Universidad de Huelva

Departamento de Tecnologías de la Información

Nuevas propuestas en el ámbito de los Algoritmos Genéticos Distribuidos para la Extracción de Reglas Clasificación

Memoria para optar al grado de doctor
presentada por:

Miguel Ángel Rodríguez Román

Fecha de lectura: 4 de febrero de 2016

Bajo la dirección del doctor:

Antonio Peregrín Rubio

Huelva, 2016

UNIVERSIDAD DE HUELVA

Nuevas Propuestas en el Ámbito de los Algoritmos Genéticos Distribuidos para la Extracción de Reglas de Clasificación

Tesis Doctoral

Miguel Ángel Rodríguez Román

Huelva, Noviembre de 2015

Departamento de Tecnologías de la Información

UNIVERSIDAD DE HUELVA

Nuevas Propuestas en el Ámbito de los Algoritmos Genéticos Distribuidos para la Extracción de Reglas de Clasificación

MEMORIA QUE PRESENTA

Miguel Ángel Rodríguez Román

PARA OPTAR AL GRADO DE DOCTOR EN INFORMÁTICA

Noviembre de 2015

DIRECTOR

Dr. Antonio Peregrín Rubio

Programa de Doctorado

Tecnologías Informáticas Avanzadas

Departamento de Tecnologías de la Información

La memoria titulada "*Nuevas Propuestas en el Ámbito de los Algoritmos Genéticos Distribuidos para la Extracción de Reglas de Clasificación)*", que presenta Miguel Ángel Rodríguez Román para optar al grado de doctor, ha sido realizada dentro del programa de doctorado "*Tecnologías Informáticas Avanzadas*" del Departamento de Tecnologías de la Información de la Universidad de Huelva bajo la dirección del doctor D. Antonio Peregrín Rubio.

Huelva, Noviembre de 2015

El Doctorando

Fdo.: Miguel Ángel Rodríguez Román

El Director

Fdo.: Dr. Antonio Peregrín Rubio

Agradecimientos

"Cada día sabemos más y entendemos menos."

Albert Einstein

Me atrevo a aventurar que Albert Einstein llegó a esta conclusión después de haber dado al mundo alguna de las teorías que cambiaron la visión estática de la física de principios del siglo pasado.

En el desarrollo de esta tesis, he sentido a menudo la sensación de que cuanto más sabía acerca del objeto de mi investigación, mayor era el número de preguntas y dudas que se alzaban acerca del comportamiento de los algoritmos.

He de agradecer que en este continuo intento de comprender la naturaleza de las cosas, mi director, Antonio Peregrín, haya tenido siempre un oído atento a las teorías, y prestado su tiempo a experimentos que no siempre han servido para avanzar en la dirección oportuna. Más aún, ha sabido encauzar las experiencias aprendidas en nuevos enfoques en una paciencia digna de elogio.

Gracias también a Paco Herrera y a su equipo de la Universidad de Granada, que siempre han estado presentes en la resolución de dudas, apoyando con la experiencia aprendida y los resultados conseguidos sin pedir nada a cambio. Dentro de este equipo reseñar a Salvador García que tantas veces nos ha guiado en el mundo de los estadísticos y cuyo artículo al respecto ha marcado un antes y un después. Asimismo dedicar una atención especial a José Manuel Benítez, cuyos consejos en paralelización de algoritmos valen su peso en oro.

Y finalmente, pero no por ello menos importante, dedico esta tesis a mi mujer, Ana cuya paciencia y apoyo incondicional a lo largo de estos años ha permitido que este trabajo llegue finalmente a su fin.

ÍNDICE

Tablas

Figuras

Algoritmos

Acrónimos

AG……	Algoritmo Génetico
AGL	Algoritmo Génetico Local
BB's.....	Building Blocks
DLF……..	Data Learning Flow
EDGAR.....	Efficient Distributed Genetic Algorithm for Rule Extraction
FP.....	False Positive
GM.....	Geometric Mean
IR.....	Imbalanced Ratio
KDD.....	Knowledge Discovery in Databases
MD.....	Minería de Datos
NB……..	Clases No Balanceadas
NBC……..	Clases No Balanceadas valores Continuos
TP.....	True Positive

Introducción

A. Planteamiento

La revolución digital propicia tanto que la captura automática de datos sea factible como que su coste de almacenamiento resulte aceptable. La celeridad en la informatización de los negocios junto al no menos destacable desarrollo del software y del hardware, favorecen que un ingente volumen de datos sea recogido y almacenado en bases de datos.

