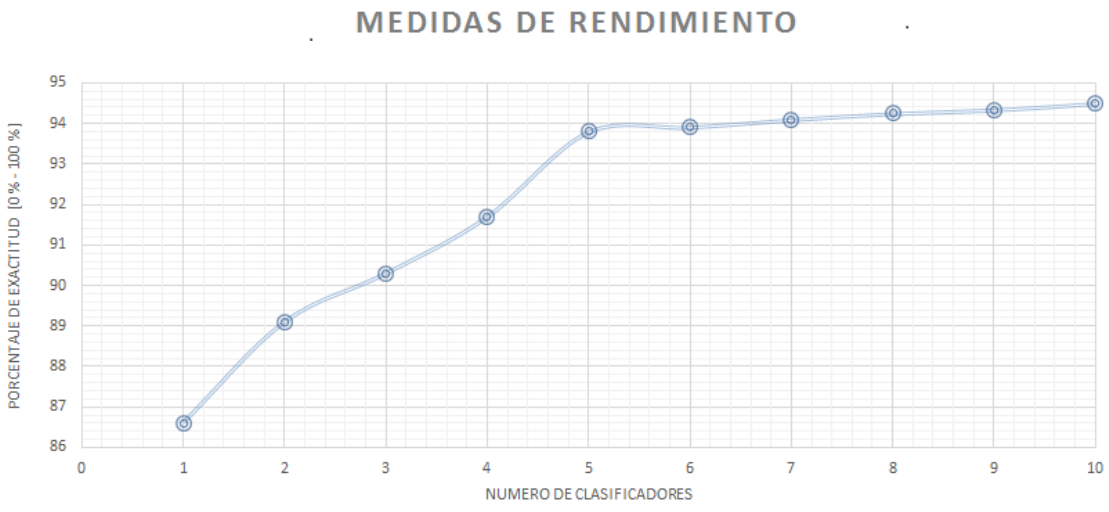


Figura 4.4: Medidas de rendimiento con diferente número de clasificadores



Podemos ver en la Figura 4.3 que el aumento del tiempo de ejecución crece de forma exponencial, ya que podemos observar que si incrementamos el número de clasificadores de base al doble (10 clasificadores) el tiempo se incrementa a más de 45 minutos. Se podría pensar que este aumento es aceptable siempre y cuando hubiera un aumento considerable en las medidas de rendimiento, pero no fue así, ya que podemos ver en la Figura 4.4 que el aumento no es estadísticamente significativo.

Para llegar a esta conclusión realizamos una prueba estadística estableciendo un intervalo de confianza del 95 % y calculando la tasa de error cuando utilizamos 5 clasificadores y el promedio de usar de 6 a 10 clasificadores, estos datos se pueden observar en la Tabla 4.2

Tabla 4.2: Datos para realizar comprobar la significación estadística de usar 5 o más clasificadores

Número de Clasificadores	Tasa de Error ((100 - exactitud)/100)	p
5 (mejor candidato)	(100-93.812)/100 = 0.06188	$p1$
8 (promedio de 6 a 10)	(100-94.202)/100 = 0.05798	$p2$

Por lo que la pregunta que se formuló fue la siguiente: ¿Existe diferencia significativa respecto del usar más de 5 clasificadores de base?, para lo cual se propusieron las siguientes hipótesis.

- H_0 (hipótesis nula) = No hay diferencia al usar más clasificadores, $\therefore$ el número máximo de clasificadores es 5.

- H_a (hipótesis alternativa) = Sí existe diferencia al usar más clasificadores, $\therefore$ escogemos el promedio de clasificadores que en este caso es 8.

Si $[p1 - p2]$ es mayor que el producto de $1.96(Z_\alpha - 0.05)$ multiplicado por el error estándar, concluimos que la diferencia es significativa. Por lo tanto, se presenta el calcular el error estándar para luego compararlo con la diferencia observada en los dos escenarios del número de clasificadores utilizados.

$$[p1 - p2] = [0.06188 - 0.05798] = 0.0039$$

$$p = [p1 + p2]/2 = [0.06188 + 0.05798]/2 = 0.05993$$

Procedemos a calcular el error estándar de la siguiente forma:

$$Errorestandar = \sqrt{p \times (1-p) \times (1/NC_1 + 1/NC_2)}$$

$$= \sqrt{0.05993 \times (1 - 0.05993) \times (1/5 + 1/8)} = 0.135314369$$

Finalmente el error estándar multiplicado por:

$$Z_\alpha - 0.05 = 0.135314369 \times 1.96 = 0.265216163$$

Dado que la diferencia de $[p1 - p2] = 0.0039$ no supera al error estándar multiplicado por $Z_\alpha - 0.05$ (0.265216163), concluimos que no existe una diferencia estadísticamente significativa al utilizar más de 5 clasificadores; razón por la cual rechazamos H_a, por ende, aceptamos la H_0 que en este caso es utilizar 5 clasificadores. Esta es la razón del por qué seleccionamos este número de clasificadores de base para esta versión del PCEM.

Una vez que presentamos el número y tipo de clasificadores de base seleccionados, en la siguiente sección vamos a describir cada uno de los esquemas paralelos en algunos casos utilizamos los trabajos reportados en la literatura y en otros se trata de implementaciones propias, para ser más puntuales el esquema paralelo de Tablas de Decisión y de Bayes Ingenuo son implementaciones propias tal y como lo mencionamos en la sección 3.13.3.

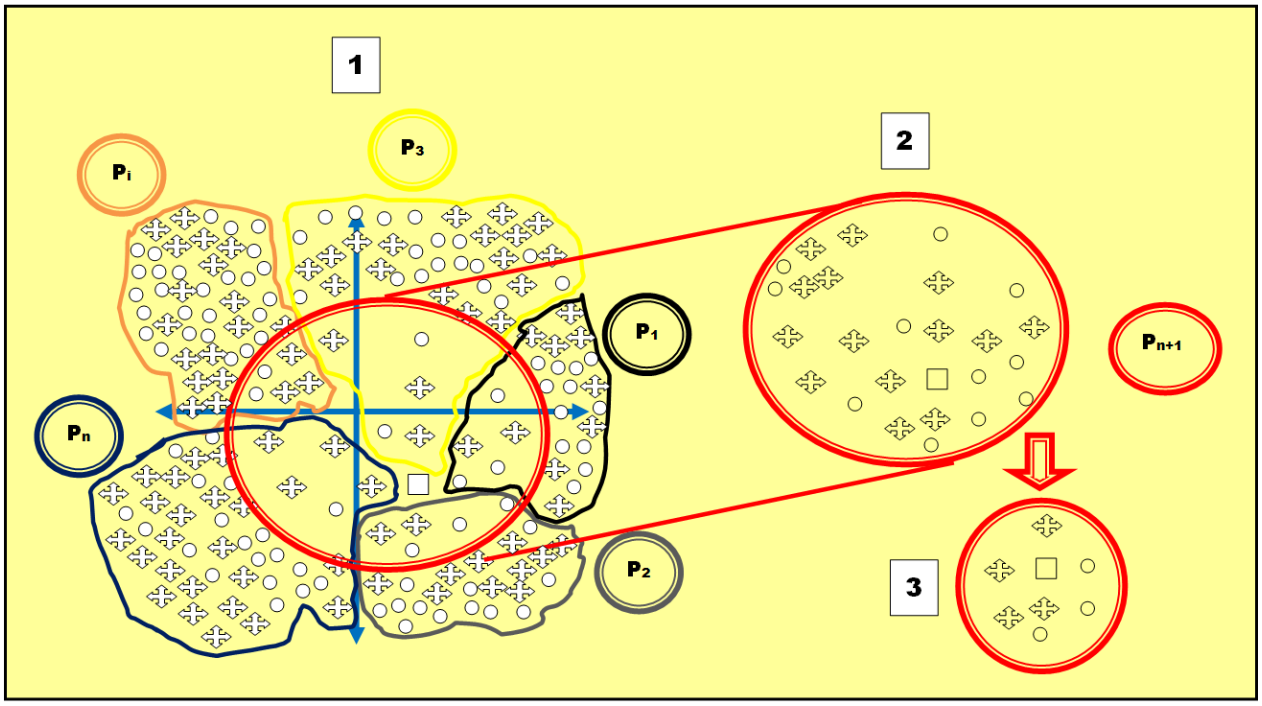
4.3.1. Esquema paralelo de k-vecinos más cercanos

El clasificador k-vecinos más cercanos (kNN) pertenece a los clasificadores basados en casos, en el cual para realizar la clasificación de datos se utilizan medidas de similitud. Usualmente la medida más utilizada es la distancia Euclidiana con la cual se definen los vecinos más cercanos (ejemplos del conjunto de entrenamiento) para cada ejemplo del conjunto de prueba.

Cabe mencionar que se decidió utilizar datos numéricos ya que el funcionamiento de kNN y K-Medias necesitan de este tipo de datos. En el caso de Bayes Ingenuo, C4.5 y Tablas de Decisión no es forzoso que sean numéricos ya que se pueden usar datos de tipo cadenas de caracteres, sin embargo, por un estándar decidimos que todos los datos se cambiarían a un formato de tipo numérico en un módulo que constituye una de las funcionalidades del PCEM.

Una vez que seleccionamos los vecinos más cercanos, se aplica un criterio de votación por mayoría para definir a que clase pertenece cada ejemplo del conjunto de prueba. Para el esquema paralelo de este clasificador de base seleccionamos el trabajo de Aparício [ABH07] revisado en la sección 3.11.2, el cual se basa en la distribución de la selección de los vecinos más cercanos, en la Figura 4.5 se muestra este esquema paralelo.

Figura 4.5: Esquema paralelo de kNN



En la Figura 4.5 se puede observar que se tienen tres pasos para el funcionamiento de este esquema paralelo los cuales son:

1. Dividimos el conjunto de entrenamiento en subconjuntos de acuerdo al número de proce-

sos seleccionados (para este ejemplo tenemos n procesos), teniendo así una distribución de los datos para la búsqueda de los candidatos a ser los vecinos más cercanos a cada ejemplo del conjunto de prueba.

2. El proceso denotado por $n+1$, recopila cada uno de los candidatos a ser vecinos más cercados por cada proceso ejecutado. Una vez que el proceso $n+1$ obtiene dichos candidatos, se establece el número de los vecinos más cercanos que se van a ocupar para un conjunto de datos en particular, en la Figura 4.5 se seccionan siete vecinos más cercanos.

3. Una vez que se obtienen los vecinos más cercanos se contabiliza los votos para cada clase y se asigna la clase de mayoría de votos por cada ejemplo de prueba. En el ejemplo de la Figura 4.5 se tienen tres ejemplos para la clase ❍ y cuatro para la clase ✤, por lo que la clase asignada es ✤.

4.3.2. Esquema paralelo de K-Medias

El término de *clusters*, en español es traducido como conglomerados, aunque en la mayoría de los trabajos que se revisaron al respecto se utiliza su término en inglés, es por ello que utilizaremos Análisis de Clusters en vez de Análisis de Conglomerados. El principal objetivo que se tiene en la técnica del análisis de clusters es agrupar un conjunto de datos en un número establecido de clusters o grupos, este agrupamiento se realiza mediante la idea de distancia o similitud entre cada una de las observaciones.

Según sea el criterio de similitud o la distancia seleccionada obtendremos diferentes estructuras de clusters. Pongamos de ejemplo el caso en que se tiene una baraja española, en donde se tienen cuatro palos (oros, copas, espadas y bastos) y cada palo tiene cartas enumeradas del 1 (representado con As) al 7 así como tres cartas con figura (sota, caballo y rey). Si deseamos dividir en un número de clusters este conjunto de datos representado por todas las cartas de la baraja, tendríamos distintos modos de hacerlo: en cuatro clusters (los cuatro palos), en ocho clusters (los cuatro palos y según sean figuras o números), en dos clusters

(figuras y números). Es decir, depende de lo que consideremos como similar.

Podemos ver que el número de combinaciones posibles de grupos y de los elementos que integran cada uno de estos grupos, se convierte en un problema muy difícil de resolver por la computadora, aun cuando el número de datos no sea muy grande. Es por esto que se desarrollaron métodos que no exploraran el conjunto total de combinaciones, solamente se maneja un subconjunto de estas para encontrar el número de clusters y cuantos componentes tiene cada uno.

El método de las k-medias permite agrupar al conjunto de observaciones en k clusters, definidos previamente, en donde esta asignación se basa en la distancia existente entre cada observación al *centroide*(media) del cluster más cercano. Con frecuencia la distancia empleada es la distancia euclidiana.

Este método es utilizado como una técnica exploratoria, en donde se irán clasificando las observaciones en cada uno de los clusters definidos y se irá iterando buscando los centroides de cada cluster para ver si hay alguna nueva asignación de una observación a un nuevo cluster, este será nuestro criterio de paro, ya que, si no existe ninguna nueva asignación en una iteración, el clasificador terminará.

Los pasos necesarios para poder desarrollar el método de las k-medias son[Mar01]:

1. Se toman de manera aleatoria k centroides iniciales.

2. Para cada una de las observaciones, se calcula la distancia que hay a los centroides y se reasignan a los que estén más próximos. Una vez que se concluye esta reasignación se vuelven a calcular los centroides de cada uno de los clusters.

3. Se repiten los dos primeros pasos hasta que no se presente una nueva reasignación de alguna observación a uno de los k centroides.

En un caso similar que k-NN, cuando el tamaño de los datos se incrementa, el número de operaciones para la búsqueda de nuevos clusters se incrementa también. Esto se debe principalmente a que en este tipo de clasificador se establece un número de iteraciones como criterio de paro. Esto se debe principalmente a que en ocasiones el paso 3 del funcionamiento

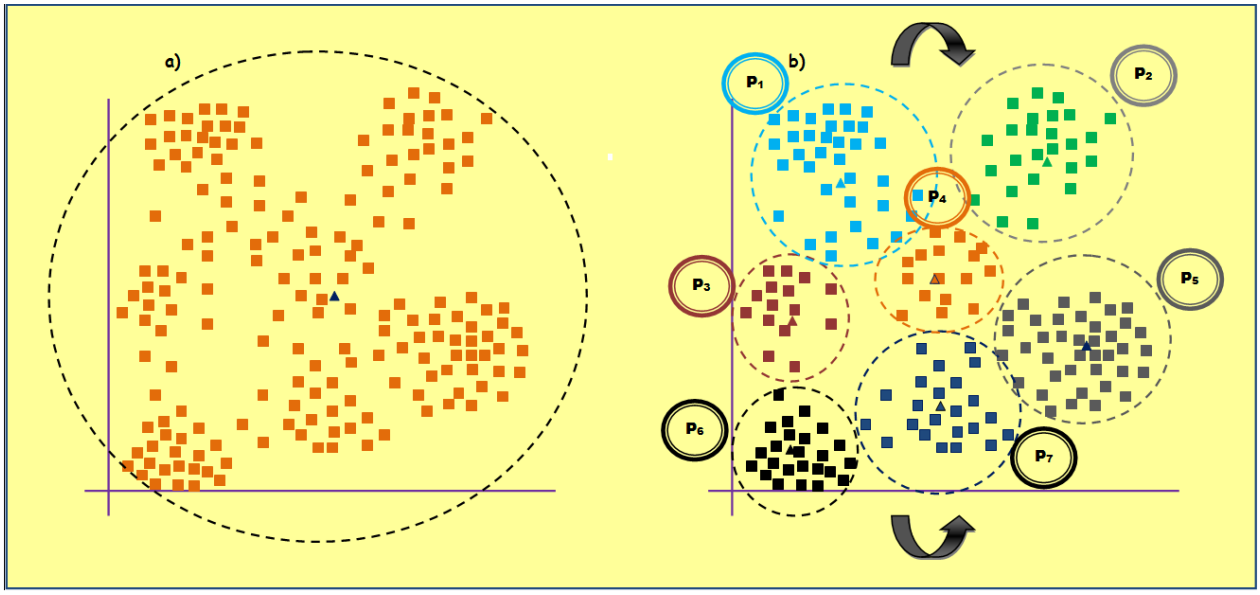
de K-Medias, no se alcanza a cumplir para una distribución de datos compleja. Es por ello que normalmente este número de iteraciones va acorde a un porcentaje en una de las medidas de rendimiento, la cual es la exactitud.

Esta medida de funcionamiento nos proporciona el porcentaje de los ejemplos que fueron clasificados de manera correcta del total de ejemplos en los datos. El porcentaje que se fija para tener un criterio de paro en el algoritmo de K-Medias se ajusta de acuerdo a la base de datos que se maneje. Por poner un ejemplo de esto para la base de datos Gemius Data del EMCL PKDD 2007 [20007], revisando los resultados reportados en la literatura, se obtiene una exactitud en un rango que va de [70 % al 77 %]. Teniendo esto como referencia una condición de paro que podríamos establecer para el algoritmo de K-Medias sería que la exactitud fuera mayor o igual que 70 %.

Al inicio de la ejecución del clasificador K-Medias se establece cual es el porcentaje de exactitud que se busca, si se buscan exactitudes altas normalmente el número de iteraciones aumenta, ya que en ocasiones la frontera de decisión entre cada clase puede ser compleja tal y como se revisó en la sección 3.2. En este esquema paralelo que seleccionamos para este clasificador de base[KMHH11], se basa en la distribución de las operaciones por cada cluster formado en número determinado de procesos, en el esquema de la Figura 4.6 se muestra dicho esquema paralelo.

En el diagrama de la Figura 4.6, se puede observar que el funcionamiento de este esquema paralelo está dividido en dos etapas. La primera etapa (a), consiste en dividir y asignar un proceso ($P_i \ \forall \ i = 1, \ldots, n$) para cada centroide (en el esquema cada centroide está representado por un triángulo) que se quiera formar, el cual es igual al número de clases presentes en los datos de prueba. La segunda etapa (b), consiste en llevar a cabo, mediante el uso de los procesos seleccionados por cada cluster, la separación de los datos de prueba hasta alcanzar el porcentaje de exactitud establecido.

Figura 4.6: Esquema paralelo de K-Medias



4.3.3. Esquema paralelo de C4.5

El clasificador C4.5 [Qui93] fue desarrollado por Quinlan, el cual representa una mejora de su predecesor llamado ID3 [Qui86]. Este algoritmo se basa en la construcción de árboles de decisión a partir de un conjunto de entrenamiento, el cual se lleva a cabo mediante la estrategia de búsqueda de ascenso a la colina (*hill climbing search*). En cada uno de los niveles de la construcción del árbol se escogen aquellos atributos del conjunto de entrenamiento de mayor ganancia de información y entropía. El árbol de decisión resultante esta formados por:

- *Nodos:* Nombres o identificadores de los atributos.
- *Ramas:* Posibles valores del atributo asociado al nodo.
- *Hojas:* Conjuntos ya clasificados de ejemplos etiquetados con el nombre de una clase.

En el caso de los atributos discretos (un atributo discreto tiene un número finito de valores) se realiza una prueba con los n posibles valores de cada atributo, para saber cuál es el valor seleccionado para la expansión del árbol. Para cada atributo discreto (se tiene un

número infinito de posibles valores), se considera una prueba con n resultados, siendo n el número de valores posibles que puede tomar el atributo. En el caso de atributos continuos (se tiene un número finito de posibles valores), se realiza una prueba binaria sobre cada uno de los valores que toma el atributo en los datos. Estas son las dos posibles pruebas que se realizan para la expansión del árbol en cuyo caso se elige aquel valor del atributo con mayor ganancia de información. El pseudocódigo del algoritmo C4.5 para la construcción de un árbol de decisión se muestra en el Algoritmo 3.

El problema de este clasificador radica en la creación del árbol de decisión mediante el uso de la estrategia *hill climbing*, ya que a medida que los datos se incrementan existe un gran número de combinaciones de cómo desarrollar el árbol de acuerdo a los valores para cada tipo de atributos presentes en los datos de entrenamiento.

El esquema paralelo que seleccionamos para este clasificador de base, se encuentra reportado en el trabajo de Anurag [AK99], del cual tomamos la idea de paralizar la construcción de un árbol de decisión mediante la división del conjunto de entrenamiento en un número determinado de procesos acordes al número de atributos, ya que como lo habíamos mencionado es aquí donde se consume el mayor tiempo de ejecución del clasificador C4.5.

En este caso, cada proceso i encargado de un atributo en particular, realiza el cálculo de las medidas de información de cada valor presente en el atributo. Un proceso coordinador realiza el monitoreo, así como la llamada a cada proceso encargado de un atributo en particular, para ir desarrollando el árbol mediante el mismo principio de funcionamiento establecido en el algoritmo C4.5. Con este esquema paralelo se obtuvo en promedio una reducción de tres cuartas partes del tiempo que obtiene el esquema secuencial. Finalmente, en la Figura 4.7 se puede observar el esquema de funcionamiento del esquema paralelo de C4.5

Entrada:

- C4.5 (R,C,S);
- R: Atributos no clasificados, C: Atributos con clases, S: Ejemplos.

si *S esta vació* **entonces**
| devolver un único nodo con Valor Falla y la clase;

fin

sinó, si *todos los ejemplos son de la misma clase* **entonces**
| C4.5 (hoja con clase)

fin

sinó, si *R esta vacío* **entonces**
| Devolver un único nodo con el valor más frecuente del atributo
| Clasificador en los registros de S

fin

en otro caso
| $D \leftarrow$ atributo de mayor Ganancia de Información (D, S)
| entre los atributos de R
| Sean $d_j, j = 1, 2, \ldots, m$, los valores del atributo D;
| Sean $S_i, i = 1, 2, \ldots, m$, los subconjuntos de S correspondientes a los valores de d_j
| respectivamente;
| Devolver un árbol con la raíz nombrada como D
| y con los arcos nombrados d_1, d_2,...,d_m, que van respectivamente a los árboles

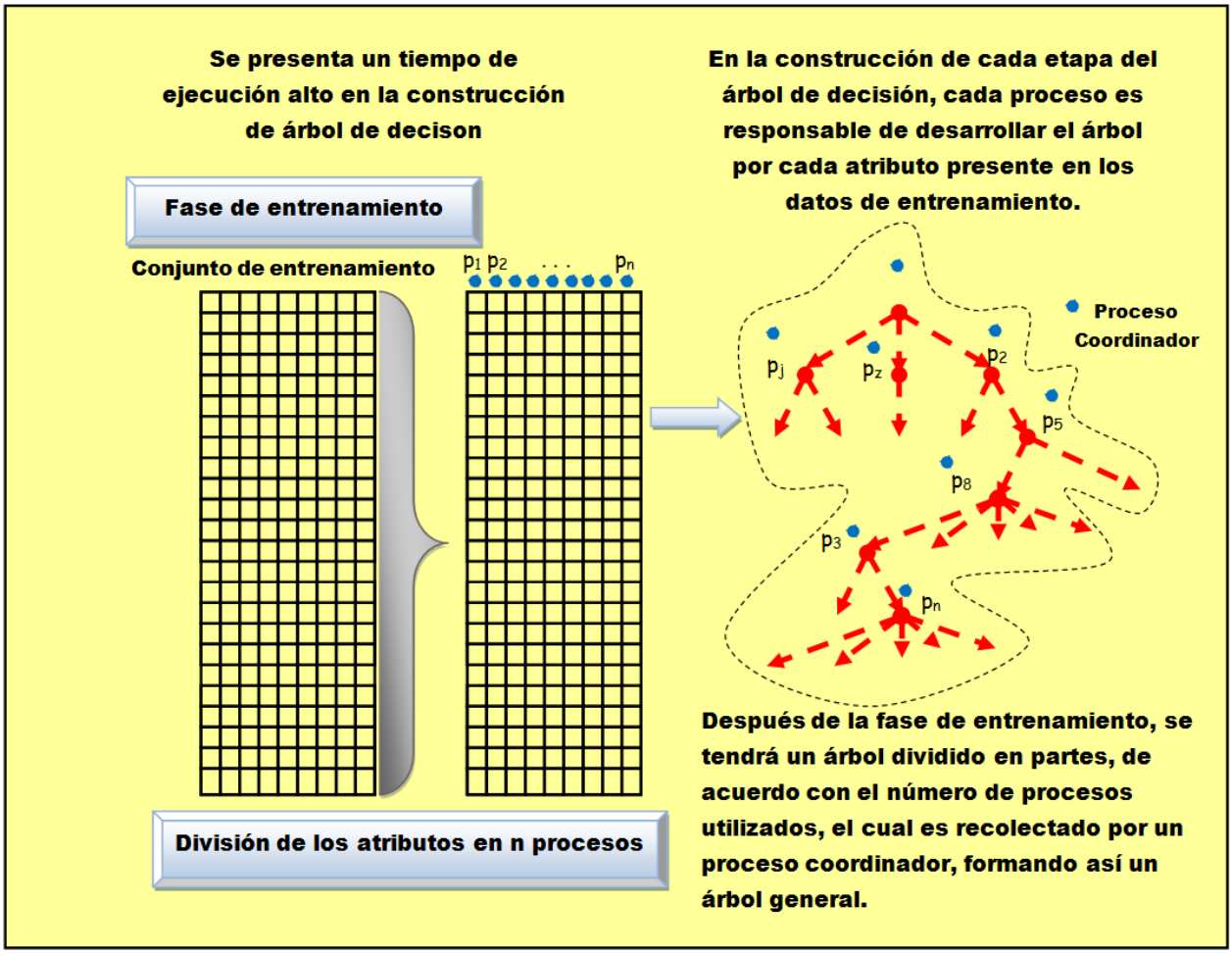
fin

para *S_i en S* **hacer**
| C4.5 $\leftarrow (R - D$, Clase Mayoritaria, $S_i)$.

fin

Algoritmo 3: C4.5

Figura 4.7: Esquema paralelo de C4.5



4.3.4. Esquema paralelo de Tablas de Decisión

Dentro de este trabajo de doctorado se desarrolló un esquema paralelo para el clasificador de tablas de decisión, ya que en la revisión del estado del arte no encontramos algún trabajo similar, el cual es descrito a continuación. Muchos procesos de toma de decisiones pueden ser tratados por medio de Tablas de Decisión[Koh95], en las que se representan los elementos característicos de estos problemas:

- Las diferentes *combinaciones de valores* que puede presentar la naturaleza denotados

por v_1, v_2, ..., v_n. En donde cada $v_i \; \forall \; i = 1, \ldots, n$, tiene un valor especifico.

- Existen además *causas* o *condiciones* que propician cada uno de los estados de la naturaleza los cuales los denotamos por $c_1, c_2, \ldots, c_n$.
- Y por último tenemos los *efectos* o *acciones* que se seleccionaran: $a_1, a_2, \ldots, a_m$.

En la Tabla 4.3 se muestra el formato general de una Tabla de Decisión, esta tabla contiene m causas que se presentan en el problema denotadas por c_1, c_2, ..., c_m. Por cada causa se tiene n combinaciones posibles, y en cada combinación se pueden tomar distintos valores denotados por $V_i \; \forall \; i = 1, \ldots, n$ y finalmente se tienen dos acciones, a_1 y a_2.

Tabla 4.3: Formato general de una Tabla de Decisión.

		Combinaciones			
Causas	*Valores*	v_1	v_2	...	v_n
c_1	V_1	x_{11}	x_{12}	...	x_{1n}
c_2	V_2	x_{21}	x_{22}	...	x_{2n}
c_i	...	...	...	...	...
c_m	V_n	x_{m1}	x_{m2}	...	x_{mn}
Acciones					
a_1		$\times$		...	$\times$
a_2			$\times$	...	$\times$

En este tipo de representación matricial, se pueden mostrar todas las situaciones posibles asociadas a una problemática presente en una BD, junto con las acciones que se deben de tomar para cada situación. El objetivo de las Tablas de Decisión [Koh95], es la representación de las asociaciones o correlaciones de los objetos dentro de los datos. Lo que se busca es la construcción de un conjunto de reglas de asociación para todos los datos de entrenamiento presentes en los datos, algo diferente con respecto a los árboles de decisión los cuales consideran atributo por atributo cada vez que se desarrolla el árbol. El objetivo de las reglas de

asociación es encontrar asociaciones o correlaciones entre los elementos u objetos de bases de datos.

El funcionamiento de las Tablas de Decisión se basa en la estrategia de seleccionar reglas de decisión con un grado de confiabilidad mayor para un determinado valor establecido. Por lo que aquellas reglas que tienen un grado menor son descartadas para la tabla de decisión final, lo cual en la mayoría de los casos no es lo adecuado, ya que se puede presentar el problema de generalización. Este problema puede presentarse cuando omitimos estas reglas que en cierto modo puedan describir una porción de los datos de prueba, los cuales sin tenerlas lo más probable es que se realice una clasificación errónea. Una de las razones principales por las que se elimina las reglas de bajo grado de confiabilidad, es para evitar la complejidad del algoritmo con la presencia de grandes cantidades de datos.

Para la estimación de la complejidad de este algoritmo consideran tres factores, el número y valores de los atributos presentes en el conjunto del entrenamiento, el número de clases y finalmente el número de ejemplos del conjunto de entrenamiento. En el peor escenario estos tres factores se consideran para completar cada una de las posibles combinaciones en la tabla de decisión. Esto se puede representar mediante el número de combinaciones sin repetición en los datos, el cual esta denotado por la variable nPr la cual puede ser calculada por la Ecuación (4.1):

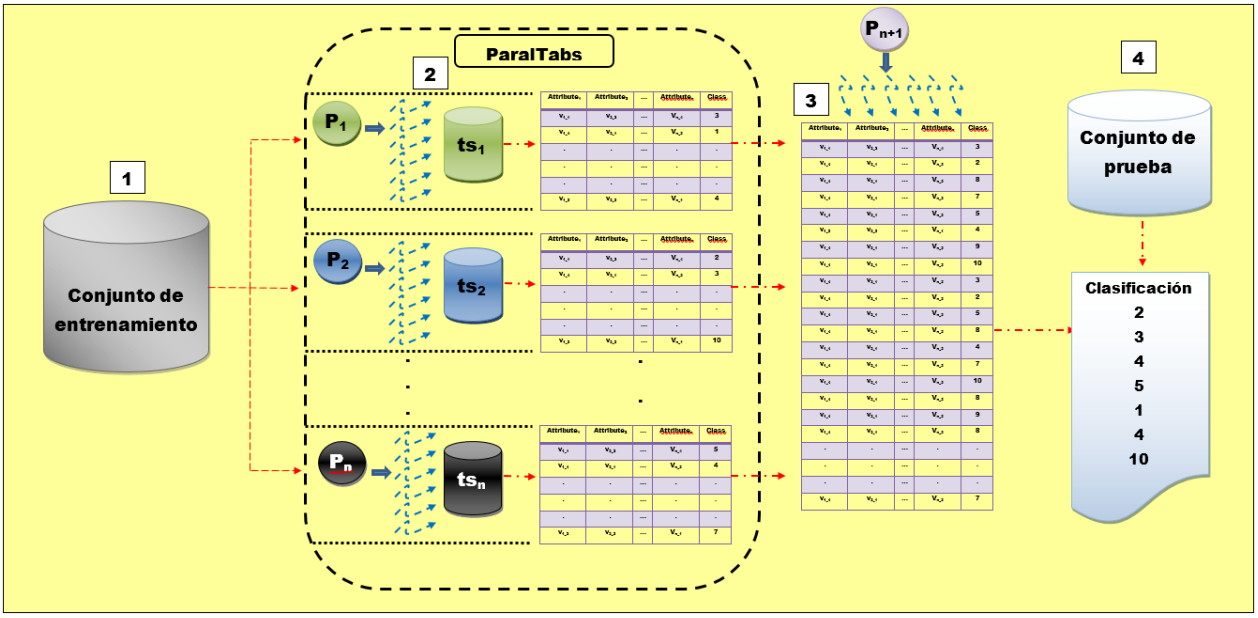
$$nPr = \prod_{i=1}^{n_a} \binom{NV_{a_i}}{1} \tag{4.1}$$

Donde, n_a es el número total de atributos en los datos y NV_{a_i} es el número de valores posibles para el atributo i, $\forall\ i = 1, 2, \ldots, n$. Mediante la Ecuación 4.1 podemos definir el número total de posibles combinaciones de los datos con n_a atributos. Sin embargo, si consideramos el segundo factor de la complejidad de las tablas de decisión, que es el número de clases, esto afecta en la complejidad del algoritmo, ya que ahora el número de combinaciones posibles aumenta tal y como se muestra en la Ecuación 4.2.

$$nPr = \left(\prod_{i=1}^{n_a} \binom{NV_{a_i}}{1} \right) \times n_c \tag{4.2}$$

Podemos ver que la complejidad aumenta cuando se tienen grandes cantidades de datos, ya que ahora se tiene un mayor número de posibles reglas de asociación para ser consideradas en la tabla de decisión final. Dado que en la literatura no encontramos trabajos en los que se tratara de solventar los principales problemas del esquema secuencial de las tablas de decisión, en un trabajo previo se desarrolló un esquema paralelo [MMMR13] el cual se utiliza la estrategia de divide y vencerás. En esta estrategia los datos de entrenamiento se dividen para reducir la complejidad de la construcción de la tabla de decisión, la cual considera un mayor número de combinaciones que en el esquema tradicional. En la Figura 4.8 se muestra el esquema paralelo propuesto para la construcción del clasificador Tablas de Decisión.

Figura 4.8: Esquema paralelo de Tablas de Decisión



En base al esquema de la Figura 4.8, el funcionamiento del esquema paralelo propuesto considera las siguientes etapas[MMMRMM17]:

1. División del conjunto de entrenamiento: En este esquema se tienen dos tipos de procesos, los procesos constructores (*processes builders - PB's*) y el proceso coordinador de la tabla de decisión (cDT). Para esta primera fase el conjunto de entrenamiento es dividido

por el cDT de acuerdo al número de PB's, el cual manda un mensaje global a cada uno de ellos con su respectivo conjunto de entrenamiento y prueba.

2. Generación de subtablas de decisión: Una vez que cada PB's obtiene su conjunto de entrenamiento, se generan de manera paralela subtablas de decisión considerando un número mayor de reglas de decisión debido a la reducción del tamaño de datos mediante los subconjuntos que se forman del paso anterior. Al momento que alguno de los PB's finaliza la construcción de sus subtablas de decisión, estos envían su tabla al cDT.

3. Construcción de la tabla de decisión global: Una vez que el cDT recolecta cada una de las subtablas de decisión, procede a construir la tabla de decisión global.

4. Clasificación del conjunto de prueba: En la última fase el cDT, se realiza la clasificación del conjunto de prueba mediante la tabla de decisión global.

Esta implementación representó un aporte novedoso, dado que en la literatura no encontramos ningún trabajo concerniente a la paralelización de esté clasificador. Se utilizó la arquitectura de programación SIMD, ya que una misma parte del algoritmo es ejecutada de manera paralela por los procesos presentes en el esquema, teniendo para cada proceso diferentes conjuntos de entrenamiento.

4.3.5. Esquema paralelo de Bayes Ingenuo

Este clasificador está inspirado en el teorema de Bayes, el cual maneja el concepto de probabilidades condicionales y *a priori* para llevar a cabo la clasificación de los datos. En este esquema se hace la suposición (que en la mayoría de los ejemplos reales no se presenta) de que todos los atributos son independientes entre sí dada la clase. El cálculo de las dos probabilidades utilizadas por este clasificador se define de la siguiente manera.

Sea $X = \{x_1, x_2, x_3, ..., x_d\}$ un vector de prueba al que queremos clasificar cada una de sus componentes y sean dos clases W_1 y W_2, con un tamaño N para el conjunto de entrenamiento. Por el Teorema de Bayes y tomando la suposición de que los atributos son independientes

entre si dada la clase, podemos expresar la probabilidad *a priori* y condicional de la siguiente forma:

Sus probabilidades *a priori* estarían dadas por:

$$P(W_1) = W_1/N \ , \ P(W_2) = W_2/N$$

Mientras que la probabilidad condicional $P(X|W_1)$ estaría definida por:

$$P(X|W_1) = P(x_1|W_1) \times P(x_2|W_1) \cdots P(x_d|W_1) \times P(W_1)$$
$$P(X|W_1) = P(W_1) \prod_{i=1}^{d} P(x_i|W_1)$$

De forma similar obtenemos la probabilidad condicional $P(X|W_2)$:

$$P(X|W_2) = P(x_1|W_2) \times P(x_2|W_2) \cdots P(x_d|W_2) \times P(W_2)$$
$$P(X|W_2) = P(W_2) \prod_{i=1}^{d} P(x_i|W_2)$$

Para definir a que clase pertenece el cálculo de estas dos probabilidades para cada uno de los ejemplos de prueba. Es por ello que cuando se presentan grandes cantidades de datos, en los que se tienen gran número de atributos, grandes conjuntos de prueba y clases a las pertenecen cada ejemplo, el tiempo de ejecución se ve fuertemente perjudicado en la fase de entrenamiento, ya que en la fase de prueba este tiempo es despreciable.

En este esquema paralelo que proponemos (esta implementación es propia) se realiza una división de los datos de entrenamiento para obtener una reducción en los tiempos de ejecución del esquema tradicional. Para la fase de entrenamiento una matriz es construida con las probabilidades condicionales y *a priori*. Para el caso de las probabilidades *a priori* no representa mayor problema el cálculo de las mismas, por lo que para este esquema paralelo se selecciona un proceso coordinador para llevar a cabo el cálculo de dichas probabilidades.

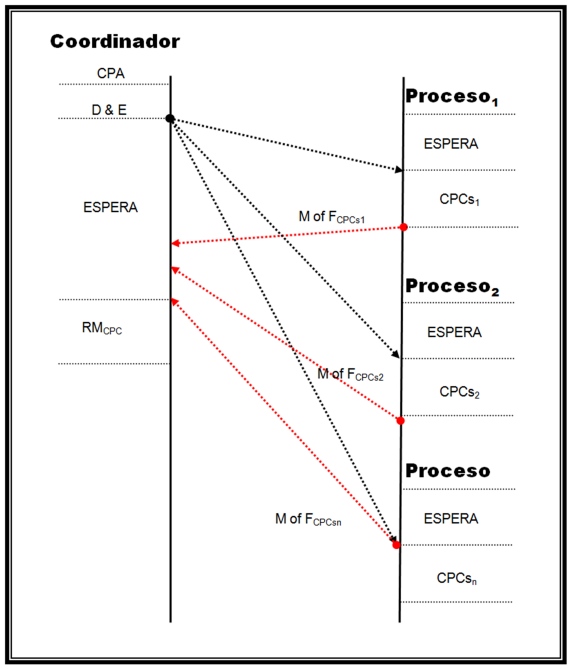
En el caso de las probabilidades condicionales, se realiza un alto número de operaciones cuando tenemos grandes cantidades de datos, ya que para cada uno de los ejemplos de entrenamiento se tiene que calcular las probabilidades condicionales de que un atributo con

un valor en específico, pertenezca o no a una de las clases presentes en los datos. Es por ello que en este esquema paralelo se asigna un proceso por cada atributo presente en los datos, los cuales llevan a cabo el cálculo de las probabilidades condicionales de que cada valor del atributo asignado pertenezca o no a una de las clases presentes.

Esto es, por un lado, el coordinador obtendrá los valores de las probabilidades *a priori* y los añadirá a la matriz de probabilidades. Por otro lado, este coordinador dividirá los datos en el número de atributos presentes, estos subrgrupos serán enviados a cada uno de los procesos que estarán encargados de llevar a cabo el cálculo de las probabilidades condicionales de cada atributo asignado.

Una vez que alguno de los procesos finalice el cálculo de las probabilidades condicionales del atributo asignado, enviará un mensaje al coordinador para recopilar toda la información necesaria para completar la matriz de probabilidades. Para ejemplificar la etapa de entrenamiento del esquema paralelo de Bayes ingenuo, utilizaremos el esquema de la Figura 4.9, el cual muestra el ciclo de vida de los dos procesos presentes.

Figura 4.9: Etapa de entrenamiento del esquema paralelo de Bayes ingenuo.



En la Figure 4.9 se puede observar que el coordinador tiene cuatro estados. Cálculo de las probabilidades *a priori* (*CPA*). Divide y envía (D & E), divide el conjunto de entrenamiento en el número de atributos presentes en los datos. El estado Espera es en el cual aguarda a que alguno de los procesos finalicen el cálculo de sus respectivas probabilidades condicionales. El último estado es para recibir los mensajes del cálculo de las probabilidades condicionales RM_{CPC}. Cuando el coordinador obtiene todos los cálculos de las probabilidades condicionales, procede a formar la matriz de probabilidades.

Dentro del esquema de la Figure 4.9, se puede observar que, para los procesos encargados de calcular las probabilidades condicionales, se tienen dos estados. En el estado Espera, los procesos aguardan el mensaje del coordinador con su respectiva porción del conjunto de entrenamiento. En el segundo estado se procede a calcular las probabilidades condicionales de cada subgrupo denotado por $CPC_{Si}\ \forall\ i = 1, 2 \ldots n$, donde n es el número de atributos presentes en los datos de entrenamiento.

Para la etapa de prueba, el coordinador mediante la matriz de probabilidades procede a clasificar cada uno de los ejemplos de prueba mediante el Teorema de Bayes, de esta manera se finaliza la ejecución de este esquema paralelo del clasificador Bayes ingenuo.

Como podemos ver, ya revisamos cada uno de los esquemas paralelos desarrollados para los clasificadores de base seleccionados en el PCEM. Por lo que resta revisar en este capítulo los pasos utilizados en el funcionamiento del PCEM, el esquema paralelo del algoritmo genético para encontrar las ponderaciones de cada clasificador de base y la implementación del criterio de votación ponderada en el cual veremos la comunicación entre cada proceso presente en el PCEM. Siguiendo este orden iremos describiendo cada uno de estos componentes del PCEM.

4.4. Funcionamiento del PCME

El PCEM consta de una serie de componentes que nos permiten tener el manejo de un conjunto de datos con diferentes características de origen. Con esto nos referimos a que los valores de los atributos tienen un formato especial, así como las fuentes de origen de los mismos. En algunas herramientas en las que se puede hacer uso de clasificadores del

aprendizaje maquinal como WEKA, SIPINA y TANAGRA, el usuario proporciona una base de datos sin preocuparse por el tipo de atributos que están presentes en ellas. Sin embargo, el formato de estas sí importa, ya que en algunos casos se debe tener una estructura especial para cada herramienta.

Por ejemplo, en el caso de WEKA, para el conjunto de entrenamiento se debe dejar un espacio en blanco entre cada atributo para que la herramienta distinga cuantos atributos existen. Una vez que la herramienta identifica cada uno de los atributos presentes en alguna base de datos, selecciona los posibles valores de cada uno sin importar que sean reales, enteros, cadenas de caracteres, etc.

En la mayoría de las herramientas de clasificación el manejo de las clases representa un caso especial. En algunas bases de datos las clases están implícitas en la última columna de los datos de entrenamiento. Dada esta situación las herramientas de clasificación establecen que, para tener un manejo adecuado de los datos, las clases deben localizarse en dicha columna para que el usuario tenga presente dicha situación.

Una vez que los datos sean proporcionados por el usuario, las herramientas de minería de datos seleccionan de manera interna el conjunto de entrenamiento y prueba para algún clasificador incorporado en la misma. Se procede entonces con la fase de entrenamiento seleccionando un método de validación para tener evaluar el correcto funcionamiento de cada clasificador.

Posteriormente se clasifica el conjunto de prueba, que en la mayoría de los casos se conoce la clase de cada ejemplo, para obtener los índices de las medidas de rendimiento. En el PCEM desarrollamos un funcionamiento similar al que tienen estas herramientas tradicionales de minería de datos.

En primera instancia pretendimos desarrollar el PCEM mediante la herramienta de programación Matlab, sin embargo, para su incorporación en el esquema de sistemas multicomputador, para ser más específico en un cluster, no represento la forma más viable de hacerlo por las licencias de la herramienta. Es por ello que decidimos utilizar la herramienta de programación llamada Octave. Esta herramienta tiene una sintaxis similar a Matlab, len-

guaje M, con la diferencia de que se trata de una herramienta de software libre. La elección de esta herramienta nos facilitó la incorporación sobre el cluster, ya que pudimos instalar la herramienta sin preocuparnos por asuntos de licencias de uso. En esta herramienta podemos hacer uso de cómputo paralelo mediante el uso de hilos (*threads*) y procesos que se comunican por medio de mensajes utilizando la herramienta de programación MPI Octave.

Retomando el funcionamiento las herramientas de minería de datos que nos permiten realizar clasificación de datos, para el caso del PCEM dividimos su funcionamiento en cuatro etapas, las cuales se pueden observar en la Figura 4.10.

En la Figura 4.10 podemos localizar cada una de las cuatro etapas que constituyen el funcionamiento del PCEM las cuales son: selección de los conjuntos de entrenamiento y prueba, clasificaciones individuales, esquema paralelo del algoritmo genético de ponderaciones y criterio de votación ponderado, estas serán revisados en las siguientes secciones del capítulo.

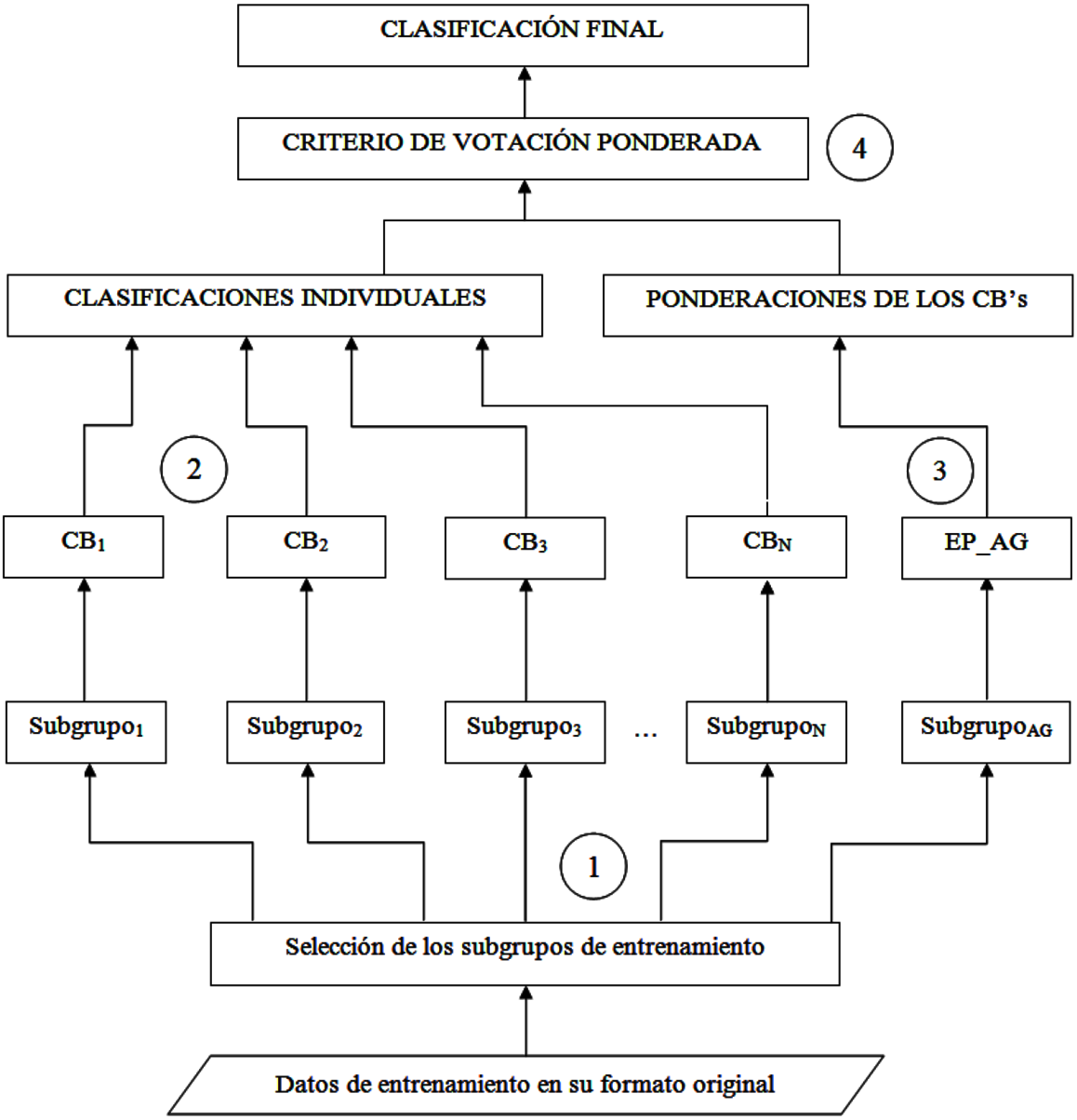
4.4.1. Selección de los subgrupos de entrenamiento

En este primer paso que podemos identificar del funcionamiento de la Figura 4.10, la mayoría de los datos tienen formatos diferentes en cuestión de valores de atributos, número y tipos de clases, valores faltantes, datos redundantes, entre otros más. Estos factores son considerados por la mayoría de las herramientas de clasificación en las cuales se considera un mecanismo interno para el manejo de estas características en los datos.