Los datos por sí solos no producen un beneficio directo, sino que su verdadero atractivo, radica en la posibilidad de extraer información valiosa, bien para la toma de decisiones, bien para el entendimiento del fenómeno que da origen a los ya citados datos. Llegados a este punto, surge por tanto la necesidad de plantear metodologías enfocadas al análisis de datos cuyo fin no es otro que el de descubrir un conocimiento útil. En este contexto se encuentra la *Minería de Datos [Tan 2005]* o *Data Mining* (MD en adelante), que busca patrones de relación en los datos. Ya en 2006 [Yang 2006] estableció una lista de 10 problemas de investigación que aún representan hoy en día [Fernández 2011] [Bekkerman 2011] un desafío abierto dentro de esta disciplina:

1. *Developing a unifying theory of data mining*
2. *Scaling up for high dimensional data and high speed data streams*
3. *Mining sequence data and time series data*

4. *Mining complex knowledge from complex data*
5. *Data mining in a network setting*
6. *Distributed data mining and mining multi-agent data*
7. *Data mining for biological and environmental problems*
8. *Data Mining process-related problems*
9. *Security, privacy and data integrity*
10. *Dealing with non-static, unbalanced and cost-sensitive data*

De este conjunto cabe destacar los problemas relacionados con la complejidad y con la capacidad de tratamiento distribuido de los datos:

- De alta dimensionalidad
- Complejos
- Sobre redes de ordenadores
- No estáticos y no balanceados

La dificultad para generar un modelo, dentro del contexto del aprendizaje supervisado, depende en gran parte de la estructura de los datos así como del volumen de los mismos [Freitas 1998]. De una parte, a medida que la dimensionalidad aumenta, el tiempo de aprendizaje se incrementa a niveles en donde la tarea puede resultar irrealizable en un tiempo admisible.

De otra parte, resulta primordial reseñar que una gran parte del conocimiento extraído de manera automática corresponde a medidas numéricas. La mayoría de los algoritmos de aprendizaje simplifican estas medidas en rangos con el fin de disminuir la complejidad del proceso. Este proceso no obstante no está exento de riesgos, ya que la mayoría de las simplificaciones no suelen tener en cuenta las relaciones entre los atributos (datos *multivariate*). En este sentido hay nuevas aportaciones [Ramírez-Gallego 2015] [García 2013] que muestran el interés de este problema.

Respecto al aprendizaje supervisado sobre conjuntos de datos no balanceados; éste es un problema de alta complejidad [Chawla 2204][Sun 2009] que resulta de gran interés, dado que la inmensa mayoría de los problemas suele tener en mayor o menor grado clases no balanceadas.

En las estrategias para aprender sobre conjuntos que tienen alguna característica que dificulta el aprendizaje (clases no balanceadas, características numéricas complejas, alta dimensionalidad) siguen dos enfoques [Barandela 2003] [McQueen 1967]. Bien se procesa el conjunto de datos (nivel de datos) para adaptarlo a las condiciones de aprendizaje del algoritmo existente (reducción de datos, discretización, rebalanceo de las clases). Bien se diseña el algoritmo (nivel algorítmico) para tener en cuenta estas características dentro de su lógica de aprendizaje.

El nivel de datos es una de las opciones más utilizadas, al permitir el uso de algoritmos de clasificación ya conocidos de antemano. Sin embargo ésta opción puede producir pérdida de precisión en las predicciones del modelo generado; esto se debe, entre otras causas, a la quimera de un algoritmo de preprocesamiento válido para cualquier tipo de fuente y algoritmo de aprendizaje. Recientemente, en [CCHJ08], se ha mostrado la relación empírica existente entre el tratamiento de los problemas de clasificación no balanceados con propuestas a nivel de datos y propuestas a nivel algorítmico. En [García 2013] se muestra el impacto de la discretización sobre un conjunto de datos y de algoritmos de clasificación.

El nivel algorítmico aplicado a la alta dimensionalidad requiere la escalabilidad en datos [Freitas 1998]. Esta opción tiene, entre otras ventajas, la disminución del coste económico causa del empleo de ordenadores de menor capacidad, y como principal finalidad la reducción del tiempo de proceso.

Los algoritmos genéticos son fácilmente paralelizables [Holland 1986] gracias a la estructura implícita basada poblaciones que permite comunicar poblaciones en distintos procesadores manteniendo la estructura básica del algoritmo genético. La aplicación de los algoritmos genéticos paralelos [Cantú-Paz 2000] a los algoritmos genéticos de aprendizaje supervisado [Janikow 1993] [Leung 1995] [Kennedy 1999] [Bacardit 2003] [Augier, 1995], permite nuevas técnicas de distribución de datos [Freitas 1998]. Los algoritmos genéticos distribuidos para el aprendizaje [Flockhart 1995] [Giordana 1994] [Anglano 1998] facilitan el tratamiento simultáneo de procesos paralelos y de particionamiento de datos de una manera menos instrumental que la mera paralelización de algoritmos existentes [Alba 1999] [Alba 2002].