En el caso del PCEM nos apoyamos de la segunda fase del proceso KDD [Cor17], implementando técnicas de selección, limpieza y transformación de datos. Para ejemplificar esta componente del PCEM vamos a utilizar un tipo de base de datos en particular, la de micro arreglos genéticos.

La base de datos de micro arreglos genéticos tiene un formato especial llamado ISCN (An International System for Human Cytogenetic Nomenclature)), el cual corresponde a toda la nomenclatura que se maneja en la área de investigación de Genética Humana. Para este tipo de base de datos, estas nomenclaturas se reducen a un subconjunto que contiene la información de posibles alteraciones genéticas asociadas con algún tipo de cáncer.

Figura 4.10: Etapas de funcionamiento del PCME.



Esta base de datos ha sido recopilada por el Centro Médico Nacional Siglo XXI a lo largo de 10 años con el seguimiento de 50 pacientes. El archivo recopilado de los pacientes antes mencionados se obtiene de progenetix[2]. Un registro de esta base es de la siguiente forma:

L1 rev ish enh(1p36.22, 1p13.1, 1qtel, 2q14, 3p14.2, 3q26.3, 5q21q22, 5q33q35, 7q32q34, 8p22, 8p22q21.3, 8q24qter, 9q22.3, 14qtel, 15qtel, 16qtel, 17ptel, 17q23, 20q13.2, 20qtel, Xq12) dim enh(19ptel) amp enh(1q21, Xp22.3)

Como podemos ver esta información es totalmente desconocida ya que se encuentra en un formato especializado, como lo es el ISCN, sin embargo, cada sigla tiene un significado especial, vamos a descomponer cada una de las siglas que aparecen en este registro teniendo lo siguiente:

- **L1:** Etiqueta asignada al paciente.
- **rev, ish, enh, dim, amp:** Técnicas que se utiliza para obtener la información genética.
- **1p36.22:** Es el primer registro dentro del paréntesis, el número que lleva al inicio "1" es el número de cromosoma, "p" es el tipo de brazo en el que se encuentra en este caso "p" es el brazo corto, si se ocupara "q" se trataría del brazo largo, y por último "36.22" hace referencia a la zona del cromosoma donde se presenta el cambio.

Para cada uno de los datos presentes se identifica la alteración genética que está presente, sin embargo, existen pacientes que presentan diferente número de cromosomas involucrados, así como diferente número de técnicas utilizadas, por este motivo decidimos buscar una manera de tener una longitud de los registros del mismo tamaño. Para poder lograr este objetivo se tuvieron que implementar técnicas de numerización y discretización aplicadas directamente a esta base de datos y algunas otras que tuvieran formatos similares.

Esto lo realizamos pensando en los clasificadores de base que utilizamos en el PCEM, ya que en la mayoría de ellos su funcionamiento se adapta de mejor manera a estructuras de bases

[2]Esta herramienta proporciona una visión general de las anomalías de número de copias en el cáncer humano de Hibridación Genómica Comparada (CGH) para generar el cariograma y agrupamiento

de datos de tipo tabla. Vamos a describir cómo se realiza dicha trasformación del formato original para este tipo de datos, utilizando las técnicas de numerización y discretización.

Implementación de Técnicas de Numerización y Discretización

Como pudimos ver en el ejemplo anterior, existen términos que son utilizados exclusivamente en el formato ISCN, las cuales representan siglas de cada uno de los elementos que conforman un registro de este tipo de base de datos de micro arreglos genéticos, es aquí donde las técnicas de numerización y discretización fueron aplicadas para cambiar estas siglas por valores enteros los cuales son manejados de mejor forma por los clasificadores de base que utilizaremos para la funcionamiento del PCEM. Como primer punto desglosaremos cada una de las siglas que se encuentran presentes en los registros de este tipo de bases de datos.

Técnicas para cada paciente.

- rev ish enh
- dim enh
- amp enh
- amp
- dim

Cromosomas.

- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, X, Y

Tipo de brazo en el cromosoma.

- p: Brazo corto.
- q: Brazo largo.

Región en donde se presenta la alteración genética.

- Valores reales asociadas a cada uno de los cromosomas que pueden ir de la región terminal (ter) a la telomérica (tel). Por lo que al aplicar las técnicas de numerización y discretización se tiene que asignar una nomenclatura para cada uno de estos componentes, por lo que tendríamos lo siguiente.

Técnicas: rev ish enh amp dim.

- rev ish enh → 101
- amp enh → 102
- dim enh → 103
- amp → 104
- dim → 105

Cromosomas:

- Del 1, 2, ..., 22
- X → 23
- Y → 24

Brazos:

- P (Brazo Corto) → 300
- Q (Brazo Largo) → 301

Región:

- Valores reales multiplicados por 100
- ter → 4000
- tel → 4001

La asignación de estos valores se realizó de manera arbitraria, teniendo la trasformación del registro que anteriormente fue presentado a un registro en el que se aplican ambas técnicas de la fase de pre-procesamiento y transformación, veamos un ejemplo de ello:

Registro en el formato ISCN:

L1 rev ish enh 1 p 36.22 1 p 13.1 1 q tel 2 q 14 3 p 14.2 3 q 26.3 5 q 21 q 22 5 q 33 q 35 7 q 32 q 34 8 p 22 8 p 22 q 21.3 8 q 24 q ter 9 q 22.3 14 q tel 15 q tel 16 q tel 17 p tel 17 q 23 20 q 13.2 20 q tel X q 12 dim enh 19 p tel amp enh 1 q 21 X p 22.3

Después de aplicar técnicas de normalización y discretización:

101 1 300 3622 1 300 1310 1 301 4000 2 301 1400 3 300 1420 3 301 2630 5 301 2100 5 301 2200 5 301 3300 5 301 3500 7 301 3200 7 301 3400 8 300 2200 8 300 2200 8 301 2130 8 301 2400 8 301 4000 9 301 2230 14 301 4000 15 301 4000 16 301 4000 17 300 4000 17 301 2300 20 301 1320 20 301 4000 23 301 1200 104 19 300 4000 101 1 301 2100 23 300 2230

En este caso se puede observar que se cuenta con un solo tipo de datos enteros por lo que se facilita su manejo para los clasificadores que utilizaremos, sin embargo, se presenta un problema ya que el registro queda demasiado largo y que cada una de las técnicas tiene diferente longitud para cada uno de los registros con los que se cuenta. Por este motivo se pensó codificar estos registros de tal manera que se tengan cadenas del mismo tamaño separando tanto las técnicas como los cromosomas presentes por renglones.

Para mostrar cómo aplicamos el cambio en los registros para tuvieran un mismo tamaño escogeremos un registro más de la base de datos de micro arreglos genéticos el cual contiene la siguiente información:

L8T2N0 amp(19ptel, 22q11.21, 22q11.2, 22q13.3, 22qtel) dim(19ptel)

Como podemos observar en el registro anterior se cuenta con dos técnicas las cuales son *amp* y *dim*, este ejemplo nos ayudara a ilustrar cada uno de los pasos que se deben de realizar para transformar este formato ISCN al formato que maneja el PCEM[MMMRMM17].

Paso 1:

Se deben de localizar cuántas técnicas van estar presentes en el registro en cuestión, por lo que si retomamos el ejemplo anterior tendríamos lo siguiente:

- amp(19ptel, 22q11.21, 22q11.2, 22q13.3, 22qtel)
- dim(19ptel)

Paso 2:

Para cada una de las técnicas separadas se deben de descomponer cada una las alteraciones genéticas conservando el orden para cada una de las técnicas, para ejemplo tendríamos lo siguiente:

- amp
    1. 19ptel
    2. 22q11.21
    3. 22q11.2
    4. 22q13.3
    5. 22qtel
- dim
    1. 19ptel

Paso 3:

Ya que se tienen separadas las alteraciones genéticas se procederá a separarlas en número de cromosoma, tipo de brazo y zona del cromosoma en donde se presenta la alteración, respetando cada una de nuevas nomenclaturas establecidas, retomando el ejempló tendríamos lo siguiente:

- amp → 104

Orden	Cromosoma	Brazo	Región
1.	19	300	4001
2.	22	301	1121
3.	22	301	1120
4.	22	301	1330
5.	22	301	4001

- dim → 105

Orden	Cromosoma	Brazo	Región
1.	19	300	4001

Paso 4:

Una vez que se tienen ambas técnicas codificadas, cada una de estas representa un nuevo registro en la base de datos, por lo que en vez de tener solamente un solo registro ahora se tendrán 6 registros, pero todos del mismo tamaño, esto se realizará para cada uno de los registros hasta codificarlos en todos los registros de la base de datos, la codificación del ejemplo que planteamos sería la siguiente:

Técnica	Cromosoma	Brazo	Región	Orden
104	19	300	4001	1
104	22	301	1121	2
104	22	301	1120	3
104	22	301	1330	4
104	22	301	4001	5
105	19	300	4001	1

Podemos ver que dentro de este formato se tiene presente diferentes técnicas en los pacientes monitoreados, esto es una información útil ya que al tener nuevos pacientes se podrá contar con la presencia de cambios con técnicas similares, por lo que es conveniente agregar

nuevos atributos que nos arrojen nueva información, estos dos atributos más se muestran a continuación:

- Número de cambios. Para este atributo se contabilizan cuantas técnicas se utilizaron.
- Secuencia. Dado que cada una de las técnicas fue codificada en un rango de 101 a 105, para este nuevo atributo se utilizan la última cifra y se van juntando dependiendo de si utilizaron una o más técnicas. Retomando el ejemplo anterior se utilizaron la técnica *amp* y *dim*, que tienen asociado los valores 104 y 105 respectivamente, por lo que el nuevo atributo quedaría como 45.

Finalmente, una vez que ya se revisó cuáles son los atributos que se agregaron el registro de prueba quedaría de la siguiente forma:

Técnica	Número de Cambios	Secuencia	Cromosoma	Brazo	Región	Orden
104	2	45	19	300	4001	1
104	2	45	22	301	1121	2
104	2	45	22	301	1120	3
104	2	45	22	301	1330	4
104	2	45	22	301	4001	5
105	2	45	19	300	4001	1

Cabe mencionar que esta base de datos fue la más grande con la que corroboramos el funcionamiento del PCEM, ya que en el estudio que se realizó a lo largo de 10 años se generaron 3,000 registros en su formato original (ISCN). Una vez que se aplicó la transformación que se explicó anteriormente, esta base de datos resulto con 7,652,000 registros en un formato de tipo tabla.

Mediante las técnicas de selección, limpieza y trasformación implícitas en este módulo ofrecemos un manejo de diferentes valores de los atributos en las bases de datos que se utilicen para probar el funcionamiento del PCEM. Una vez que a lo datos en su formato original se les aplica este módulo, se procede a obtener los subgrupos de entrenamiento para cada uno de los componentes del PCEM.

Para el caso de los esquemas paralelos de cada clasificador de base (CB_1, CB_2, ..., CB_N) se utiliza un conjunto de entrenamiento y de prueba. En el caso del esquema paralelo del algoritmo genético de ponderación (*PGA - Parallel Genetic Algorithm*) solamente se selecciona un conjunto de entrenamiento mediante el cual podemos encontrar las ponderaciones de cada clasificador de base. Estas son las tareas que se realizan en esta primera fase del funcionamiento del PCEM.

4.4.2. Clasificaciones individuales

Una vez que los clasificadores de base obtienen su respectivo conjunto de entrenamiento y prueba, proceden a encontrar la clasificación individual del conjunto de prueba. Para poder explicar esta parte del funcionamiento del PCEM, vamos a utilizar un esquema en el cual se mostrará la ejecución de los procesos coordinadores. Por un lado, tenemos un coordinador general que se encargara de recopilar la información de la clasificación individual de cada clasificador de base y las ponderaciones obtenidas por el esquema paralelo del algoritmo genético. Estos dos componentes a su vez tienen coordinadores locales que tendrán comunicación con el coordinador general, así como una comunicación entre el coordinador local del *PGA* y los coordinadores locales de cada clasificador de base.

En importante resaltar que en el funcionamiento del PCEM radica en las fases de clasificaciones individuales y ponderación de pesos. El escoger una u otra para iniciar no afecta como tal el funcionamiento PCEM, sin embargo, existe una diferencia en cuestión de tiempos de ejecución, por lo que vamos a revisar algunos detalles sobre esto.

Para encontrar la ponderación de cada clasificador de base al *PGA*, el coordinador general le proporciona un porcentaje (que será detallado más adelante) del conjunto de entrenamiento. Con este porcentaje el *PGA* se da a la tarea de encontrar una solución que se aproxime más a la óptima para cada base de datos distinta. Es por ello que de las n-iteraciones que se lleven a cabo, solo basta con realizar una sola clasificación individual por cada clasificador de base ya que el objetivo es encontrar que pesos son los más convenientes.

Por otra parte tenemos la fase de las clasificaciones individuales, en donde mediante una

estrategia de validación cruzada, se clasifica el conjunto total de la base de datos, que en la mayoría de los casos es un conjunto 80 % más grande que el conjunto que seleccionamos para el algoritmo genético. Este proceso es el que consume más tiempo debido a que los clasificadores, a pesar de ser esquemas paralelos, siguen teniendo un tiempo mayor al que pudiera tener el *PGA*.

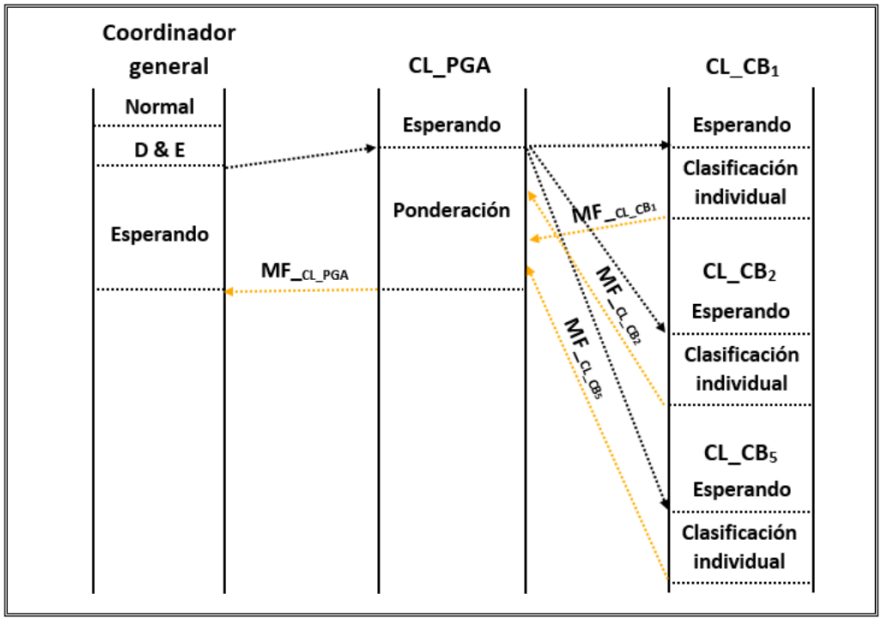
Es por ello que decidimos empezar con la búsqueda de las ponderaciones de cada clasificador de base, ya que es un fase que dura menos tiempo de ejecución. Dentro de esta primera fase existe una comunicación entre el coordinador local del *PGA* con cada clasificador de base. Esta comunicación inicia una vez que el coordinador general le proporciona el conjunto de entrenamiento y prueba al coordinador local del esquema paralelo del algoritmo genético. Este envía un mensaje global[3] a cada coordinador local de los clasificadores de base y estos a su vez le regresan la clasificación individual que obtienen del conjunto de entrenamiento.

Una vez que el coordinador local del *PGA* obtiene las clasificaciones individuales, se da a la tarea de buscar cual es la mejor ponderación para un base de datos en particular. Cabe resaltar que este proceso es necesario que se realice siempre que se cambia de base de datos, ya que cada base tiene diferentes características que hacen que los clasificadores puedan tener diferentes comportamientos. Esta parte de la comunicación inicial que se tienen con los componentes del sistema de clasificación paralela que proponemos en este trabajo se puede observar en la Figura 4.11.

En la Figura 4.11 podemos observar que tenemos tres tipos de coordinadores, el general del cual ya habíamos hablado anteriormente, las siglas *CL_PGA* que hace referencia al coordinador local del esquema paralelo del algoritmo genético y finalmente tenemos los coordinadores locales de cada clasificador de base denotados por las siglas $CL\_CB_i \ \forall \ i = 1, 2, 3, 4, 5$.

[3]Este mensaje global se realiza mediante el comando *MPI_Comm_spawn* definido en el MPI Toolbox de Octave (MPITB)

Figura 4.11: Fase para encontrar la ponderación de cada clasificador, con la interacción del coordinador general y los locales



El coordinador general tiene tres estados, el estado Normal es cuando se inicia el funcionamiento del PCEM. El estado *D & E*, se refiere a la etapa dentro del ciclo de vida del coordinador general en la que se generan los conjuntos de entrenamiento y prueba para ser enviados a cada uno de los coordinadores locales. El último estado es el de Esperando, en el cual el coordinador general espera el mensaje del *CL_PGA* o de cada uno de los *CL_CBi*. En el esquema se pone una flecha punteada de color naranja denotada por *MF_CL_PGA* y $MF\_CL\_CB_i \; \forall \; i = 1, \ldots, n$) para indicar que la etapa de encontrar las ponderaciones y obtener la clasificación individual del conjunto de prueba ha finalizado.

En la Figura 4.11, también pudimos observar la comunicación que existe entre el *CL_PGA* y cada uno de los CL_CB_i. Como explicamos anteriormente, esta comunicación es necesaria para que el *CL_PGA* conozca las clasificaciones individuales de cada clasificador de base y de esta manera pueda encontrar que configuración de pesos es la más adecuada para cada uno de ellos. En este caso el *CL_PGA* solo lo representamos con 2 estados, el estado Esperando es el que se tiene al inicio del funcionamiento del *PCEM* el cual hace referencia a la espera

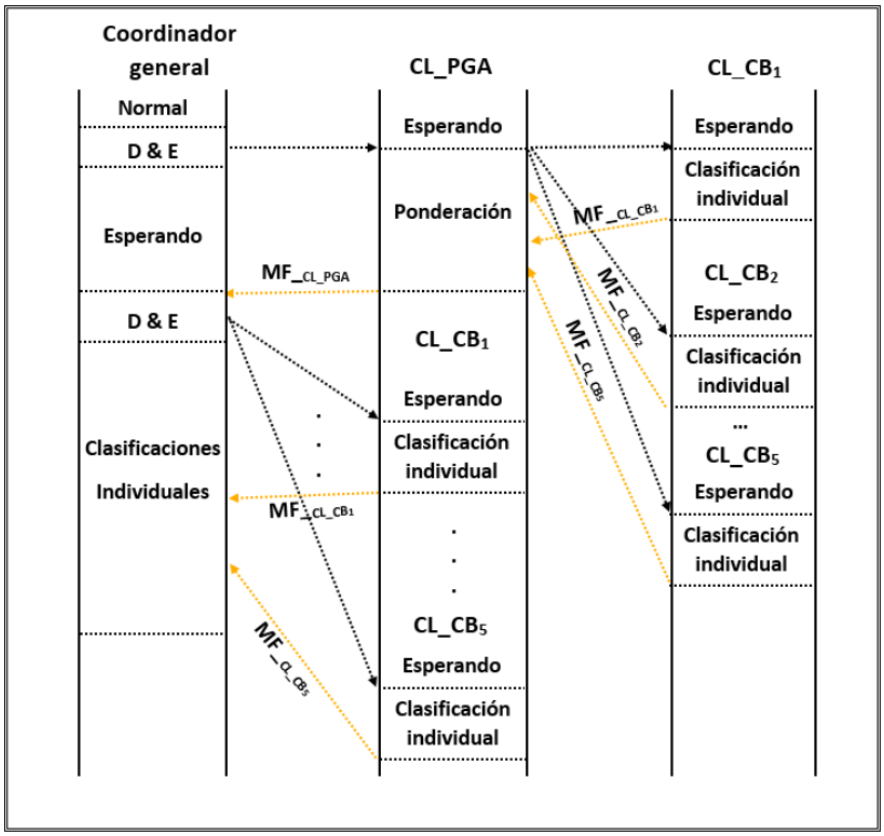
del conjunto de entrenamiento y prueba para que cada CL_CB_i obtenga las clasificaciones individuales.

No está presente un estado de Enviar pero se puede deducir que una vez que el CL_PGA obtiene estos dos conjuntos los pasa por medio de mensajes a cada uno de los clasificadores de base. Una vez terminado el envío el CL_PGA entra en un estado de Espera, hasta que todos los clasificadores de base finalicen la clasificación del conjunto de prueba. Una vez que se envían las clasificaciones individuales, el $PCEM$ se da a la tarea de encontrar las ponderaciones para cada clasificador enviándolas al coordinador general para que se continúe con la siguiente fase, que es la de encontrar las clasificaciones individuales del conjunto de prueba.

Es importante resaltar que, en esta fase para encontrar los pesos de cada clasificador, realizamos una porción de la siguiente fase que es la de encontrar las clasificaciones individuales de forma paralela, ya que, por medio del coordinador general les proporcionamos un conjunto de entrenamiento a los clasificadores de base con los cuales cada uno construye su modelo de clasificación. De esta manera en la fase de clasificaciones individuales su tarea solo será clasificar el conjunto de prueba. En la fase de clasificación individual los involucrados son el coordinador general y cada uno de los coordinadores locales de los clasificadores de base ($CL\_CB_i \ \forall \ i = 1, 2, 3, 4, 5$), la comunicación entre ambas entidades se puede observar en la Figura 4.12.

En la Figura 4.12 podemos ver la comunicación entre el coordinador general y cada uno de los coordinadores locales de los clasificadores de base. Para los coordinadores locales se tienen dos estados, Esperando y Primera fase. Para el estado Esperando, los coordinadores locales esperan su respectivo conjunto prueba para el caso de los clasificadores de base, esto debido a que en la fase de ponderaciones los modelos quedaron construidos en su totalidad, es por ello que esta fase se centra en lograr la clasificación individual del conjunto de prueba.

Figura 4.12: Fase de clasificaciones individuales, mediante la comunicación del coordinador general y los coordinadores locales de los clasificadores de base



Una vez que cada CL_CB_i recibe su respectivo conjunto de prueba, se procede al estado de clasificación individual. Al momento que cada CL_CB_i envié la clasificación individual, el coordinador con las ponderaciones realiza la clasificación ponderada del conjunto de prueba, obteniendo las medidas de rendimiento que el modelo obtiene para cada base de datos con la que se pruebe. Estas son las tareas que se permiten llevar a cabo el funcionamiento del *PCEM*. Hemos mencionado a grandes rasgos como se obtienen los pesos asignados a cada clasificador de base por medio del esquema paralelo del algoritmo genético de ponderaciones, sin embargo no hemos dado ningún detalle de la implementación de este componente que es de suma importancia para el funcionamiento del *PCEM*, es por ello que vamos a revisarlo con más detalle en la siguiente sección.

4.4.3. Esquema paralelo del algoritmo genético de ponderaciones

Para la selección de los pesos en los ensambles de tipo mezcla de expertos, normalmente se procede mediante el uso de una red neuronal simple como la que vimos en la sección 3.9. Existen diferentes formas de como poder asignar pesos a los clasificadores dado un análisis del comportamiento que puedan tener estos en una prueba previa a realizar la clasificación de los datos de prueba. Recordemos el funcionamiento del esquema paralelo de *boosting* revisado en la sección 3.6, veíamos que la construcción de cada aprendiz débil se forma con base al comportamiento de su predecesor, formando de esta maneja un conjunto de clasificadores con un mayor reforzamiento de su entrenamiento.

En el caso de los ensambles de tipo mezcla de expertos, tenemos diferentes modelos de clasificación que se ajustan de diferente forma a la distribución de los datos. En algunos casos algunos de los clasificadores de base pueden tener una mejor clasificación para una porción de datos en particular, por lo que sería lógico darle un mayor peso a su clasificación, tal y como se realiza en el algoritmo de *AdaBoost*, sobre los otros clasificadores de base presentes.

Esta asignación de pesos es un problema que puede ser resuelto por diferentes métodos tales como redes neuronales, métodos de optimización, algoritmos evolutivos y búsqueda exhaustiva. En particular, la búsqueda exhaustiva consiste en enumerar sistemáticamente todos los posibles candidatos para la solución de un problema, con el fin de comprobar si dicho candidato satisface la solución. Esta técnica nos permitirá introducir la complejidad en la búsqueda de los pesos para cada clasificador de base.

Supongamos que queremos hacer uso de esta técnica para encontrar los pesos de cada clasificador de base, en este caso se tendría que calcular las permutaciones sin repetición de escoger cinco posibles valores ya que se cuenta con cinco clasificadores de base. Supongamos que el intervalo de los posibles valores para cada clasificador lo fijamos en [0.5, 1, 1.5, 2, 2.5, 3, 3.5, ..., 19.5, 20], con lo cual podemos ver que tenemos 40 posibles valores para cada clasificador de base, por lo que el cálculo de las permutaciones sin repetición estaría dado por la Formula 4.3.

$$nPsr = \frac{n!}{(n-r)!} \tag{4.3}$$

Donde n es el número de los posibles valores que puede tomar cada clasificador de base que en este caso sería 40, y r representa el número de los clasificadores de base, que en este caso seria 5. Realizando el cálculo para este ejemplo tendríamos 78,960,960 posibles permutaciones sin repetición ($nPsr$). Podemos ver que tenemos un valor alto de las $nPsr$, el cual se incrementa al aplicarlo a la búsqueda de los pesos para cada clasificador, ya que tendríamos que calcular cada una de las posibles combinaciones de pesos al conjunto de prueba que seleccionáramos para esta búsqueda exhaustiva.

En algunas de las bases de datos se cuentan con más 7 millones de registros como la base de datos de micro arreglos genéticos, supongamos que para esta base de datos con un 10 % del tamaño de la base de datos se considera una muestra estadísticamente significativa, tendríamos que hacer $5.090904e^{13}$ comparaciones, lo cual claramente se puede ver que resultaría un problema muy complicado de realizar.

Otro de los aspectos importantes en la asignación de pesos es como definir el intervalo para aplicar la ponderación de clasificador presente. Una de las primeras cosas que podría pensar es que entre más grande sea el intervalo es mejor, sin embargo, esta conclusión es errónea, ya que podríamos sin darnos cuenta sobreajustar el ensamble, dándole pesos muy grandes a clasificadores que aporten una ventaja mínima. Supongamos que escogemos un intervalo del [1 al 100], si a cuatro de los cinco clasificadores por razones aleatorias del algoritmo para encontrar las ponderaciones, se le asigna un peso en el rango [1 al 20] y al quinto asignamos un peso en el intervalo [80 al 100], claramente podemos ver que en el mejor de los escenarios los cuatro clasificadores solo lograrían empatar la clasificación del más apto.

Aún si en un algoritmo genético o algún otro tipo de sistema de búsqueda bioinspirada o búsqueda exhaustiva encontrara una solución equitativa, el tiempo para darle solución sería muy alto y uno de los objetivos en el desarrollo del *PCEM* fue el hacerle frente a los altos tiempos de ejecución. Es por ello que lo que más conviene es crear intervalos en los cuales los valores no estén tan distantes y así obtener configuraciones más equitativas. Un ejemplo

de esto lo podemos observar en las ponderaciones que se les dan a la selección de rectores de alguna Universidad, en particular podemos mencionar el caso de la Universidad de Granada en España.

En el proceso de elección a Rector o Rectora, la Junta Electoral de la Universidad aplicará los coeficientes correspondientes al voto a candidaturas válidamente emitido en cada sector. El coeficiente de ponderación de cada sector o, en su caso, subsector es el cociente entre el porcentaje del sector o, en su caso, subsector y el número total de votos válidamente emitidos a candidatos o candidatas en cada sector o, en su caso, subsector.

Seguidamente, se determinará el porcentaje de votos ponderados de cada candidato o candidata en cada uno de los sectores o, en su caso, subsectores, que se hallará multiplicando el número de votos obtenido por cada candidato por el coeficiente de ponderación del sector o, en su caso, subsector. En esta última operación se realizará la aproximación a la milésima. La suma de las cantidades resultantes en cada uno de los sectores o, en su caso, subsectores determinará el porcentaje de votos ponderados obtenido por la persona candidata.

Claramente podemos darnos cuenta que existen muchos métodos solo para definir los intervalos de los pesos que les podríamos dar a los clasificadores, pero para nuestro caso decidimos fijar este intervalo de forma arbitraria en [0.5 al 5], esto es, se incrementaran los valores de 0.5 en 0.5 teniendo un total de 10 posibles valores. Si aplicamos la Formula 4.3, tenemos 30,240 de permutaciones sin repetición de como asignar los pesos a los clasificadores, lo cual no es un número grande como el que habíamos mencionado en el intervalo de tamaño 40 pero tampoco es tan despreciable considerando el tamaño de las bases de datos que se probaron.

Retomando el ejemplo que pusimos previamente, si tenemos una base de datos con un millón de registros de la cual tomamos un subgrupo del 10 %, en un método de búsqueda completo o exhaustivo tendríamos que realizar un proceso con más de 3 mil millones (3,024,000,000) de iteraciones para lograr una solución óptima. Aunque en el esquema de búsqueda exhaustiva este intervalo puede representar un alto costo computacional, con el *PGA* el objetivo es evitar este problema fijando este rango para las posibles ponderacio-

nes con lo cual por un lado no afectaríamos el tiempo de ejecución del *PCEM* y por otro lado el intervalo se encuentra comprometido lo que nos permite evitar obtener soluciones inadmisibles.

Como se puede observar el problema de asignación de pesos es un problema complejo debido a las posibles soluciones que se pueden tener, es por ello que para darle una posible solución a este problema se decidió utilizar métodos de Optimización Bioinspirada. Para este trabajo utilizamos un esquema paralelo de algoritmos genéticos de ponderación.

Los algoritmos genéticos (*Genetic algorithms - GAs*) fueron inventados por John Holland [Hol92] en los años 60, el cual junto con colegas y estudiantes de la Universidad de Michigan a lo largo de una década. En contraste con las estrategias evolutivas y la programación evolutiva, Holland planteo la meta de no diseñar a los algoritmos genéticos para resolver un problema en específico. Su objetivo radicó en él estudió formal del fenómeno de la adaptación que se produce en la naturaleza, para desarrollar formas en que los mecanismos de adaptación natural pueden ser importados a los sistemas informáticos.

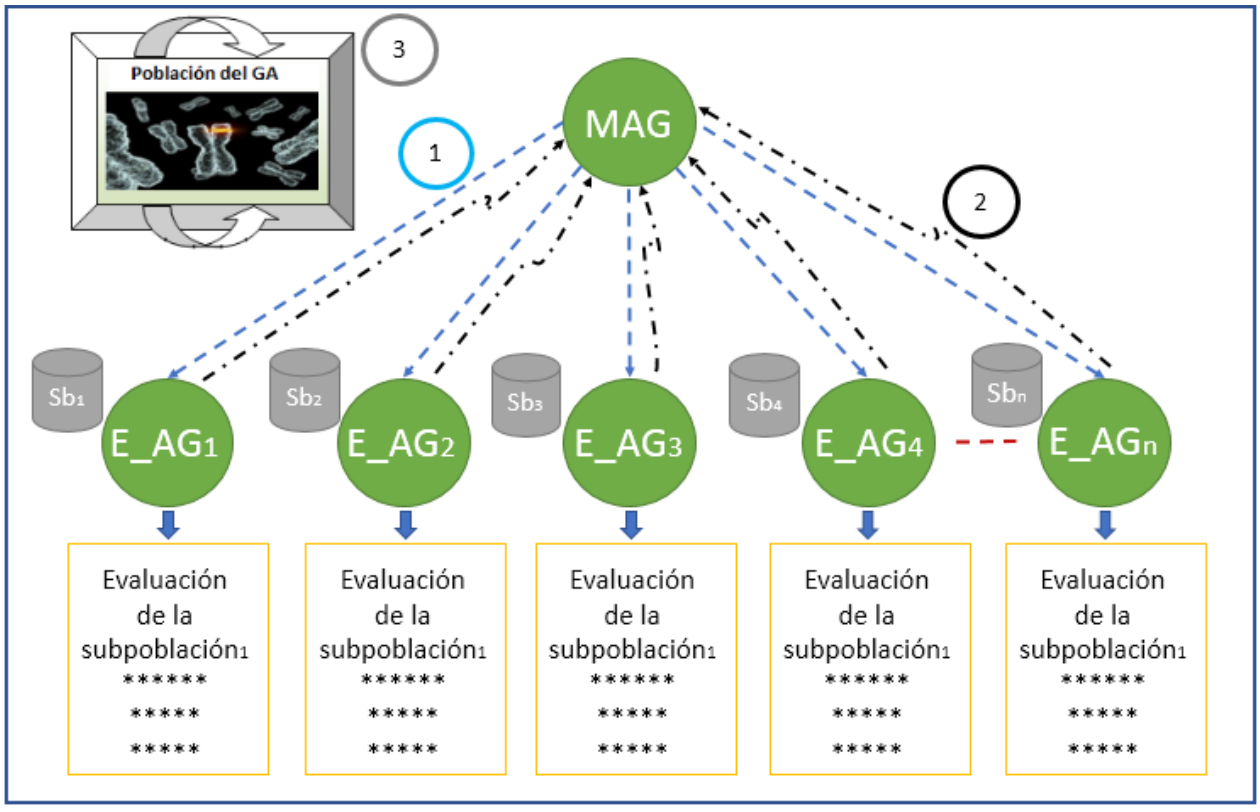
En el año de 1975, Holland presenta por primera vez a los GA en el libro *Adaptation in Natural and Artificial Systems* [Hol92], como una abstracción de la evolución biológica, dando así un marco teórico a la adaptación bajo un GA. En un principio este método se ejecutaba con una población de *cromosomas* (cadenas de enteros con unos y ceros, *bits*), los cuales evolucionaban formando nuevas poblaciones a lo largo de un número determinado de iteraciones, mediante el uso de una *selección natural*, utilizando operadores de inspiración genética como cruce, mutación e inversión.

Cada cromosoma constaba de un conjunto de *genes* (bits), empezando con una instancia de un *alelo* en particular (0,1). Los operadores de selección, eligen a los cromosomas de la población que se van a reproducir, los cuales en promedio son los más aptos a producir más descendencia que los menos aptos. Una vez generados los nuevos elementos de la población, se realiza un intercambio cruzado en subpartes de dos cromosomas, imitando la recombinación biológica entre dos cromosomas (*haploides*). El operador mutación, cambia aleatoriamente los valores de los alelos de algunas partes del cromosoma. Finalmente, la inversión invierte

el orden de una sección contigua del cromosoma. Mediante esta reordenación se cambia el orden en que están dispuestos en los genes.

Dado que el PCEM se realiza una paralelización global (cada uno de los componentes del PCEM son paralelos) se desarrolló un esquema paralelo de un algoritmo genético para no tener problemas de tiempos de ejecución con este componente del PCEM, el cual nos referiremos simplemente como el algoritmos genético de ponderación. En este caso seleccionamos una arquitectura paralela del algoritmo genético basada en la población simple global bajo el esquema maestroesclavo (*global single-population masterslave GAs*)tomando como referencia los trabajos de [CP98] y [PRSMGH17], esta arquitectura se muestra en la Figura 4.13.

Figura 4.13: Arquitectura del esquema paralelo del algoritmo genético



Podemos ver que en la arquitectura de la Figura 4.13 tenemos 2 componentes, por un lado, tenemos el proceso maestro denotado por las siglas *MAG* (Maestro del Algoritmo Genético)

y por otro lado, tenemos una serie de procesos que serán los esclavos o la parte distribuida del esquema paralelo que utilizamos lo cuales estan denotado por las siglas $E\_GA_i \ \forall \ i = 1, 2, \ldots, n$.

Dentro de la Figura 4.13 también se pueden identificar 3 pasos que conforman el funcionamiento del algoritmos genético de ponderación, los cuales se describen a continuación:

1. El paso 1 denotado con un circulo de color azul, hace referencia a la distribución de las subpoblaciones que son distribuidas por parte del MGA a cada uno de los E_GA_i. Este esquema paralelo que utilizamos se basa en que el maestro distribuye grupos de población global a cada uno de sus esclavos, para que estos evalúen cada uno de los cromosomas con la función de aptitud establecida, que para este caso es la medida de rendimiento llamada exactitud. La comunicación se puede observar en la Figura 4.13 con las lineas rectas de color azul.

   Decidimos paralelizar solo esta parte del algoritmo genético, sin embargo, el cruzamiento y la mutación pueden ser paralelizados usando la misma idea de dividir a la población y distribuir el trabajo entre múltiples procesadores. Sin embargo, estos operadores son tan simples que es muy probable que el tiempo requerido para enviar a los individuos de ida y vuelta compensaría cualquier aumento de rendimiento [CP98].

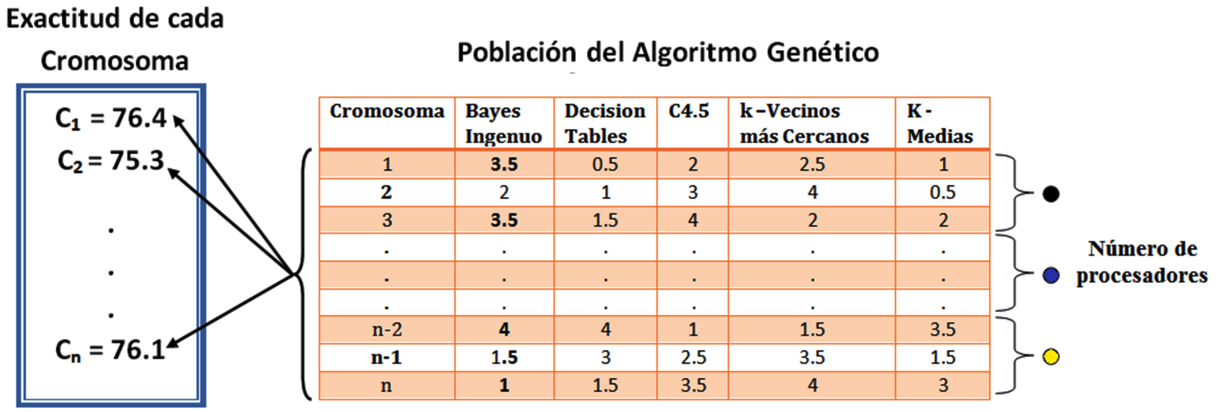
2. En el segundo paso denotado por un circulo de color negro, después de que cada uno de los E_GA_i recibe su población, estos proceden a evaluar cada uno de los cromosomas y regresan dicha evaluación al maestro. La comunicación se puede observar en la Figura 4.13 con las lineas curvadas de color negro.

3. El paso 3 denotado con un circulo de color gris, consiste en que el MGA en base a las evaluaciones que obtuvieron sus E_GA_i aplica los operadores genéticos de cruza, mutación y selección para generar una nueva población y de esta manera repetir los pasos 1 y 2. El criterio de paro que se estableció esta dado por un umbral entre las diferencias de poblaciones pasadas y actuales el cual sera descrito más adelante.

Una vez que se definió cual fue la arquitectura seleccionada para el algoritmo genético de ponderación y los pasos para llevar su funcionamiento, vamos a describir a más detalle la representación de los cromosomas, el criterio de paro y los operadores genéticos que se seleccionaron en este trabajo.

Cada población en el algoritmo genético de ponderación representa una combinación diferente de pesos para cada clasificador. La codificación de cada cromosoma, se compone de diferentes pesos en el rango definido anteriormente. Como se escogieron cinco clasificadores de base, el tamaño de cada cromosoma es de cinco. La función de aptitud que se escogió para la evaluación de cada cromosoma, es el valor de exactitud que tiene cada cromosoma. Cabe resaltar que para obtener los cromosomas tomamos una porción del conjunto de entrenamiento de entre el 10 % y el 15 % en base a una prueba estadística que mencionaremos más adelante, en el esquema de la Figura 4.14 se muestra la representación del cromosoma.

Figura 4.14: Representación de los cromosomas en el algoritmo genético de ponderaciones



Podemos ver en la Figura 4.14 que para este esquema paralelo la población, se divide entre el número de procesadores presentes en el cluster, por lo que si tenemos N individuos y M procesadores se tendría que dividir la población total en subpoblaciones de N/M individuos cada una, llegando a tener poblaciones de más de 1000 individuos. Para cada uno de los procesadores se ejecuta un determinado número de procesos, por simplicidad en esta primera

versión del esquema paralelo se ejecuta un proceso por cada procesador presente en el nodo del cluster que seleccionado.

Una vez que se asigna una porción de la población a cada procesador, se ejecuta el algoritmo genético de manera paralela tal y como se realiza en un esquema tradicional. Es por ello que describiremos cada uno de los componentes que seleccionamos para el funcionamiento del algoritmo genético de ponderación.

En la Figura 4.14 podemos observar que cada uno de los cromosomas se evalúa para obtener la función de aptitud que tiene cada uno de ellos. En este caso no se puede tomar una porción al azar del conjunto de entrenamiento que es asignado para el algoritmo genético de ponderación, porque esto sería una decisión sin fundamento teórico.

Para definir el apropiado porcentaje del conjunto de entrenamiento para el algoritmo genético de ponderación, que para cada base de datos es diferente, se utilizó una prueba basada en la varianza de una muestra de ensayo [HO85]. Considerando un nivel de confianza del 0.95, un error máximo del 0.1 y mediante un valor de varianza de 154.5, el posible tamaño de la muestra esta definido de la siguiente forma:

$$n^{'} = \frac{z^2_{\alpha/2} \cdot \sigma^2}{e^2}$$

Donde $n^{'}$ es el posible tamaño de la muestra, $z^2_{\alpha/2}$ es el grado de nivel de confianza seleccionado, σ^2 es la varianza teórica y e^2 el error máximo permitido. Si se cumple que $N > n^{'}(n^{'} - 1)$, donde N es el tamaño total del conjunto de datos, se toma el tamaño $n^{'}$ como el tamaño de la muestra, de lo contrario se calculará un nuevo tamaño de la muestra n, tal y como se muestra a continuación:

$$n = \frac{n^{'}}{1 + \frac{n'}{N}}$$

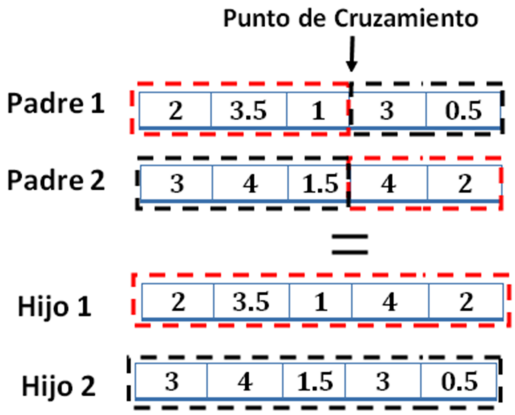
Este es el valor que calculará el coordinador general, para asignarle un conjunto de entrenamiento al algoritmo genético de ponderación. Para darnos una idea mucho más puntual del porcentaje que representa el conjunto de entrenamiento del algoritmo genético de ponderación, vamos a retomar el tamaño de la base de datos de micro arreglos genéticos con

7,652,000 registros. Con base a esta prueba estadística, con 1,034,894 registros se asegura un correcto funcionamiento del algoritmo genético de ponderación, el cual equivale a un 13.5 % del tamaño total del conjunto de prueba.

El método que se utilizó para la selección fue el de la ruleta, en donde se hace una semejanza con la ruleta utilizada en los casinos, la cual se gira n veces (n es el número de cromosomas de la población). Para representar los resultados de la ruleta se genera una sucesión de números r en el rango $[0, \ldots, 1]$. Para cada valor r se selecciona un cromosoma para obtener una nueva población (en dependencia de las probabilidades acumuladas).

Para la cruza se utilizó un valor pc de probabilidad del 0.5, la cruza se lleva a cabo de manera simple en un punto, el funcionamiento de este operador genético se puede observar en el esquema de la Figura 4.15.

Figura 4.15: Operador cruza



Se puede ver el proceder del operador cruza, en el cual se escoge una posición $PC = [1, \ldots, n]$ que va a corresponder al punto de cruce. Posteriormente se genera un número aleatorio s en el rango $[0, \ldots, 1]$ por cada cromosoma, el cual se va a comparar con pc y si es menor el cromosoma será un candidato a ser cruzado.

El operador de mutación habitual cuando no se representan a los cromosomas como cadenas de bits, puede dar lugar a tener soluciones inadmisibles, por lo que los operadores de mutación deben cambiar al menos dos elementos del cromosoma. En este caso para represen-

taciones enteras y reales, como la que presentamos en este trabajo de investigación doctoral, tenemos dos formas de llevarlas a cabo las cuales se enlistan a continuación:

1. Inserción: En este tipo de mutación se eligen dos elementos del cromosoma de forma aleatoria y después se coloca el segundo justo después del primero. Este tipo de operadores se utiliza cuando los elementos del cromosoma del cromosoma tienen un orden en cuestión de los valores presentes.