Existen pocas propuestas evolutivas que traten los problemas de complejidad de los datos y alta cardinalidad de manera distribuida. Por este motivo la presente memoria se centra en proponer nuevas estrategias de aprendizaje distribuido competitivas con las mejores propuestas existentes en el ámbito de la clasificación supervisada.

B. Objetivos

En base al planteamiento realizado en la subsección anterior el presente trabajo tiene como objetivo proponer mejoras tanto en la escalabilidad como en la precisión en el ámbito de los algoritmos genéticos distribuidos para la clasificación. Para desarrollar este objetivo general se definen los siguientes objetivos particulares:

- Mejorar el tratamiento de datos de alta dimensionalidad. Se pretende mejorar la escalabilidad del modelo distribuido eliminando la interacción directa de un nodo supervisor y dotando al modelo de una dinámica orientada a datos. Se propone asimismo el fraccionamiento del conjunto de datos en distintos procesadores con algoritmos genéticos idénticos y técnicas de equilibrado descentralizadas para mantener el sistema escalable.
- Mejorar la precisión en conjuntos de datos con clases no balanceadas. Se pretende desarrollar métodos propios de tratamiento de las clases no balanceadas realizando una combinación de copia de las clases positivas por distribución en distintos nodos y métodos de sensibilidad al coste del error en la clasificación de ejemplos de dicha clase.
- Mejorar la precisión de datos con atributos numéricos continuos. Se proyecta desarrollar técnicas de aprendizaje de las particiones para mejorar la precisión en los conjuntos de datos con atributos de carácter numérico mediante el uso del clasificador como validación de las particiones en el proceso de aprendizaje de las mismas.

C. Resumen

Para desarrollar los objetivos propuestos, este trabajo se estructura en 4 capítulos, una sección de comentarios finales y dos apéndices. La estructura de cada una de estas partes se introduce brevemente a continuación

El Capítulo 1 contiene una introducción a este trabajo con el objeto de orientar al lector sobre el tipo de problemas que pretendemos resolver. En el caso de esta memoria, trataremos la mejora de la precisión en la clasificación de conjuntos de datos con algoritmos genéticos distribuidos. Para ello se introducen los AG distribuidos para el aprendizaje describiendo en detalle las características de cada estrategia de paralelización y la estructura de aprendizaje en el ámbito de los AG.

En el Capítulo 2, se presenta una propuesta distribuida de aprendizaje de reglas mediante un modelo evolutivo. En la primera Sección se describe el modelo de distribución seguido en base a la tipología descrita en el Capítulo 1. En la segunda Sección se describen de manera detallada los componentes del algoritmo genético de aprendizaje. Finalmente en la tercera Sección se presenta un estudio experimental y se muestran los resultados obtenidos tras ser validados éstos mediante el uso de test estadísticos no paramétricos. El uso y aplicabilidad de éstos se explica en el apéndice A al final del documento.

En el Capítulo 3, se presenta una propuesta para mejorar el tratamiento de las clases no balanceadas en el ámbito de los algoritmos genéticos distribuidos. Para ello inicialmente se describe la problemática asociada a la clasificación de este tipo de tipologías de datos y el efecto de la distribución de datos sobre la misma. En la segunda Sección se muestran las propuestas de diseño para hacer compatible la distribución de datos con la correcta clasificación de clases no balanceadas. Finalmente se desarrolla un estudio experimental en dos partes. En la primera parte se realiza un estudio comparativo con otros algoritmos distribuidos y no distribuidos del ámbito de la clasificación. En la segunda parte se compara con técnicas de preprocesamiento aplicadas a los mismos algoritmos de la primera parte.

En el Capítulo 4, se presenta una propuesta para mejorar la precisión de conjuntos de datos de carácter numérico continuo. Para ello inicialmente se describen las técnicas de preprocesamiento necesarias para tratar con datos

numéricos y los efectos sobre la precisión de los resultados. En la segunda Sección se proponen técnicas de aprendizaje de las particiones como un proceso de aprendizaje interno basado en el algoritmo genético CHC con codificación real. Se describe en esta Sección los operadores de este algoritmo y su relación con el algoritmo genético de aprendizaje de reglas del modelo distribuido. Finalmente se muestra un estudio experimental que compara los resultados obtenidos con los obtenidos en el Capítulo 3 para el mismo conjunto de algoritmos. Al igual que en resto de estudios experimentales en este trabajo, se hace uso de los tests no paramétricos para validar los resultados.

En la Sección de Comentarios Finales se resumen los resultados obtenidos en esta memoria así como las principales conclusiones a las que hemos llegado en la investigación. Esta Sección incluye las líneas futuras de investigación que pretenden ser la continuación de este trabajo. Entre ellas cabe destacar la adaptación del modelo distribuido al paradigma *MapReduce*.