2. Intercambio: Para esta opción se seleccionan dos alelos de forma aleatoria y después se intercambian.

Se puede observar que cada uno de los operadores de mutación que mostramos anteriormente afecta de forma diferente al orden relativo y a las relaciones de adyacencia. Otra de las cosas que hay que mencionar es que estos operadores se aplicar normalmente a cromosomas con longitudes pequeñas, sin embargo, existen otros dos tipos de mutaciones si tuviéramos cromosomas de longitud grande, los cuales son los siguientes:

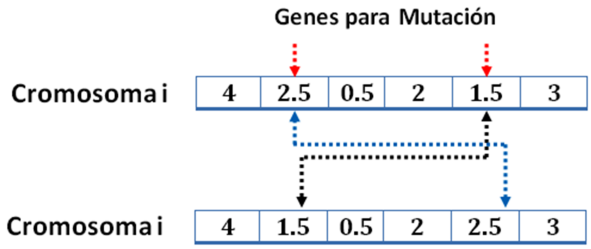
1. Inversión: Para esta opción se deben seleccionar dos elementos del cromosoma aleatoriamente y posteriormente se invierten los elementos de la subcadena que se forma.

2. Revuelto (*scramble*): En este caso se selecciona un subconjunto de genes para posteriormente reordenar aleatoriamente los alelos (el subconjunto que se selecciona puede no ser contiguo).

Para el trabajo de investigación doctoral decidimos utilizar el modelo de mutación por intercambio, aplicándolo en dos puntos de un cromosoma elegido de forma aleatoria, la probabilidad de mutación que escogimos fue de $pm = 0.12$[4] por lo que se espera que $pm \times 100\,\%$ de la población cambie de valor. Para cubrir cada uno de los cromosomas en un población se deben de generar *# de Cromosomas* $\times$ *# de genes* números aleatorios en el rango $[0, \ldots, 1]$. Los cuales serán comparados con pm para saber si un gen cambiará o no, en la Figura 4.16 se muestra el procedimiento de este operador genético.

[4]Este valor lo obtuvimos de un trabajo previo en el cual se desarrolló el *Hybrid Classifier with Genetic Weighting*[MMMR11]

Figura 4.16: Operador mutación



En el algoritmo genético de ponderaciones se aplica el criterio de elitismo para que el algoritmo converja en probabilidad al óptimo global. Lo que resta por describir es el criterio de paro que utilizamos para este algoritmo genético de ponderación. Del trabajo de Dinabandhu, et al. [BMP12], tomamos nuestro criterio de paro para el algoritmo genético de ponderación el cual se enlista a continuación:

1. Una población de soluciones aleatorias es creada y se genera un valor ϵ en un rango de $[0, \ldots, 1]$.
2. Cada solución es evaluada con base a la función de aptitud.
3. Almacenar la mejor solución, si esta es mejor que la anterior mejor (elitismo).
4. Calcular la diferencia de la mejor solución obtenida hasta ahora.
5. Si la diferencia es mayor que el valor predefinido de ϵ, seguir con el siguiente paso, otro caso terminar el algoritmo.
6. Una nueva generación de soluciones es creada a partir de una generación vieja usando selección, cruzamiento y mutación.
7. Los pasos 2 al 6 anteriores se repiten hasta que la condición en el paso 5 se satisface.

Podemos ver que este criterio de paro se basa en el cálculo de la diferencia para un esquema en el que se utiliza el criterio de elitismo, el cual nos dio mejores combinaciones de

peso con respecto al método tradicional de paro de los algoritmos genéticos, que es usar un número determinado de iteraciones. El valor de ϵ que mejor se ajustó para nuestras pruebas fue de 10^{-4}, obteniendo en promedio 42,567 iteraciones para bases de datos de más de un millón de registros con poblaciones en un rango de 50 a 100 individuos.

Hay que señalar que para el criterio de selección que utilizamos en este esquema paralelo del algoritmo genético que planteamos en este trabajo de investigación doctoral, utilizamos por una parte *fitness* o la función de evaluación y por otro lado las veces que se repite el elemento más apto el cual en cada una de las iteraciones, sea padre o hijo. Ha sido demostrado que un algoritmo genético requiere del elitismo para poder converger al óptimo global, tal y como lo podemos ver en el trabajo de Günter Rudolph [Rud94].

Esta es la parte final de la revisión del funcionamiento del algoritmo genético de ponderaciones, por lo que resta revisar la última fase del funcionamiento del PCEM. En esta fase revisaremos la comunicación de cada componente del PCEM para la aplicación del criterio de votación ponderada.

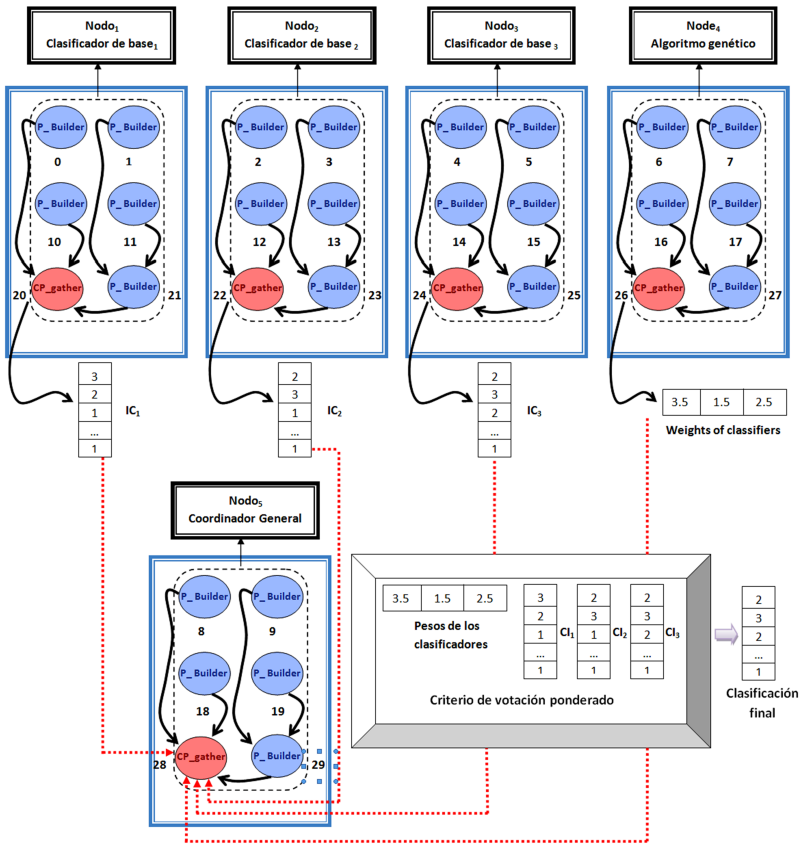
4.4.4. Combinación de las clasificaciones individuales

Una vez que el coordinador general recibe las clasificaciones individuales y los pesos encontrados por el algoritmo genético de ponderación, se procede a implementar el criterio de votación ponderado para obtener la clasificación final del conjunto de prueba. Para describir la comunicación entre cada uno de los componentes del PCEM, vamos a utilizar el esquema de la Figura 4.17.

En donde:

- *Clasificador de base*$_i$ $\forall\ i = 1, 2 \ldots n$, son todos los esquemas paralelos de los clasificadores de base del PCME.
- *Nodo*$_i$ $\forall\ i = 1, 2 \ldots m$, son los m nodos disponibles en el sistema multicomputador.
- *P_Builder*, son el conjunto de procesos en cada nodo encargados de llevar a cabo el funcionamiento de los clasificadores de base, el algoritmo genético de ponderación y el

Figura 4.17: Communication of PCME.



coordinador general.

- *CP_gather*, son los coordinadores locales de cada componente del PCEM.
- $CI_i \ \forall \ i = 1, 2 \ldots n$, son las clasificaciones generadas por cada clasificador.

Para ilustrar la comunicación del PCEM, vamos a suponer que tenemos tres clasificadores de base, el algoritmo genético de ponderación y el coordinador general, en los cuales se ejecutan ocho procesos por cada nodo. En la Figure 4.17 se puede observar que en cada nodo se efectúa una comunicación local que está representada por la línea de color negro. Mediante esta comunicación local los P_Builders envían su respectiva parte de la información generada a los CP_gathers.

Una vez que cada CP_gathers obtiene las CI y los pesos asignados a cada clasificador, estos envían un mensaje (líneas punteadas) al CP_gather del coordinador general. Con esta información el coordinador general aplica el criterio de votación ponderado, mediante un conjunto de P_Builders, a fin de encontrar la clasificación final del conjunto de prueba. El coordinador general a su vez también se encarga de calcular las medidas de rendimiento para evaluar el funcionamiento del PCEM, la cual representa la última fase de la operación del PCEM.

Una vez revisado en este capítulo a detalle cada uno de los componentes del PCEM, en el siguiente capítulo revisaremos los experimentos y resultados que obtuvimos al implantar el PCEM sobre un conjunto de bases de datos reales y sintéticas, los cuales los comparamos contra un conjunto de clasificadores tradicionales y paralelos.

Capítulo 5

Experimentos y Resultados

5.1. Introducción

En este capítulo mostraremos los experimentos y resultados que obtuvimos al utilizar el PCEM sobre un conjunto de bases de datos reales y artificiales. Los resultados obtenidos con el PCEM los comparamos contra un conjunto de clasificadores tradicionales y paralelos, a fin de obtener los principales aportes que ofrece nuestro sistema de clasificación paralelo con respecto a estos esquemas de clasificación. Todos los experimentos fueron realizados en el cluster Pacífico del cual utilizamos 7 nodos que contienen una unidad de procesamiento simple de tipo AMD FX 6300 con 6 nucleos or cores a una velocidad de 3.5 Ghz con 16 Gb en memoria RAM, cada uno de estos con una comunicación InfiniBand DDR y Gigabit Ethernet.

Primero mostraremos las bases de datos seleccionadas para llevar a cabo los experimentos, de éstas veremos que en algunos casos tenemos bases de datos del mundo real. Estas bases de datos tienen diferentes distribuciones, valores de atributos, diferentes tamaños y número de clases presentes.

Para los resultados comenzaremos con las principales medidas de funcionamiento utilizadas en aprendizaje maquinal, en particular mostraremos los resultados de exactitud, precisión, mejora y memoria. Una vez que veamos estos resultados, mostraremos una prueba teórica llamada *t-test* con la cual pudimos constatar que los resultados obtenidos de las medidas de funcionamiento son significativamente mejores con respecto a los esquemas tradicionales.

Revisados los resultados de las medidas de funcionamiento, procederemos con los experimentos realizados con respecto a los tiempos de ejecución. En este caso revisaremos los resultados de los tiempos de ejecución del PCEM así como de cada esquema paralelo de los clasificadores de base implementados. Finalizaremos este capítulo con un análisis tiempos de ejecución vs. índices en medidas de rendimiento que ofrece el PCEM con respecto a los esquemas paralelos de los clasificadores tradicionales que seleccionamos para los experimentos.

5.2. Bases de datos utilizadas

En la Tabla 5.1 se muestran cada una de las bases de datos utilizadas para realizar los experimentos.

De la Tabla 5.1 podemos ver que tenemos diferentes tipos de datos, en particular para el caso de la base de datos 1 y 2, se trata de la base datos de micro arreglos genéticos revisada en las secciones 2.7.1 y 4.4.1. La base de datos 7 fue proporcionada por la Empresa Gemius [Com07] dedicada al monitoreo de Internet sobre Europa central y oriental para el ECML PKDD del 2007[20007]. El resto de las bases de los datos de la Tabla 5.1 se tomaron del repositorio de la Universidad de Irvine en California [BM98].

Podemos ver en la Tabla 5.1 que tenemos bases de datos grandes y pequeñas, esto se debe a que quisimos probar la hipótesis que presentábamos en la sección 3.2, de emplear ensambles cuando tenemos pequeñas y grandes cantidades de datos. Una vez que presentamos las bases de datos con las cuales realizamos los experimentos, en la siguiente sección revisaremos cuales fueron las medidas de rendimiento que utilizamos para evaluar el desempeño del PCEM así como el método de validación para los experimentos.

En la mayoría de las bases de grandes cantidades de datos que presentamos en la Tabla 5.1, nos enfocamos en presentar bases de datos con un gran número de ejemplos, sin embargo, también se considera una base de datos grande cuando se tiene un gran número de atributos. Para probar con una base de datos de estas características escogimos la base de datos llamada *p53 Mutants* del repositorio de la Universidad de California en Irvine. Esta base de datos cuenta con 5,409 atributos de tipo real y 16,772 ejemplos.

Tabla 5.1: Bases de datos utilizadas para realizar los experimentos sobre el PCEM

Acrónimo	Nombre	Número de clases	Número de registro	Número de atributos	Año
BD_CaLu	Base de datos de Cáncer de laringe	2	7,652,000	7	2013
BD_RLC	Record Linkage Comparison (Cancer)	2	5,749,132	12	2011
BD_KDD99	KDD Cup 1999 Data	14	4,000,000	42	1999
BD_YT	YouTube Comedy Slam Preference	2	1,138,562	3	2012
BD_PH	Poker Hand	10	1,025,010	11	2007
BD_Cover	Covertype	7	581,012	54	1998
BD_EMCL07	EMCL PKDD 2007 Gemius Data	2	379,485	8	2010
BD_LDPA	Localization Data for Person Activity	10	164,860	8	2010
BD_BanMark	Bank Marketing	4	45,211	17	2012
BD_p53	p53 Mutants	53	16,772	5409	2010

5.3. Medidas de rendimiento y método de validación

En aprendizaje maquinal tenemos diferentes medidas de rendimiento para evaluar el desempeño de los clasificadores, en este trabajo vamos a utilizar las medidas de rendimiento definidas a partir del siguiente esquema, en el cual se muestra la matriz de confusión [WFH11]. Esta matriz de confusión contiene cada una de las posibles clasificaciones para un problema de clasificación con dos clases (A y B). Teniendo esta información se pueden identificar los ejemplos correctamente clasificados o también llamados verdaderos positivos (a + y d +) y los ejemplos que son clasificados incorrectamente o falsos positivos (c - y b -).

	Predicción Clase A	*Predicción Clase B*
Verdadera Clase A	**a +**	b -
Verdadera Clase B	c -	**d +**

Mediante estos dos tipos de ejemplos se pueden calcular las cuatro medidas de rendimiento de aprendizaje maquinal.

$$Exactitud = \frac{(a+d)}{(a+b+c+d)}$$

$$Precisión = \frac{a}{(a+c)}$$

$$Memoria = \frac{a}{(a+b)}$$

$$Mejora = \frac{a/\,(a+b)}{(a+c)\,/\,(a+b+c+d)}$$

En los experimentos utilizamos la validación cruzada de 10 iteraciones (*10-fold cross-validation*). Por cada iteración en esta validación se utiliza un subgrupo para el conjunto de prueba y el resto de los datos conforman el conjunto de ejemplo. Este proceso se ejecuta un número definido de iteraciones para completar la clasificación de todos los datos de la base de datos que se seleccione. En la siguiente sección mostraremos los resultados de las cuatro medidas de rendimiento obtenidos al comparar el PCEM contra un conjunto de clasificadores tradicionales y paralelos, utilizando todas las bases de datos de la Tabla 5.1

5.4. Resultados de las medidas de rendimiento

La exactitud (*accuracy*) nos dice cuántos ejemplos de las clases presentes fueron clasificados de manera correcta (positivos de ambas clases) del conjunto total de ejemplos. La

precisión (*precision*) expresa el porcentaje de los positivos de cada clase presente en los datos. Para la medida llamada memoria (*recall*), retomando la matriz de confusión, se tienen un conjunto de ejemplos clasificados, tanto positivos como negativos, como parte de una de las clases presentes en los datos (a + y b -). Con base a estos ejemplos la memoria nos ofrece el porcentaje de los ejemplos positivos del conjunto de clasificados como parte de una clase en particular, ya sean positivos o negativos. Finalmente, la mejora (*lift*) se puede entender como una proporción de dos porcentajes: el porcentaje de clasificaciones positivas correctas hechas por el modelo y el porcentaje de clasificaciones positivas de los datos de la prueba. Los resultados de estas cuatro mediadas de funcionamiento los presentamos en la Tabla 5.2, en donde para el caso de la exactitud se muestra la varianza obtenida para cada base de datos.

Cabe resaltar algunos puntos importantes de los resultados que se mostraran en la Tabla 5.2, los cuales se enlistan a continuación:

1. El *HCGW* es la versión secuencial del ensamble de tipo mezcla de expertos que decidimos tomar como referencia comparativa para el PCEM, dada las características similares que se tienen el uno del otro. Sin embargo, la diferencia más representativa radica en que todos los clasificadores de base del PCEM son versiones paralelas que en algunos casos mejoran los índices de las medidas de rendimiento dela versión secuencial. Es por ello que en los resultados que mostramos en la Tabla 5.2 se obtienen mejorías en todas las bases de datos con respecto al *HCGW*, esto será analizado más adelante en este capítulo.

2. Se muestra el resultado con respeto a los tres tipos de ensambles que fueron revisados en el capítulo 3, *AdaBoost*, *Bagging* y *Stacking* . Para cada uno de ellos utilizamos la versión secuencial que nos proporciona la herramienta WEKA. Se esta consciente que esta comparación no es equitativa ya que en la literatura se reportan trabajos sobre esquemas paralelos de estos tres ensambles, sin embargo, tratar de implementar todos basándonos en las descripciones de los autores es un trabajo complicado y que en su momento se descartó.

3. Para el caso de los esquemas paralelos del clasificador C4.5 y SVM, tomamos estos clasificadores de la herramienta llamada *ECL-ML Machine Learning Module*[1].

4. En el caso de las medidas de rendimiento precisión, mejora y memoria mostramos un promedio de los resultados para todas las clases presentes en cada base de datos probada. Finalmente, todas las pruebas se llevaron a cabo mediante un esquema de validación cruzada de 10 iteraciones.

Tabla 5.2: Comparación de los resultados de las cuatro medias de funcionamiento

a) Exactitud

Acrónimo	PCME	Esquema paralelo de SVM	Esquema paralelo de C4.5	HCGW Secuencial	AdaBoost Secuencial	Bagging Secuencial	Stacking Secuencial
BD_CaLu	**92.1 ± 0.2**	83.4 ± 0.2	83.4 ± 0.8	84.1 ± 0.3	83.7 ± 1.1	83.8 ± 1.5	84.0 ± 1.3
BD_RLC	**89.7 ± 0.8**	81.4 ± 1.2	80.7 ± 1.2	81.2 ± 2.1	80.1 ± 1.3	79.9 ± 1.8	80.3 ± 2.7
BD_KDD99	**80.6 ± 0.3**	73.1 ± 0.6	75.1 ± 0.0	77.4 ± 0.9	76.7 ± 1.6	76.4 ± 0.9	75.2 ± 0.6
BD_YT	**81.2 ± 1.5**	77.1 ± 1.8	76.3 ± 1.3	78.2 ± 1.7	76.1 ± 1.2	75.7 ± 2.1	76.1 ± 1.6
BD_PH	**73.8 ± 1.1**	63.1 ± 0.5	68.1 ± 0.8	64.3 ± 1.3	69.2 ± 0.5	68.5 ± 0.8	68.3 ± 1.7
BD_Cover	**81.1 ± 0.6**	75.1 ± 0.7	73.1 ± 1.0	78.1 ± 0.7	76.1 ± 1.1	77.5 ± 1.3	77.9 ± 1.3
BD_EMCL07	**76.8 ± 1.4**	73.1 ± 1.2	71.5 ± 2.0	76.3 ± 1.8	75.7 ± 1.7	74.8 ± 0.7	75.2 ± 0.7
BD_LDPA	**97.3 ± 2.7**	93.2 ± 1.9	86.6 ± 4.2	94.9 ± 0.4	95.1 ± 1.7	94.7 ± 1.3	95.3 ± 1.2
BD_BanMark	**94.4 ± 2.2**	88.6 ± 2.4	93.8 ± 3.7	84.6 ± 3.2	85.1 ± 3.2	87.8 ± 2.6	89.3 ± 3.7
BD_p53	**70.1 ± 1.1**	67.3 ± 1.3	63.1 ± 1.5	64.1 ± 1.8	67.2 ± 1.8	66.8 ± 3.2	65.9 ± 1.7

b) Precisión

Acrónimo	PCME	Esquema paralelo de SVM	Esquema paralelo de C4.5	HCGW Secuencial	AdaBoost Secuencial	Bagging Secuencial	Stacking Secuencial
BD_CaLu	**82 ± 1.3**	78 ± 2.6	79 ± 1.8	81 ± 2.2	80 ± 1.6	77 ± 2.4	79 ± 2.6
BD_RLC	**75 ± 1.5**	72 ± 3.3	73 ± 2.3	70 ± 1.7	69 ± 3.1	71 ± 1.6	73 ± 2.8
BD_KDD99	**75 ± 0.4**	67 ± 3.2	61 ± 2.1	72 ± 1.2	69 ± 2.4	66 ± 3.2	70 ± 1.8
BD_YT	**75 ± 1.8**	71 ± 0.8	70 ± 1.6	69 ± 3.1	71 ± 1.7	69 ± 3.3	70 ± 2.3
BD_PH	**72 ± 1.3**	57 ± 1.5	61 ± 2.1	60 ± 3.1	62 ± 2.7	63 ± 3.1	62 ± 2.7
BD_Cover	**77 ± 1.7**	70 ± 3.7	69 ± 2.1	72 ± 1.5	70 ± 1.9	70 ± 2.7	67 ± 3.6
BD_EMCL07	**72 ± 0.6**	67 ± 1.8	65 ± 2.1	68 ± 2.7	64 ± 1.8	67 ± 2.7	68 ± 3.1
BD_LDPA	**93 ± 0.7**	84 ± 1.8	83 ± 1.3	88 ± 2.6	87 ± 1.7	85 ± 1.6	84 ± 1.9
BD_BanMark	**94 ± 1.5**	82 ± 2.5	88 ± 1.9	79 ± 2.4	81 ± 1.9	83 ± 2.1	86 3.3
BD_p53	**76 ± 1.6**	71 ± 3.8	69 ± 1.8	70 ± 2.9	69 ± 2.6	69 ± 3.1	70 ± 1.9

[1]ECL-ML Machine Learning Module: `http://hpccsystems.com/ml`

c) Memoria

Acrónimo	PCME	Esquema paralelo de SVM	Esquema paralelo de C4.5	HCGW Secuencial	AdaBoost Secuencial	Bagging Secuencial	Stacking Secuencial
BD_CaLu	**85 ± 0.8**	79 ± 1.7	75 ± 1.1	83 ± 0.7	76 ± 2.1	75 ± 1.4	80 ± 0.9
BD_RLC	**80 ± 1.5**	78 ± 1.3	77 ± 1.8	76 ± 1.2	78 ± 2.1	77 ± 1.7	76 ± 1.9
BD_KDD99	**82 ± 0.6**	68 ± 1.4	63 ± 1.1	72 ± 0.8	71 ± 1.5	69 ± 2.1	71 ± 1.3
BD_YT	**73 ± 1.2**	69 ± 1.3	68 ± 0.7	76 ± 1.7	73 ± 1.6	72 ± 1.9	73 ± 2.3
BD_PH	**74 ± 2.2**	60 ± 1.3	64 ± 1.1	63 ± 1.8	62 ± 2.5	64 ± 0.9	65 ± 1.7
BD_Cover	**76 ± 0.7**	69 ± 1.7	67 ± 2.1	76 ± 0.9	73 ± 1.1	70 ± 1.5	72 ± 1.2
BD_EMCL07	**74 ± 0.7**	70 ± 0.9	64 ± 0.8	69 ± 1.7	68 ± 1.7	66 ± 1.3	67 ± 1.5
BD_LDPA	**90 ± 1.8**	82 ± 1.1	84 ± 1.3	84 ± 0.9	81 ± 0.9	83 ± 1.7	82 ± 1.3
BD_BanMark	**88 ± 0.8**	83 ± 0.9	79 ± 1.7	75 ± 1.2	76 ± 1.2	74 ± 0.9	82 ± 1.8
BD_p53	**69 ± 2.4**	64 ± 1.4	66 ± 1.9	65 ± 0.9	63 ± 2.1	62 ± 1.7	65 ± 1.3

d) Mejora

Acrónimo	PCME	Esquema paralelo de SVM	Esquema paralelo de C4.5	HCGW Secuencial	AdaBoost Secuencial	Bagging Secuencial	Stacking Secuencial
BD_CaLu	**1.84 ± 0.94**	1.54 ± 0.91	1.52 ± 1.21	1.61 ± 1.02	1.71 ± 0.94	1.58	1.63 ± 1.31
BD_RLC	**1.62 ± 0.94**	1.46 ± 0.97	1.54 ± 1.09	1.53 ± 1.15	1.50 ± 1.13	1.53	1.57 ± 1.21
BD_KDD99	**1.72 ± 0.67**	1.58 ± 0.68	1.57 ± 0.71	1.57 ± 0.73	1.51 ± 0.70	1.53 ± 0.68	1.55 ± 0.79
BD_YT	**1.54 ± 0.93**	1.37 ± 1.14	1.42 ± 1.07	1.38 ± 1.31	1.46 ± 1.06	1.34 ± 0.97	1.38 ± 1.23
BD_PH	**1.76 ± 0.5**	1.56 ± 0.76	1.69 ± 0.57	1.60 ± 1.01	1.63 ± 0.61	1.57 ± 0.59	1.60 ± 0.73
BD_Cover	**1.61 ± 1.1**	1.52 ± 1.81	1.50 ± 1.94	1.46 ± 2.11	1.54 ± 1.21	1.50 ± 1.36	1.49 ± 1.43
BD_EMCL07	**1.74 ± 0.71**	1.65 ± 0.98	1.46 ± 1.16	1.63 ± 0.83	1.68 ± 1.32	1.65 ± 0.93	1.61 ± 1.13
BD_LDPA	**1.63 ± 1.34**	1.59 ± 0.83	1.53 ± 1.02	1.50 ± 0.93	1.53 ± 0.87	1.52 ± 1.18	1.57 ± 0.93
BD_BanMark	**1.72 ± 0.82**	1.54 ± 0.94	1.58 ± 1.12	1.61 ± 1.15	1.48 ± 1.86	1.53 ± 1.21	1.59 ± 0.94
BD_p53	**1.46 ± 1.32**	1.42 ± 1.15	1.40 ± 0.94	1.39 ± 0.89	1.40 ± 1.11	1.37 ± 1.21	1.35 ± 0.97

Para el caso de la exactitud los resultados que están almacenados en la Tabla 5.2 (a), en este caso para cada una de las bases de datos que se utilizaron el PCEM obtiene los mejores resultados con respecto a los clasificadores con los que fueron comparados. Este incremento representa un aumento del 10 % en la exactitud promedio, en el cual la varianza obtenida en algunos casos es mejor que los esquemas tradicionales (bases de datos BD_RLC, BD_KDD99, BD_YT, BD_Cover, BD_BanMark y BD_p53).

Cabe destacar que en los experimentos realizados se ocuparon bases de datos con un número mayor a 2 clases. Estas bases de datos suelen ser difíciles de clasificar ya que se incrementa la complejidad de clasificar de manera correcta cada porción de los datos en el

conjunto de prueba. Esto lo podemos corroborar con los resultados de la Tabla 5.2, con las bases de datos BD_KDD99, BD_PH, BD_Cover, BD_LDPA y BD_p53. Para cada una de las bases de datos en promedio los clasificadores que seleccionamos para las comparaciones con el PCEM obtienen 71.7 %, 74.2 %, 75.7 %, 92.6 % y 65.2 respectivamente.

Podemos ver que para este tipo de bases de datos el PCEM obtiene una mejoría con respecto al promedio de los demás clasificadores que utilizamos para las comparaciones. Estas mejoras son del 6.4 % en la BD_KDD99, 7.7 % para la BD_PH, 5.4 % en la BD_Cover, 4.7 % para la BD_LPDA y finalmente 4.9 % con la BD_p53.

En el caso de los resultados de Precisión y Memoria, estas dos mediadas de funcionamiento dependen del número de clases de cada base de datos al igual que de la Mejora. Los resultados de estas dos medidas de rendimiento están representados en la Tabla 5.2 (c) y (d), en términos del promedio que se obtiene con cada una de las clases presentes de cada base de datos. En los resultados de la Tabla 5.2 (c) y (d), se puede observar que el PCEM es el que obtiene los mejores resultados con respecto a los clasificadores tradicionales y paralelos que consideramos para realizar estos experimentos.

La Mejora se utiliza comúnmente para medir el rendimiento de los modelos de respuesta en las aplicaciones de marketing. El propósito de un modelo de respuesta es identificar segmentos de la población con concentraciones potencialmente altas de personas que contesten de manera favorable (positivos) a una campaña de marketing. La Mejora revela que porcentaje de la población debe ser solicitada para obtener el mayor porcentaje de posibles participantes. Por ejemplo, si el 40 % de los clientes en una encuesta de marketing han respondido favorablemente (la clasificación positiva) a una campaña de promoción y el modelo predice con exactitud del 75 %, la mejora se obtendría dividiendo 0.75 entre 0.40 . La mejora resultante sería 1.875.

La Tabla 5.2 (d) contiene los resultados de esta medida de funcionamiento los cuales los representamos como un promedio de todas las clases presentes en cada una de las bases de datos. El PCEM en este caso obtiene una mejora del 0.146 % con respecto al mejor resultado que lo obtiene el esquema paralelo de máquinas de soporte vectorial (SVM) [GCB$^+$04].

En los resultados que mostramos en la Tabla 5.2 para la base de datos BD_p53, la cual como habíamos mencionado anteriormente tiene un gran número de atributos, se obtuvo un 70.1 % de exactitud promedio, lo que representa un 2.82 % de mejora con respecto al mejor resultado que se obtuvo con el esquema paralelo de SVM.

Estos incrementos en cada una de las medidas de rendimiento se deben a que en el PCEM se utilizaron esquemas paralelos de cinco clasificadores tradicionales, los cuales tienen mejoras individuales con respecto a los esquemas secuenciales. Para constatar este punto vamos a revisar los resultados individuales que tiene cada esquema paralelo con respecto a su esquema secuencial, así como la comparación del PCEM con el HCGW [MMMR11].

En la Tabla 5.3 se muestran los resultados de la comparación de los esquemas paralelos y tradicionales para confirmar el punto de que se obtienen mejoras de manera individual, para este caso se presentan los resultados de la exactitud que se obtuvieron con la base de datos llamada Poker Hand. Seleccionamos esta base de datos ya que la consideramos como una base de datos difícil debido a los resultados pobres que se obtienen usualmente en las medidas de rendimiento con varios clasificadores, esto es, el PCEM obtiene la mayor mejoría con respecto a los esquemas paralelos y secuenciales que utilizamos para las comparaciones.

Tabla 5.3: Comparación individual de la exactitud

Nombre del clasificador	Esquema paralelo (exactitud)	Esquema secuencial (exactitud)
Bayes Ingenuo	**59.5**	**59.5**
Tablas de Decisión	**65.78**	58.82
C4.5	**68.12**	59.15
k-NN	**67.53**	60.11
KMdias	**69.34**	47.16
Mezcla de expertos	**73.83**	61.03

En los resultados de la Tabla 5.3 para el caso de la comparación del PCEM con respecto a su esquema secuencial (HCGW), podemos observar que la mejora es de más del 12 % en promedio. En el caso de los clasificadores de base en la Tabla 5.3 cuatro de estos (K-

Medias, k-NN, Tablas de Decisión y C4.5) se obtienen mejoras con respecto a los esquemas secuenciales.

Claramente se puede observar en la Tabla 5.3 que los mayores incrementos se obtuvieron para K-Medias y Tablas de Decisión. En el caso de tablas de decisión se obtiene una mejoría debido a que el número de combinaciones permitidas (bajo grado de confiabilidad) en el esquema paralelo son mayores con respecto al esquema secuencial.

Para el caso de K-Medias permitimos un mayor número de iteraciones (cada cluster se forma por un conjunto de procesos) para poder alcanzar la exactitud que se obtiene en el esquema paralelo. En el caso del esquema secuencial la ejecución del clasificador se paraba antes de que se pudiera alcanzar la exactitud del esquema paralelo, ya que se tendría un tiempo de ejecución muy alto, en particular esto se alcanzaría en trece veces el tiempo que necesitamos en el esquema paralelo.

Finalmente, para el caso del esquema paralelo de C4.5 y k-NN los aumentos en los índices de exactitud nos son comparables con como los que se obtienen en K-Medias y Tablas de Decisión, sin embargo también representa un factor en el aumento de los índices de exactitud del PCEM. Para el caso de Bayes Ingenuo la exactitud de ambos métodos es similar que el esquema secuencial ya que para este esquema paralelo nos enfocamos mas en cuestiones de tiempos de ejecución. Podemos constatar que en efecto debido a los aumentos individuales de cada clasificador de base, el PCEM obtiene mejores resultados con respecto a los esquemas secuenciales y paralelos con los que se comparó.

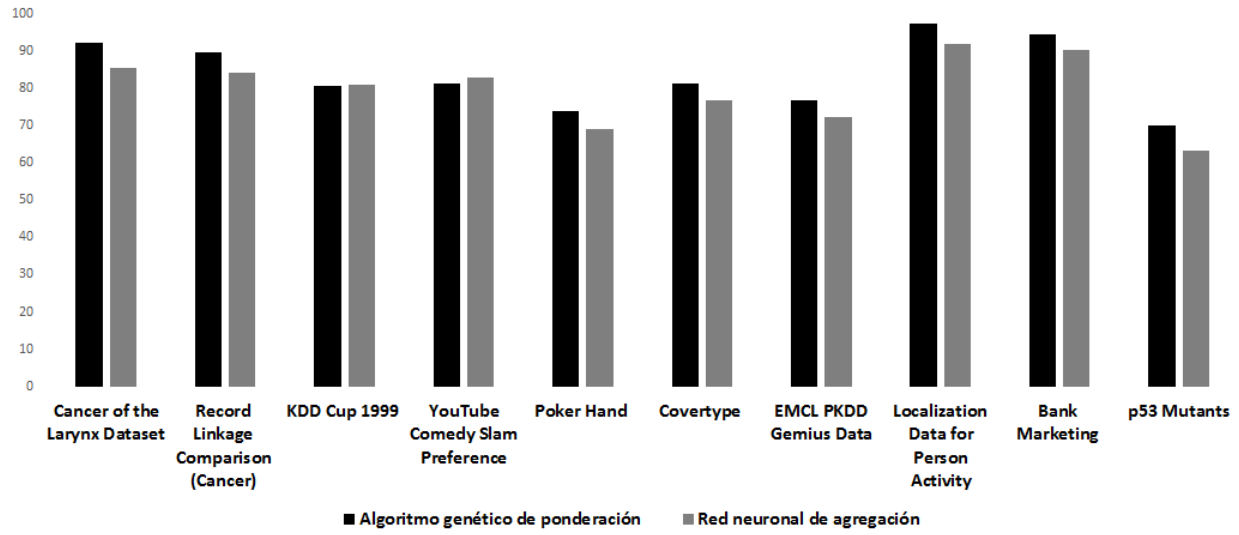
A pesar de que los resultados de la Tabla 5.2 muestran mejoras con respecto a cada una de las medidas de rendimiento estándar, decidimos llevar a cabo una prueba estadística para definir si los resultados del PCEM son significativamente mejores con respecto a los esquemas paralelos y tradicionales de los clasificadores del aprendizaje maquinal.

Para este caso utilizamos la prueba t-test, este método estadístico se utiliza para saber cuál es el nivel de significación entre dos conjuntos de muestras [Dem06]. En este caso, las muestras serán los valores de alguna de las medidas de rendimiento que ocupemos, en este caso la exactitud que obtuvimos con el PCEM y el HCGW.

Una vez que realizamos dicha prueba obtuvimos un nivel de significación alto con un valor del 99.95 %, por lo cual la mejoría observada en los resultados del PCEM es estadísticamente significativo con respecto a los esquemas tradicionales y paralelos.

En un trabajo de investigación previo [MMMR11] pudimos constatar sus beneficios con respecto al esquema tradicional de los ensambles de tipo mezcla de expertos que es utilizar redes neuronales. Para esta investigación doctoral realizamos una serie de experimentos con el algoritmo genético de ponderación y una red neuronal de agregación tal y como la que se mostró en la sección 3.10, utilizando todas las bases de la Tabla 5.1. Estos experimentos se realizando mediante utilizando validación cruzada de los cuales los resultados los podemos ver en la gráfica de la Figura 5.1.

Figura 5.1: Comparación de la Exactitud entre un Algoritmo genético de ponderación y un red neuronal de agregación



Podemos ver en la gráfica de la Figura 5.1, que en el algoritmo genético de ponderación obtiene en promedio mejores índices de medidas de rendimiento con respecto al esquema tradicional de las redes neuronales de agregación para las bases con las cuales probamos, sin embargo, en algunos casos las redes neuronales sueles obtener mejores o iguales resultados que el PCEM.

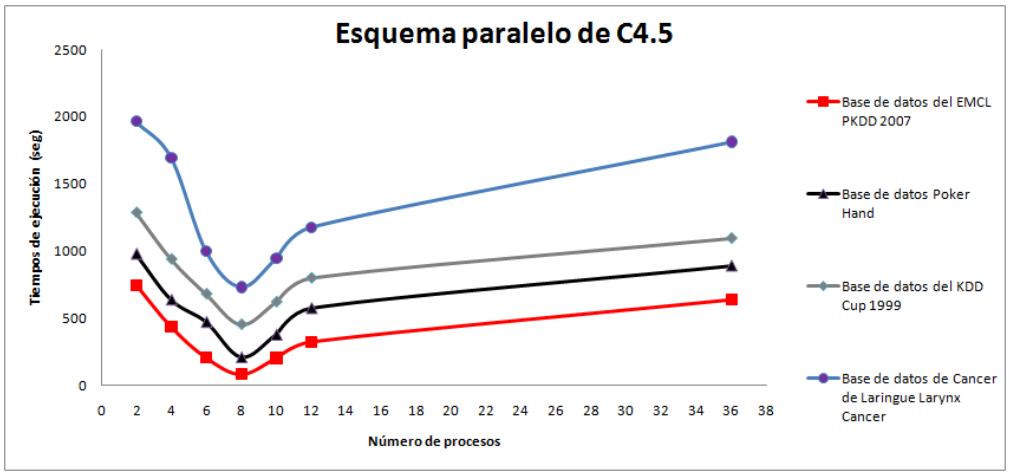
De estas pruebas decidimos utilizar este método para encontrar los pesos, pero estamos conscientes que pueden existir métodos diferentes de cómo hacerlos los cuales discutiremos en el capítulo de Conclusiones y Trabajo a Futuro. Con esta comparación finalizamos los resultados obtenidos por el PCEM para las mediadas de funcionamiento, por lo que en la siguiente sección presentaremos los resultados de los tiempos de ejecución.

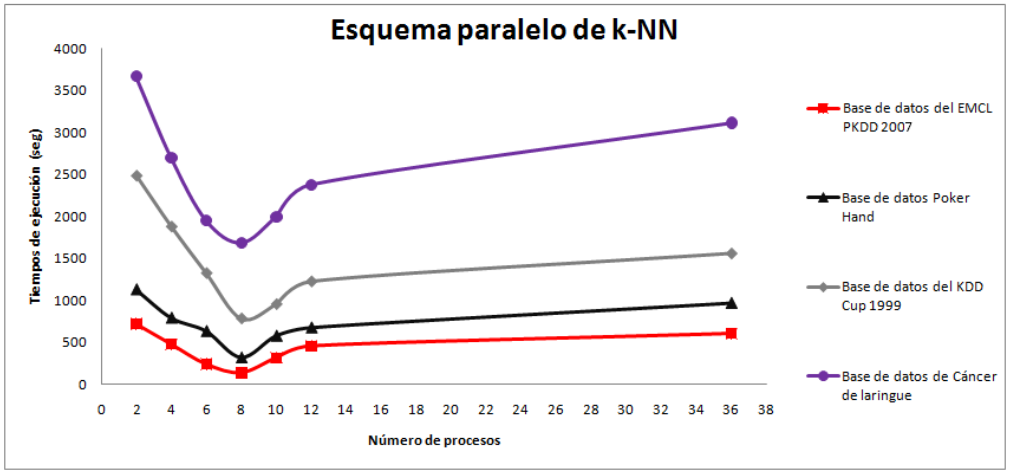
5.5. Resultados del análisis de los procesos para ejecución del PCEM

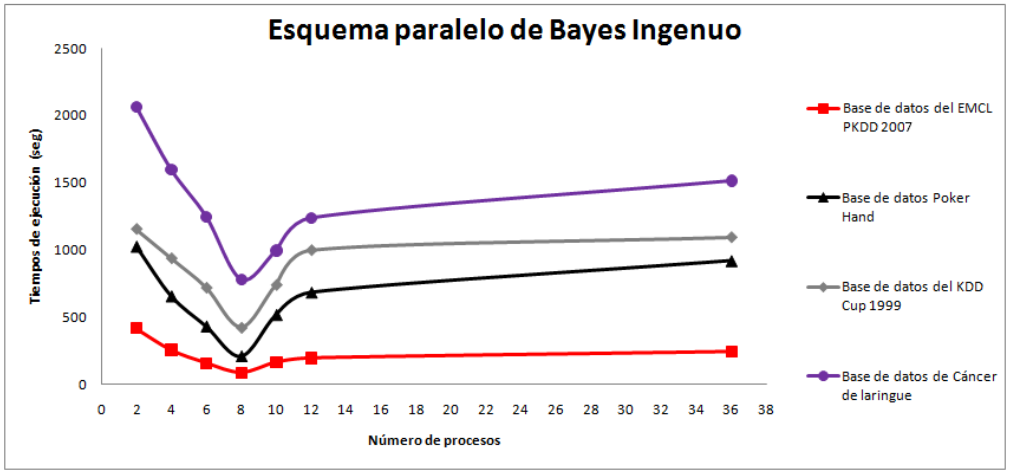
Para introducir los resultados de los tiempos de ejecución obtenidos con el PCEM, primero presentaremos la prueba que llevamos a cabo para determinar el número de procesos por cada uno de los esquemas paralelos de los clasificadores de base. Para esta prueba ejecutamos cada una de los esquemas paralelos en un nodo del cluster Pacífico, con 8 procesadores y 8 GB en memoria RAM. Para este caso utilizamos las bases de datos BD_CaLu, BD_KDD99, BD_PH y BD_EMCL07 de la Tabla 5.1, en donde para cada base de datos probamos con 2, 4, 6, 8, 10, 12 y 36 procesos obteniendo los resultados que mostramos en la Figure 5.2.

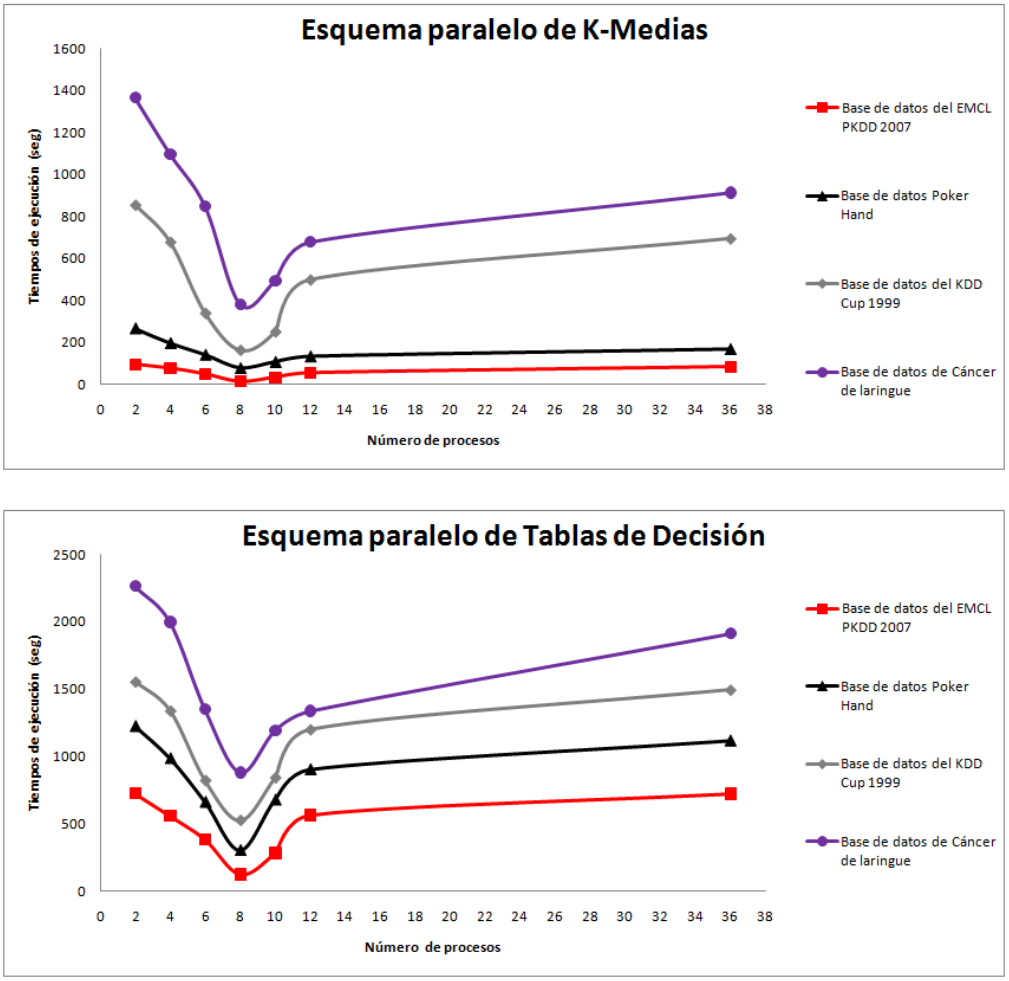
Se decidieron utilizar estas cuatro bases de datos para cubrir las dos categorías que establecimos para decir que tenemos grandes cantidades de datos. Por un lado, se consideraron las dos bases con mayor número de registros las cuales son BD_CaLu y BD_KDD99 omitiendo a BD_RLC al ser muy similar con respecto a BD_CaLu. Por el otro lado, consideramos que tenemos grandes cantidades de datos cuando se tiene un gran dimensionalidad en los datos, esto precisamente se presenta en BD_PH ya que es la de mayor peso entre número de registros y atributos. Finalmente, decidimos escoger la BD_EMCL07 ya que en un trabajo previo[MMMR11], se pudo constatar que es una base de datos difícil debido a la distribución de los datos de cada clase presente.

Figura 5.2: Resultados de las pruebas individuales para determinar el número adecuado de procesos.







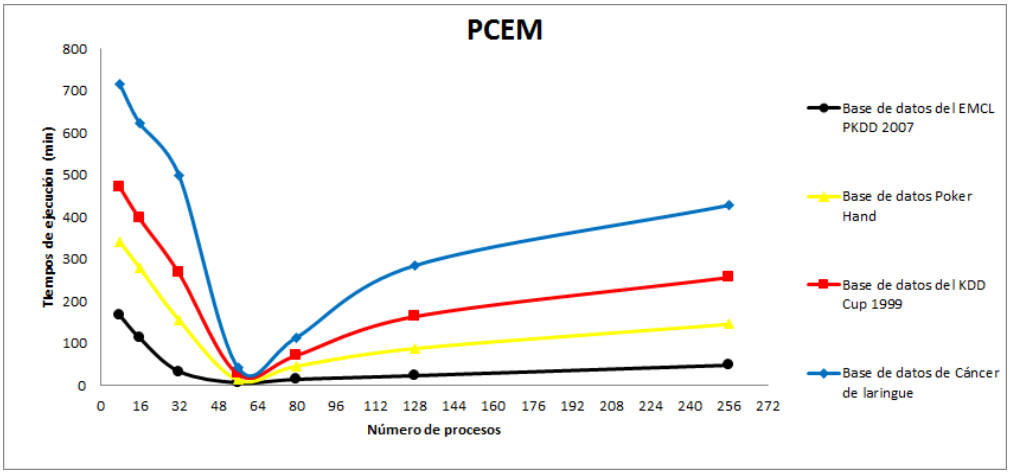


En la Figura 5.2 se presenta los resultados de cada esquema paralelo para cada clasificador de base. Cuando utilizamos 2, 4 y 6 procesos, el paralelismo no es explotado de manera adecuada. En el caso en que se ocuparan 8 procesos, los recursos de cada proceso fueron explotados de manera equitativa, obteniendo los mejores resultados.

Cuando se utilizó un número mayor a 8 procesos, la carga de trabajo de cada procesador se incrementó y lo que obtuvimos fue un incremento en los tiempos de ejecución. La sobrecargan en el PCEM se presenta ya que no hay un control de balance de carga para evitar que cualquier procesador de alguno de los nodos del cluster atienda a todos los procesos que son asignados por el CPU, llevando a un estado de *overloading* o sobrecarga el cual afecta el tiempo de ejecución de la aplicación. Dado estos resultados, concluimos que el más adecuado número de procesos ejecutados en un nodo de 8 procesadores es igual a ocho.

Ahora vamos a revisar los resultados que se obtuvieron en la prueba que realizamos para determinar el número de procesos para llevar a cabo el funcionamiento del PCEM. Para determinar la configuración de procesos que nos ofreciera los mejores resultados, realizamos diferentes experimentos con las cuatro bases de datos anteriormente mencionadas en las pruebas individuales de los tiempos de ejecución. El número de procesos que utilizamos fueron 8, 16, 32, 56, 80, 128 y 256, los cuales fueron ejecutados en el cluster Pacífico. En la Figura 5.3 se pueden observar los resultados de estas pruebas.

Figura 5.3: Tiempos de ejecución del PCEM en un cluster de 7 nodos (8 procesadores por nodo).



En la Figura 5.3 se puede observar que los mejores tiempos de ejecución son obtenidos con $7 \times 8 = 56$ procesos. Si recordamos la arquitectura que presentábamos en la sección 4.2, los componentes del PCEM son: cinco clasificadores de base, el algoritmo genético de ponderación y el coordinador general.

En la arquitectura proponíamos que cada uno de estos componentes fuera ejecutado en un nodo del cluster pacifico, pues mediante esta serie de experimentos que realizamos obtuvimos el mejor tiempo de ejecución cuando utilizamos 56 procesos, teniendo de esta manera de manera equitativa cada uno de los procesadores del cluster.

Podemos ver en la Figura 5.3 que si utilizamos un número menor de 56 procesos el tiempo de ejecución se incrementa, debido a que en este caso algunos procesadores entraban en un estado de inactividad sin explotar el paralelismo. En el caso en el que se utilizó un número mayor a 56, se presentó una sobrecarga de trabajo en alguno de los procesadores presentes en el cluster. En este caso la carga se presenta debido al alto número de comunicaciones que existe entre los procesadores que están en los diferentes nodos del cluster, los cuales se encuentran procesando instrucciones en todo momento para resolver una tarea de clasificación de datos.

5.6. Resultados de tiempos de ejecución individuales del *PCEM*

Con esta configuración de procesos se realizaron las pruebas de medidas de rendimiento que presentamos en la sección 5.4, por lo que esta sección se revisaran los resultados de las pruebas individuales con la base de datos BD_PH, de la Tabla 5.1. Seleccionamos esta base de datos ya que tiene un balance en cuestión de dimensionalidad y de número de registros (manejo de grandes cantidades de datos). Al igual que en los resultados que revisamos de las medidas de rendimiento, vamos a presentar los resultados individuales de cada uno de los esquemas paralelos de los clasificadores de base, así como los del PCEM que serán comparados con el *HCGW*, los resultados de estas pruebas los podemos observar en la Tabla 5.4.

En la Tabla 5.4 se puede observar que con cada uno de los clasificadores obtienen mejores resultados comparados con la versión secuencial de cada clasificador de base. En la comparación del ensamble de tipo mezcla de expertos, tenemos por una parte el PCEM que representa el esquema paralelo y por otra parte el HCGW que representa el esquema secuencial. El tiempo de ejecución obtenido por el PCEM representa el 6 % del tiempo de ejecución que obtiene el HCGW, lo que representa una gran reducción en cuestión de tiempos de ejecución.

Con estos resultados nos podemos dar una idea de cuál es el aporte que represento el PCEM con respecto a los clasificadores tradicionales y paralelos utilizados en esta compara-

Tabla 5.4: Comparación de los tiempos de ejecución individuales

Nombre de los clasificadores	Esquema paralelo (tiempo/min)	Esquema secuencial (tiempo/min)
Bayes Ingenuo	**7**	22
Tablas de Decisión	**11**	38
C4.5	**13**	41
k-NN	**7**	32
KMedias	**3**	15
Mezcla de expertos	**20**	445

ción. No obstante, en el último resultado que mostraremos en este capítulo, ocuparemos las bases de datos de mayor tamaño de la Tabla 5.2 para constatar el aporte final del PCEM en cuestión de los tiempos de ejecución (TimEx). Estos resultados se encuentran en la Tabla 5.5 en los cuales solo presentamos la comparación con el mejor esquema paralelo que se utilizó en este experimento, el cual es el esquema paralelo de máquinas de soporte vectorial [GCB+04] (PSVM).

En la Tabla 5.5 se encuentran dos columnas que nos interesan para esta comparación que es el tiempo de ejecución extra y mejora de exactitud. Esta proporción se calcula por un lado mediante la resta de los tiempos de ejecución del PCEM menos los tiempos de ejecución del PSVM y por el otro lado de la resta de la exactitud del PCEM menos la exactitud del PSVM, de los cuales vamos a analizar cada uno de los resultados obtenidos.

Para el caso de la base de datos de Cáncer de Laringe se tiene que esperar 7.2 minutos para obtener un incremento del 8.7 % de exactitud promedio. En la base de datos Record Linkage Comparison Patterns (Cáncer), tenemos una relación de 8.2 minutos de tiempo de ejecución extra vs. un incremento del 8.3 % de exactitud promedio.

Tabla 5.5: Comparación tiempos de ejecución vs exactitud

Acrónimo	TimEx del PCME (min)	TimEx del PSVM (min)	Exactitud del PCME	Exactitud del PSVM
BD_CaLu	54.4	**47.2**	**92.1**	83.4
BD_RLC	46.4	**38.2**	**89.7**	81.4
BD_KDD99	35.1	**29.3**	**88.6**	73.1
BD_YT	23.4	**17.7**	**81.2**	77.1
BD_PH	14.4	**9.2**	**73.8**	63.1
BD_Cover	13.4	**8.4**	**81.1**	75.1
BD_EMCL07	5.8	**3.1**	**76.8**	71.5

Acrónimo	TimEx extra (min)	Porcentaje de Mejora de exactitud
BD_CaLu	7.2	8.7 %
BD_RLC	8.2	8.3 %
BD_KDD99	5.8	15.5 %
BD_YT	5.7	4.1 %
BD_PH	5.2	10.7 %
BD_Cover	5	6 %
BD_EMCL07	2.7	5.3 %

En el caso de la base de datos KDD Cup 1999, tenemos que esperar 5.8 minutos de tiempo de ejecución extra para obtener un aumento del 15.5 % la exactitud promedio. Para la base de datos YouTube Comedy Slam Preference, se obtiene un incremento de la exactitud promedio del 4.1 % al esperar 5.7 minutos de tiempo de ejecución extra.

Para las bases de datos Poker Hand y Covertype, se tiene que esperar 5.2 y 5 minutos de tiempos de ejecución extra vs. una mejora de 10.7 % y 6 % de exactitud promedio respectivamente. Finalmente, para la base de datos PKDD 2009 Gemius dataset, el incremento de

la exactitud promedio fue del 5.3 %, con una espera de 2.7 minutos de tiempo de ejecución extra.

Podemos ver que el tiempo extra con respecto al PCEM y el PSVM es bajo para cada una de las bases de datos, en promedio se debe esperar 5.6 minutos, para obtener un incremento del 8.37 % de exactitud promedio. Con esta prueba finalizamos este capítulo por lo que solo resta presentar las conclusiones y trabajo a futuro de este trabajo.

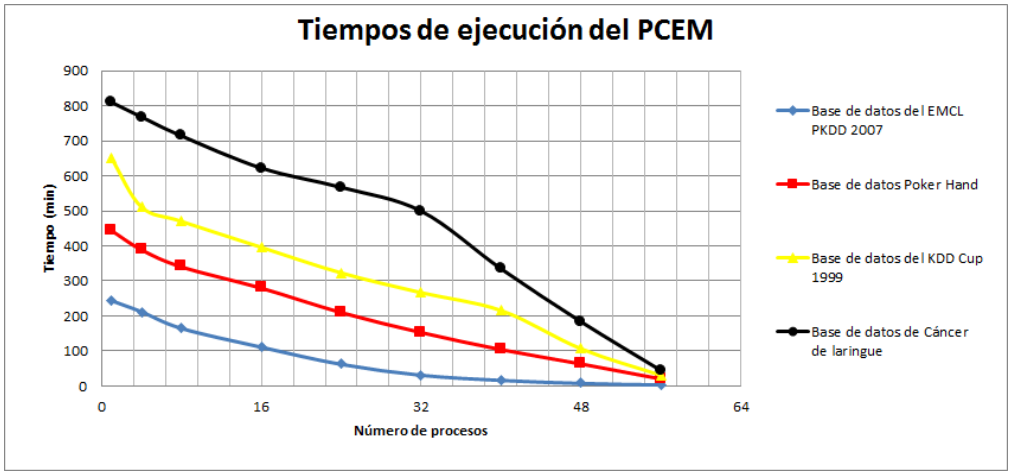
5.7. Rendimiento paralelo del *PCEM*

En esta sección presentamos el desempeño del *PCEM*, tomando algunas de las medidas de rendimiento que se presentaron en la sección 3.2. Para este caso mostraremos los resultados de tiempos de ejecución, el factor de mejora del rendimiento o *speedup* y la eficiencia que obtuvo el *PCEM* en una serie de pruebas tomando las bases de datos BD_CaLu, BD_KDD99, BD_PH y la BD_EMCL07, por las mismas razones que señalamos en la sección 5.4.

Vamos a comenzar con el tiempo de ejecución que obtiene el *PCEM* con las cuatro bases de datos mencionadas anteriormente, así como el uso de diferente numero de procesos que van desde 1, el cual representa el esquema secuencial, hasta 56 que fue el número maximo que encontramos en las pruebas que presentamos en la sección 5.5, los resultados de esta prueba se pueden observar en la Figura 5.4.

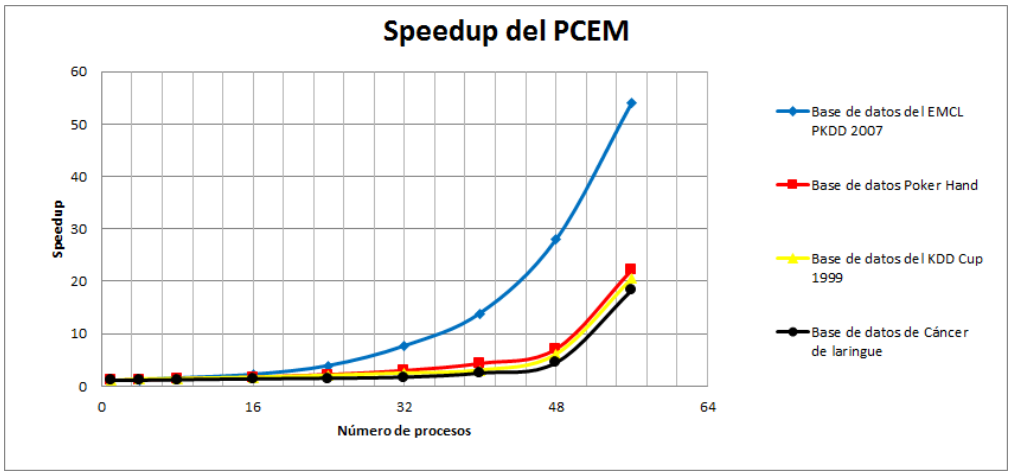
En la Figura 5.4 podemos observar que a medida que aumentamos el número de procesos el *PCEM* obtuvo una reducción considerable con respecto a la versión secuencial (*HCGW*), tal y como lo mencionamos en la seccion 5.5, en algunos casos el *PCEM* solo representa el 6 % del tiempo total de la versión secuencial.

Figura 5.4: Tiempos de ejecución del *PCEM* vs *HCGW*.



Una vez que revisamos los tiempos de ejecución del *PCEM* vamos a presentar los resultados del factor de mejora de rendimiento o también llamado factor de aceleración o simplemente por su nombre en ingles *speedup*[HJ11], lo resultados se muestran en la Figura 5.5.

Figura 5.5: Speedup del *PCEM* utilizando las bases de datos BD_CaLu, BD_KDD99, BD_PH y la BD_EMCL07.

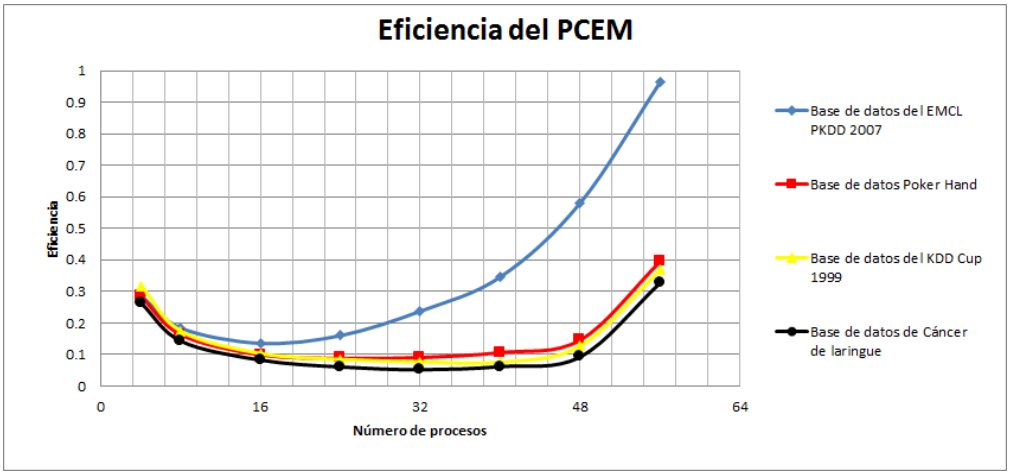


Podemos ver en la Figura 5.5, que el *speedup* para las cuatro bases de datos se mantiene en un crecimiento lineal hasta un número de procesos igual a 24. Después de este número de

procesos el crecimiento se convierte en un crecimiento super lineal para todas las bases de datos lo cual nos proporciona un resultado alentador del comportamiento del *PCEM* y sobre todo cabe resaltar el buen desempeño que se tiene para la BD_EMCL07 de aproximadamente 55 %.

Finalmente, mostraremos la eficiencia que obtuvo el *PCEM* para cada una de las bases de datos que se seleccionaron para esta prueba, estos resultados se presentan en la Figura 5.6.

Figura 5.6: Eficiencia del *PCEM* utilizando las bases de datos BD_CaLu, BD_KDD99, BD_PH y la BD_EMCL07.



En base a la Figura 5.6, podemos observar que para las bases de dato BD_CaLu, BD_KDD99 y BD_PH se tiene un decrecimiento de la eficiencia del número de procesos iniciales, que en este caso es 4, hasta un número de procesos igual a 24. Después de este número de procesos la eficiencia se mantiene constante hasta que se incrementa después de un número de procesos igual a 48.

Para la base de datos BD_EMCL07, el decrecimiento es más corto ya que solo se encuentra en el intervalo de 4 a 16 procesos. Después de 16 procesos se puede observar que se tiene un incremento considerable en la eficiencia hasta llegar a un 96 %, lo cual en base a la teoría de Hwang et al.[HJ11], esta eficiencia se obtiene cuando todos los procesadores están siendo completamente utilizados durante todo el periodo de ejecución. Con estas pruebas

concluimos este capítulo, por lo que ahora presentaremos las conclusiones y trabajo a futuro que generamos de esta investigación.

Capítulo 6

Conclusiones y Trabajo a Futuro

6.1. Conclusiones

En este trabajo se desarrolló un Sistema de Clasificación Paralelo basado en un Ensamble de tipo Mezcla de Expertos (Parallel Classification System based on an Ensemble of Mixture of Experts - PCEM), con el cual se obtuvieron altos índices en las medidas de rendimiento y bajos tiempos de ejecución al aplicar la tarea de clasificación sobre grandes cantidades de datos.

Para el PCEM propusimos desarrollar esquemas paralelos de los componentes del ensamble, por lo que ahora la búsqueda se enfocaría a encontrar esquemas paralelos de clasificadores de base y métodos de ponderación para cada uno de estos. Para el caso del método de asignación de los pesos a cada clasificador de base en el esquema tradicional de los ensambles de tipo mezcla de expertos, se realiza mediante el uso de una red neuronal, tal y cómo se mostró en la sección 3.9, sin embargo, en este trabajo desarrollamos un esquema paralelo de un algoritmo genético de ponderación, el cual representa una forma novedosa de cómo llevar a cabo este componente en un ensamble de tipo mezcla de expertos.

En base a una serie de experimentos pudimos constatar que el algoritmo genético de ponderaciones es una mejor opción para este tipo de ensambles con respecto al método tradicional de utilizar una red neuronal de asignación. En base a las pruebas experimentales que presentamos en la Figura 5.1, el algoritmo genético de ponderaciones obtiene en promedio 5 % más de exactitud con respecto a la red de asignación tradicional. Este resultado hace

que la hipótesis planteada al inicio de este documento referente a que si se exploraba otro método de búsqueda no informada o informada nos proporcionara mayores beneficios en las medidas de rendimiento con respecto al método tradicional, se cumple en base a las pruebas experimentales de la Figura 5.1.

Una de las ventajas que represento utilizar el algoritmo genético de ponderaciones con respecto a los métodos tradicionales, fue la exploración de un conjunto mayor de posibles soluciones al problema de asignar los pesos a cada clasificador de base, debido a que se trata de un esquema paralelo. Otra de las ventajas que representó este esquema paralelo con respecto a trabajos previos, cómo el HCGW [MMMR11], fue el criterio de paro basado en la diferencia de las soluciones [BMP12], (como se vió en la sección 4.4.3), con el cual pudimos obtener mejores soluciones con respecto a criterios de paro tradicionales (número de iteraciones).

Una vez que tuvimos cubiertos cada uno de los componentes del PCEM, nos dimos a la tarea de seleccionar un conjunto de bases de datos para poner a prueba el PCEM. Dentro de este conjunto, la mayoría de las bases de datos se seleccionaron del repositorio de la Universidad de California en Irvine [BM98] (UCI), de los cuales en algunos casos se tratan de bases de datos del mundo real. Una de las bases de datos que seleccionamos, debido a su complejidad, fue la base de datos generada en un estudio de todos los pacientes con Cáncer de Laringe [PBV$^+$10] a lo largo de 10 años, por lo que representa uno de los repositorios más grandes a nivel República Mexicana.

En este trabajo planteamos la hipótesis inicial que si combinamos un conjunto de clasificadores paralelos mediante un ensamble de tipo mezcla de expertos podríamos obtener tiempos de ejecución bajos y altos índices de medidas de rendimiento mientras se clasifican grandes cantidades de datos. Esta hipótesis inicial fue cumplida ya que en cada uno de las pruebas con el PCEM obtuvimos los mejores resultados con respecto a un conjunto de clasificadores tradicionales y paralelos.

Esta hipótesis inicial fue cumplida ya que en cada uno de las pruebas con el *PCEM* obtuvimos los mejores resultados con respecto a un conjunto de clasificadores tradicionales y paralelos. En base a las pruebas experimentales que se realizaron con el *PCEM*, se pudieron

manejar grandes cantidades de datos (pudimos manejar bases de datos con tamaños de 7,652 millones de registros) obteniendo en cada caso altos índices en las medidas de rendimiento. Los resultados de estas pruebas los presentamos en la Tabla 5.2, la Tabla 5.3 y la Tabla 5.5 de las secciones 5.4 y 5.6.

Para los resultados de la Tabla 5.2 pudimos constatar que para cada una de las bases de datos que se utilizó el *PCEM* obtuvo los mejores resultados en cada una de las medidas de rendimiento. En promedio el *PCEM* obtiene 83.71 % de exactitud tomando todos los resultados de la Tabla 5.2, el promedio de la exactitud para los clasificadores tradicionales fue de 77.35 % y para los esquemas secuenciales de los clasificadores basados en ensambles fue de 78.54 %. Dado los promedios anteriores el *PCEM* obtiene una mejora del 6.35 % con respecto a los clasificadores tradicionales y un 5.17 % con respecto a los clasificadores basados en ensambles secuenciales.

En la Tabla 5.3 mostramos una comparación individual que se realizó con cada uno de los esquemas paralelos de cada clasificador de base que se consideró en el *PCEM*, en cuyos resultados pudimos constatar que para cuatro de los cinco clasificadores de base se obtuvieron incrementos en las medidas de rendimiento con respecto al esquema tradicional. De los resultados de la Tabla 5.3 pudimos concluir que se obtiene un aumento del 10 % para todas las bases de datos que se consideraron, comprobando que la hipótesis de que es posible obtener mejoras en los índices de las medidas de rendimiento con algunos esquemas paralelos de los clasificadores de base, se cumple.

En cuestión de los tiempos de ejecución que obtuvimos con el *PCEM*, realizamos una serie de pruebas cuyos resultados los presentamos en la Figura 5.2, en la Figura 5.3, en la Tabla 5.4 y en la Tabla 5.5. Los resultados que presentamos en la Figura 5.2 nos permitieron encontrar la mejor combinación de procesos para la ejecución de cada uno de los clasificadores de base del *PCEM* de forma individual, en cuyo caso el mejor número es igual a 8 procesos por cada componente, optando por utilizar un esquema equitativo con respecto a la carga de trabajo de las unidades de procesamiento presentes en cada nodo del cluster.

Una vez que presentamos los resultados de las pruebas para determinar el mejor núme-

ro de procesos para cada clasificador, procedimos a realizar una prueba experimental para determinar el correcto funcionamiento del *PCEM* para cada uno de sus componentes cuyos resultados los presentamos en la Figura 5.3. En estos resultados determinamos que el mejor número de procesos es igual a 56, esto se debió a que cómo tenemos siete componentes, cotando los cinco clasificadores de base, el algoritmo genético de ponderación y un coordinador general, por cada uno de ellos la mejor configuración de manera individual fue utilizar 8 procesos en base a la prueba de la Figura 5.2.

En los resultados de la Tabla 5.4, pudimos constatar que de manera individual con cada uno de los clasificadores de base se obtiene una reducción en los tiempos de ejecución con respecto a los esquemas secuenciales. Mediante estos resultados en cuestión de los tiempos de ejecución, concluimos que el tiempo de ejecución obtenido por el *PCEM* representa el 6 % del tiempo de ejecución que obtiene el HCGW (esquema secuencial de mezcla de expertos), lo cual representa una gran reducción en cuestión de tiempos de ejecución, cómo era de esperarse ya que en este caso se construyó un sistema de clasificación con un enfoque de un paralelismo global para obtener una reducción de los tiempos de ejecución en cada uno de sus componentes. Con esto se comprueba la hipótesis de que es posible obtener una reducción de menos de la mitad de los tiempos de ejecución del esquema secuencial.

Para reforzar los resultados que obtuvimos en cuestión de los índices de las medidas de rendimiento, se presentó un análisis con las bases de datos de mayor tamaño, los resultados los presentamos en la Tabla 5.5. Esta prueba consistió en comparar los resultados de tiempos de ejecución vs. índices en las medidas de rendimiento, entre el mejor esquema paralelo de los clasificadores de base y el PCME. Los resultados que obtuvimos para esta prueba nos permitieron tener una conclusión con respecto a que tan conveniente es usar el *PCEM* con respecto a otros sistemas de clasificación con los cuales lo comparamos. Podemos concluir que el *PCEM* nos da una ventaja en cuestión de medidas de rendimiento, ya que debemos esperar 5.6 minutos para obtener un incremento del 8.37 % de exactitud promedio con respecto a los demás esquemas tradicionales y paralelos de clasificación.

Finalmente, mostramos la evaluación del rendimiento paralelo del *PCEM* utilizando las

bases de datos BD_CaLu, BD_KDD99, BD_PH y la BD_EMCL07, que como explicamos en la sección 5.5 cubren un problema de clasificación sobre grandes cantidades de datos. En la Figura 5.4 se mostró que el *PCEM* obtiene una reducción considerable para las cuatro bases de datos hasta llegar a la mejor configuración del maximo número de procesos (56 procesos).

De la Figura 5.5 se mostró que el factor de aceleración o *speedup* para las cuatro bases de datos alcanza un crecimiento super lineal, lo cual es lo deseable para cualquier aplicación paralela que se desarrolló, inclusive observamos que para la base de datos BD_EMCL07 el *speedup* alcanzado fue de más del 55 %. La base de datos BD_EMCL07, como lo mostramos en la Tabla 5.1, consta de 379,485 registros, 2 clases, y 8 atributos, con un distribución de las clases es de [73 %, 27 %].

En la Figura 5.6 se mostró la eficiencia del *PCEM*, la cual para las bases de datos BD_CaLu, BD_KDD99, BD_PH, en promedio se obtiene un 40 % máxima para la mejor combinación de procesos, sin embargo, para la base de datos BD_EMCL07 el porcentaje de eficiencia máxima fue del 96 %, lo cual indica que se alcanza una utilización máxima de todos los procesadores involucrados en el cluster Pacífico.

Mediante la implementación del *PCEM* obtuvimos una reducción en la probabilidad de caer en le problema de sobreajuste ya que en la mayoría de los resultados de las medidas de rendimiento no se presentó un modelo que fuera mejor que el nuestro. Sin embargo, esto no quiere decir que no haya ningún método de clasificación que sea mejor que nuestro modelo que presentamos, ya que siempre van existir mejoras en alguno de los trabajos que existen en la literatura, tanto en términos de medidas de rendimiento cómo en tiempos de ejecución. Es por ello que este proyecto de investigación doctoral no concluye con estas pruebas y resultados que obtuvimos, se tienen una serie de trabajos a futuro para incrementar la utilidad, rendimiento y escalabilidad del *PCEM*, los cuales serán presentados en la siguiente sección.

6.2. Trabajo Futuro

Si bien nuestros resultados nos dan un buen panorama del funcionamiento del *PCEM*, siempre existen comparaciones que no se pueden realizar por los tiempos que se plantearon al inicio de este proyecto de investigación doctoral. Particularmente nos referimos a pruebas con nuevas aplicaciones que nos permiten utilizar el cómputo paralelo sobre clasificadores del aprendizaje maquinal, cómo por ejemplo Hadoop/MapReduce, Spark, Giraph, GraphLab [GGP+17].

Si bien es cierto que nos faltó realizar pruebas con estas nuevas herramientas de programación, para el caso de los esquemas paralelos del clasificador C4.5 y SVM, tomamos estos clasificadores de la herramienta llamada *ECL-ML Machine Learning Module*[1] la cual representa un trabajo reciente (2013) en la que se propone un conjunto de esquemas paralelos de clasificadores. tradicionales. Sin embargo consideramos que un trabajo a futuro es la comparación de nuestros resultados con las herramientas de programación paralela que presentamos anteriormente.

Otro de los trabajos a futuro que tenemos con el *PCEM* es lograr una escalabilidad de su funcionamiento sobre un cluster de mayor tamaño. En los experimentos que mostramos en el capítulo 5, mencionábamos que utilizamos parte del cluster Pacifico con 7 nodos, con 8 procesadores y 8 GB en memoria RAM cómo parte de las características del cluster. Podemos ver que se trata de un cluster pequeño ya que en la actualidad se manejan cluster más de 100 nodos, un ejemplo de esto es el cluster Aitzaloa con 2160 procesadores distribuidos en 540 Intel Xeon quad-core's. El objetivo de este trabajo futuro es migrar nuestro sistema de clasificación paralelo a un cluster con estas características para poder obtener mejoras en cuestiones de tiempos de ejecución, manejo de grandes cantidades de datos o índices de medidas de rendimiento.

Dado el punto anterior, suponga que se realizara una prueba exhaustiva sobre el *PCEM* en un cluster cómo el cluster Aitzaloa, y suponga que por alguna razón el algoritmo genético de

[1]ECL-ML Machine Learning Module: `http://hpccsystems.com/ml`

ponderación dejara de funcionar, la finalización del *PCEM* simplemente no se llevaría a cabo porque el coordinador general quedaría esperando infinitamente el mensaje del coordinador local del algoritmo genético de ponderación sin que este nunca le respondiera. Este tipo de escenarios se plantean cuando una aplicación paralela o distribuida necesita un mecanismo de tolerancia a fallas, es por ello que el siguiente trabajo a futuro sería el de implementar dicho mecanismo sobre el funcionamiento del *PCEM* para obtener una aplicación mucho más robusta.

Como vimos en el capítulo 5, proporcionamos una configuración del número de procesos más adecuado de utilizar por cada nodo del cluster, lo cual representa una forma equitativa de distribuir el trabajo en el cluster y así evitar la sobrecarga de alguno de los procesadores. Estas asignaciones nos ofrecieron buenos resultados en cuestión de tiempos de ejecución con respecto a la versión secuencial, sin embargo, no es la forma más óptima de hacerlo. En este caso dentro del trabajo a futuro se plantea incorporar una estrategia de balance de carga para que se utilice un mayor número de procesos en cada nodo del cluster y de esta manera explotar aún más las virtudes del cómputo paralelo.

Finalmente, cómo parte del trabajo futuro que tenemos con el *PCEM*, es mejorar a los índices de las medidas de rendimiento, en particular del método de asignación de pesos. Si bien es cierto que los métodos de asignación de los pesos tradicionales (redes neuronales, métodos de investigación de operación y algoritmos evolutivos) ofrecen una buena alternativa para incrementar las medidas de rendimiento, su funcionamiento se centra en ocupar solamente una de ellas que es la exactitud.

Sin embargo, las otras tres medidas de rendimiento son importantes en un cierto grado para diferentes áreas de investigación, por ello sería una buena opción maximizar a la par cada una de estas medidas de rendimiento, esto podría ser resuelto mediante la implementación de un método de Optimización Multiobjetivo [CLV06].

El objetivo principal de cualquier técnica de optimización es encontrar el óptimo (o los óptimos) globales de cualquier problema. En el mundo real la mayoría de los problemas que se quieren optimizar son multiobjetivo. Esto es, suelen tener dos o más funciones objetivo

que deben satisfacerse simultáneamente y que posiblemente están en conflicto entre sí.

La optimización con objetivos múltiples (llamada también optimización vectorial y con criterios múltiples) puede definirse cómo[CLV06]:

"El problema de encontrar un vector de variables de decisión que satisfaga las restricciones y optimice una función vectorial cuyos elementos representen las funciones objetivo. Estas funciones forman una descripción matemática de los criterios de desempeño que usualmente están en conflicto entre sí. Por lo tanto, el término *optimizar* significa encontrar una solución tal que proporcione valores para todos los objetivos que resulten aceptables para el diseñador".

Esta es la última parte de los trabajos a futuro que tenemos planeados para el *PCEM*, consiste en implementar un método de optimización multiobjetivo en el cual las funciones objetivo a maximizar serían cada una de las medidas de rendimiento del aprendizaje maquinal, teniendo de esta manera un balance entre cada una de las medidas de rendimiento.

Referencias

[20007] ECML/PKDD 2007. European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases. `http://www.ecmlpkdd2007.net`, 2007.

[ABH07] G. Aparício, I. Blanquer, and V. Hernández. *A Parallel Implementation of the K Nearest Neighbours Classifier in Three Levels: Threads, MPI Processes and the Grid*, pages 225–235. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[AK99] S.and Eui-Hong H. Anurag and Vipin K. Parallel formulations of decision-tree classification algorithms. *Data Mining and Knowledge Discovery*, 3:237–261, 1999.

[B.09] Moreno Montiel B. Tesis de maestría para el PCyTI. `http://tesiuami.izt.uam.mx/uam/aspuam/presentatesis.php?recno=14789&docs=UAMI14789.pdf`, 2009.

[BCRV00] Primo Becuzzi, Massimo Coppola, Salvatore Ruggieri, and Marco Vanneschi. *Parallelisation of C4.5 as a Particular Divide and Conquer Computation*, pages 382–389. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[Béj06] J. Béjar. Apuntes de aprendizaje automático. *Universitad Politécnica de Catalunya*, 2006.

[BK99] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1):105–139, Jul 1999.

[BM98] C. Blake and C. Merz. Uci repository of machine learning databases., 1998.

[BMP12] Dinabandhu Bhandari, C. A. Murthy, and Sankar K. Pal. Variance as a stopping criterion for genetic algorithms with elitist model. *Fundam. Inf.*, 120(2):145–164, April 2012.

[BP66] Leonard E. Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *Ann. Math. Statist.*, 37(6):1554–1563, 12 1966.

[Bre96] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, Aug 1996.

[CB01] Ronan Collobert and Samy Bengio. Svmtorch: Support vector machines for large-scale regression problems. *J. Mach. Learn. Res.*, 1:143–160, September 2001.

[CLV06] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Com07] Gemius Company. `http://www.gemius.com/`, 2007.

[Cor17] Fair Isaac Corporation. Falcon Fraud Manager. `http://www.fico.com/en/Pages/default.aspx`, 2017.

[CP98] Erick Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, 10(2):141–171, 1998.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[Dem06] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, December 2006.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[Die00] Thomas G. Dietterich. Ensemble methods in machine learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, pages 1–15, London, UK, UK, 2000. Springer-Verlag.

[Elk97] Charles Elkan. Boosting and naive bayesian learning. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 1997.

[ET93] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*. Macmillan Publishers Limited. All rights reserved, 1993.

[Fis36] Ronald Aylmer Fisher. The use of multiple measurements in taxonomic problems. *Annals Eugen.*, 7:179–188, 1936.

[Fly72] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960, Sept 1972.

[FM99] Yoav Freund and Llew Mason. The alternating decision tree learning algorithm. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 124–133, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[FN75] K. Fukunaga and P. M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Transactions on Computers*, C-24(7):750–753, July 1975.

[FP98] C. L. Fancourt and J. C. Principe. Modeling time dependencies in the mixture of experts. In *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*, volume 3, pages 2324–2327 vol.3, May 1998.

[FS97] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997.

[GCB+04] Hans P. Graf, Eric Cosatto, Léon Bottou, Igor Dourdanovic, and Vladimir Vapnik. Parallel support vector machines: The cascade svm. In *Advances in Neural Information Processing Systems 17*, pages 521–528. MIT Press, Cambridge, MA, 2004.

[GGP+17] Giraph, GraphLab, Phoebus, Spark, Piccolo, PageRank on Piccolo, and PageRank on HaLoop. Giraph: `http://incubator.apache.org/giraph/`, graphlab: `http://graphlab.org/`, phoebus: `https://github.com/xslogic/phoebus`, spark: `http://spark-project.org/`, piccolo: `http://piccolo.news.cs.nyu.edu/`, pagerank on piccolo: `http://kermit.news.cs.nyu.edu/browse.cgi/piccolo/file/6586de8af3de/src/examples/pagerank.pp`, haloop: `http://code.google.com/p/haloop/`, pagerank on haloop: `http://code.google.com/p/haloop/source/browse/#svn\%2Ftrunk\%2Fsrc\%2Fexamples\%2Forg\%2Fapache\%2Fhadoop\%2Fexamples\%2Fpagerank`,graphx: `http://spark.apache.org/docs/latest/graphx-programming-guide.html`, 2017.

[GMVC14] Fernando Gaxiola, Patricia Melin, Fevrier Valdez, and Oscar Castillo. Interval type-2 fuzzy weight adjustment for backpropagation neural networks with application in time series prediction. *Information Sciences*, 260:1 – 14, 2014.

[HA11] Tor Gunnar Houeland and Agnar Aamodt. *An Efficient Hybrid Classification Algorithm – An Example from Palliative Care*, pages 197–204. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[HJ11] Kai Hwang and Naresh Jotwani. *Advanced Computer Architecture, 3e.* McGraw-Hill Education, 2011.

[HO85] Larry V. Hedges and Ingram Olkin. Front matter. In *Statistical Methods for Meta-Analysis*, pages iii –. Academic Press, San Diego, 1985.

[Hol92] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* MIT Press, Cambridge, MA, USA, 1992.

[HORQFR04] José Hernández Orallo, María José Ramírez Quintana, and César Ferri Ramírez. *Introducción a la Minería de Datos.* Pearson Educación, 2004.

[HSTZ17] Ernst Moritz Hahn, Sven Schewe, Andrea Turrini, and Lijun Zhang. Synthesising strategy improvement and recursive algorithms for solving 2.5 player parity games. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 266–287. Springer, 2017.

[Hub92] Peter J. Huber. *Issues in Computational Data Analysis*, pages 3–13. Physica-Verlag HD, Heidelberg, 1992.

[HYY+16] Guo Haixiang, Li Yijing, Li Yanan, Liu Xiao, and Li Jinling. Bpso-adaboost-knn ensemble learning algorithm for multi-class imbalanced data classification. *Engineering Applications of Artificial Intelligence*, 49:176 – 193, 2016.

[JKK06] Mahesh V Joshiy, George Karypisy, and Vipin Kumary. Scalparc: A new scalable and e cient parallel classification algorithm for mining large datasets. *In: Intl. Parallel Processing Symposium*, 2006.

[KMHH11] Jitendra Kumar, Richard T. Mills, Forrest M. Hoffman, and William W. Hargrove. Parallel k-means clustering for quantitative ecoregion delineation using large data sets. *Procedia Computer Science*, 4:1602 – 1611, 2011. Proceedings of the International Conference on Computational Science, ICCS 2011.

[Koh95] Ron Kohavi. The power of decision tables. In *Proceedings of the 8th European Conference on Machine Learning*, ECML '95, pages 174–189, London, UK, UK, 1995. Springer-Verlag.

[KPL01] Vladimir Koltchinskii, Dmitriy Panchenko, and Fernando Lozano. Further explanation of the effectiveness of voting methods: The game between margins and weights. In *Proceedings of the 14th Annual Conference on Computational Learning Theory and and 5th European Conference on Computational Learning Theory*, COLT '01/EuroCOLT '01, pages 241–255, London, UK, UK, 2001. Springer-Verlag.

[LJL17] Wonji Lee, Chi-Hyuck Jun, and Jong-Seok Lee. Instance categorization by support vector machines to adjust weights in adaboost for imbalanced data classification. *Information Sciences*, 381:92 – 103, 2017.

[LL00] Hongjun Lu and Hongyan Liu. Decision tables: Scalable classification exploring rdbms capabilities. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 373–384, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[LYC+16] Haibiao Luo, Haojie Yuan, Shendong Cheng, Ying Li, Feng Yuan, and Mingzhu Wei. Parallelization of adaboost algorithm on intel mic architecture, 2016.

[Mac67] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathemati-*

cal Statistics and Probability, Volume 1: Statistics, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[Mar01] D. Marín. Tema 5: Análisis de cluster y multidimensional scaling. Universidad Carlos III de Madrid., 2001.

[MC14] Patricia Melin and Oscar Castillo. A review on type-2 fuzzy logic applications in clustering, classification and pattern recognition. *Applied Soft Computing*, 21:568 – 577, 2014.

[Mit97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[MMMM13] Benjamín Moreno-Montiel and Carlos Hiram Moreno-Montiel. Prediction system of larynx cancer. In *COMPUTATION TOOLS 2013, The Fourth International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking*, pages 23–30, 2013.

[MMMR11] Benjamín Moreno-Montiel and R. MacKinney-Romero. A hybrid classifier with genetic weighting. In *ICSOFT 2011 - Proceedings of the 6th International Conference on Software and Data Technologies*, volume 2, pages 18–21, 2011.

[MMMR13] B. Moreno-Montiel and R. MacKinney-Romero. Paraltabs: A parallel scheme of decision tables construction. In *2013 Mexican International Conference on Computer Science*, pages 47–54, Oct 2013.

[MMMR14] Benjamín Moreno-Montiel and René MacKinney-Romero. Parallel classification system based on an ensemble of mixture of experts. In *Proceedings of the 3rd International Conference on Pattern Recognition Applications and Methods*, pages 271–278, 2014.

[MMMRMM17] Benjamin Moreno-Montiel, René MacKinney-Romero, and Carlos Moreno-Montiel. Detection of genes in individual associated with laryngeal cancer using paraltabs. *IETE Journal of Research*, 2017.

[MU96] David J. Miller and Hasan S. Uyar. A mixture of experts classifier with learning based on both labelled and unlabelled data. In *NIPS*, 1996.

[NYD09] Thomas Ngo-Yel and Abhijit Dutt. A study on efficacy of ensamble methods for classification learning. *ICIS 2009 Proceedings*, page 69, 2009.

[PBV+10] Raúl Peralta, Michael Baudis, Guelaguetza Vazquez, Sergio Juárez, Rocío Ortiz, Horacio Decanini, Dulce Hernandez, Francisco Gallegos, Alejandra Valdivia, Patricia Piña, and Mauricio Salcedo. Increased expression of cellular retinol-binding protein 1 in laryngeal squamous cell carcinoma. *Journal of Cancer Research and Clinical Oncology*, 136(6):931–938, Jun 2010.

[PG09] Kemal Polat and Salih GüneşŸ. A novel hybrid intelligent method based on c4.5 decision tree classifier and one-against-all approach for multi-class classification problems. *Expert Systems with Applications*, 36(2):1587 – 1592, 2009.

[PMR14] L. Parra, E. Miranda, and A. Rocha. Proteja su dinero. CONDUSEF, `http://www.condusef.gob.mx/Revista/PDF-s/2011/132/blindaje_tarjetas_132.pdf`, 2014.

[Pol06] R. Polikar. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, Third 2006.

[PR12] I. Palit and C. K. Reddy. Scalable and parallel boosting with mapreduce. *IEEE Transactions on Knowledge and Data Engineering*, 24(10):1904–1916, Oct 2012.

[PRSMGH17] Juan M. Palomo-Romero, Lorenzo Salas-Morera, and Laura García-Hernández. An island model genetic algorithm for unequal area facility layout problems. *Expert Systems with Applications*, 68:151 – 162, 2017.

[Qui86] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[Ros61] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961.

[Rud94] G. Rudolph. Convergence analysis of canonical genetic algorithms. *Trans. Neur. Netw.*, 5(1):96–101, January 1994.

[Sch03] Robert E. Schapire. *The Boosting Approach to Machine Learning: An Overview*, pages 149–171. Springer New York, New York, NY, 2003.

[SM14] Daniela Sánchez and Patricia Melin. Optimization of modular granular neural networks using hierarchical genetic algorithms for human recognition using the ear biometric measure. *Engineering Applications of Artificial Intelligence*, 27:41 – 56, 2014.

[ST10] N Suguna and K Thanushkodi. An improved k-nearest neighbor classification using genetic algorithm. *International Journal of Computer Science Issues*, 7(2):18–21, 2010.

[SZL08] Shiliang Sun, Changshui Zhang, and Yue Lu. The random electrode selection ensemble for eeg signal classification. *Pattern Recognition*, 41(5):1663 – 1675, 2008.

[WFH11] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[WKRQ+08] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, Jan 2008.

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr 1997.

[Wol92] D Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

[Yia93] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 311–321, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[YS01] C Yu and DB Skillicorn. Parallelizing boosting and bagging. *Queen's University, Kingston, Canada, Tech. Rep*, 2001.

[ZXMO06] Yufang Zhang, Zhongyang Xiong, Jiali Mao, and Ling Ou. The study of parallel k-means algorithm. In *2006 6th World Congress on Intelligent Control and Automation*, volume 2, pages 5868–5871, 2006.

Apéndice A

Publicaciones del Doctorado

A continuación se enlistan cada uno de los artículos que fueron publicado en diversos congresos y revistas internacionales en el periodo del duración de esta investigación doctoral:

I. Benjamín Moreno-Montiel and R. MacKinney-Romero. *A hybrid classifier with genetic weighting*. In ICSOFT 2011 - Proceedings of the 6th International Conference on Software and Data Technologies, volume 2, pages 18-21, 2011.

II. Benjamín Moreno-Montiel and Carlos Hiram Moreno-Montiel. *Prediction system of larynx cancer*. In COMPUTATION TOOLS 2013, The Fourth International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking, pages 23-30, 2013.

III. B. Moreno-Montiel and R. MacKinney-Romero. Paraltabs: *A parallel scheme of decision tables construction*. In 2013 Mexican International Conference on Computer Science, pages 47-54, Oct 2013.

IV. Benjamín Moreno-Montiel and René MacKinney-Romero. *Parallel classification system based on an ensemble of mixture of experts*. In Proceedings of the 3rd International Conference on Pattern Recognition Applications and Methods, pages 271-278, 2014.

V. Benjamin Moreno-Montiel, René MacKinney-Romero, and Carlos Moreno- Montiel. *Detection of genes in individual associated with laryngeal cancer using paraltabs*. IETE Journal of Research, 2017.

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

ACTA DE DISERTACIÓN PÚBLICA

No. 00003
Matrícula: 2111802069

SISTEMA DE CLASIFICACIÓN PARALELO BASADO EN UN ENSAMBLE DE TIPO MEZCLA DE EXPERTOS

En la Ciudad de México, se presentaron a las 12:00 horas del día 30 del mes de agosto del año 2017 en la Unidad Iztapalapa de la Universidad Autónoma Metropolitana, los suscritos miembros del jurado:

DR. EDUARDO MORALES MANZANARES
DR. EDUARDO RODRIGUEZ FLORES
DRA. GRACIELA ROMAN ALONSO
DR. HUGO JAIR ESCALANTE BALDERAS
DR. RENE MACKINNEY ROMERO

Bajo la Presidencia del primero y con carácter de Secretario el último, se reunieron a la presentación de la Disertación Pública cuya denominación aparece al margen, para la obtención del grado de:

DOCTOR EN CIENCIAS (CIENCIAS Y TECNOLOGIAS DE LA INFORMACION)

DE: BENJAMIN MORENO MONTIEL

y de acuerdo con el artículo 78 fracción IV del Reglamento de Estudios Superiores de la Universidad Autónoma Metropolitana, los miembros del jurado resolvieron:

Aprobar

Acto continuo, el presidente del jurado comunicó al interesado el resultado de la evaluación y, en caso aprobatorio, le fue tomada la protesta.

BENJAMIN MORENO MONTIEL
ALUMNO

REVISÓ

LIC. JULIO CESAR DE LARA ISASSI
DIRECTOR DE SISTEMAS ESCOLARES

DIRECTOR DE LA DIVISIÓN DE CBI

DR. JOSE GILBERTO CORDOBA HERRERA

PRESIDENTE

DR. EDUARDO MORALES MANZANARES

VOCAL

DR. EDUARDO RODRIGUEZ FLORES

VOCAL

DRA. GRACIELA ROMAN ALONSO

VOCAL

DR. HUGO JAIR ESCALANTE BALDERAS

SECRETARIO

DR. RENE MACKINNEY ROMERO