Se incluye un apéndice dedicado a la descripción de los test estadísticos no paramétricos utilizados en las secciones de experimentación de esta memoria y otro apéndice con las tablas de la experimentación detallada en escalabilidad.

Capítulo 1

Preliminares: Algoritmos Genéticos Distribuidos de Clasificación

Las tareas de clasificación sobre conjuntos de datos de alta cardinalidad se han convertido hoy en día una prioridad dado que cada vez somos capaces de generar y almacenar más datos de manera automática y es necesario encontrar metodologías que puedan extraer la máxima utilidad sin que sea preciso un preprocesamiento de los mismos. [Fayyad 1996]. Esta necesidad de automatización del proceso de extracción de conocimiento debe llevarnos a proponer algoritmos de minería de datos que incorporen la inteligencia necesaria para que sea posible la automatización de la extracción de conocimiento.

El tratamiento de alta dimensionalidad en algoritmos de aprendizaje se puede tratar bien, con estrategias de muestreo de datos o bien, con soluciones algorítmicas [Fayyad & Piatetsky 1996]. La reducción de datos puede tener efectos en la calidad de la solución [Quinlan 1991], por tanto no es de extrañar que haya un gran interés en desarrollar soluciones capaces de manejar un mayor volumen de datos. Respecto a las soluciones algorítmicas, la opción de tratamiento de los datos ha pasado en pocas décadas de las soluciones basadas en *mainframe*, pasando por arquitecturas paralelas más o menos dedicadas y finalmente a soluciones distribuidas sobre maquinas independientes [Park 2002].

En este Capítulo se revisan brevemente las teorías que sustentan los modelos distribuidos, y de una manera más concreta y en profundidad, aquellas que guardan relación con los algoritmos genéticos y el aprendizaje de reglas de clasificación.

La Sección 1.1 está dirigida a revisar alguno de los paradigmas en el paralelismo asociado a la minería de datos, analizando la importancia del particionamiento de datos en las arquitecturas distribuidas. La Sección 1.2 estudia las características de los algoritmos genéticos distribuidos. En la Sección 1.3 se realiza una taxonomía de los principales algoritmos genéticos de clasificación tanto distribuidos como no distribuidos. En la Sección 1.4 se describen las características que determinan el rendimiento de una arquitectura distribuida.

1.1 Principales Estrategias de Distribución en el Aprendizaje Supervisado

Los problemas de alta cardinalidad devienen un reto para los algoritmos de aprendizaje supervisado. La inducción de reglas en base a ejemplos tiene un coste computacional ligado a la cardinalidad del mismo, y en consecuencia para estos casos se presume que la reducción de datos sea la alternativa asumida para trabajar con los algoritmos de clasificación clásicos. Esta alternativa resulta en una calidad muy variable de los resultados [Freitas 1998], y en dependencia tanto de la tipología del conjunto de datos como del tipo de algoritmo de aprendizaje y de reducción de datos utilizado.

Otra aproximación al problema consiste en distribuir el conjunto de datos, para que el algoritmo de aprendizaje sea capaz de trabajar sobre conjuntos de datos mayores en tiempos razonables. Esta estrategia es denominada paralelización de datos o espacial [Hwang 1993], consistente en dividir el conjunto de datos a procesar en particiones de menor tamaño (no necesariamente disjuntas). De tal manera, aplicado a la MD, la partición puede ser realizada bien sobre las instancias (partición horizontal) bien sobre las características (partición vertical).

Respecto a los algoritmos genéticos, los algoritmos distribuidos, que trabajan sobre estos datos particionados, no originan los mismos resultados que sus versiones compactas [Cantú-Paz 2000]. Ahora bien, existen otros argumentos que justifican que los algoritmos genéticos distribuidos pueden alcanzar un mejor comportamiento que sus versiones no distribuidas [Alba 1999], ya sea en parte por una mayor eficiencia [Gordon 1993] o por la capacidad de mantener la diversidad de la población durante el proceso, evitando caer en el problema de los óptimos locales [Lin 1994].

La división de los datos en algoritmos de aprendizaje distribuidos posee unas características entre las que cabe destacar que cada procesador llegará a soluciones locales sustentadas en los datos asignados al mismo. Una vez procesados los datos locales, los resultados han de ser tratados con el objetivo de formar un modelo global coherente. Los modelos clásicos para crear el modelo global son:

- Integración [Fayyad 1996], genera un modelo global que es un subconjunto de las reglas aportadas por los modelos locales, abarcando la totalidad de los datos.

- *Meta-Learning*: [Chan 1995], el cual se sirve de cada modelo local como una caja negra, ponderando el valor de clasificación de cada uno para dar una predicción, no obstante a medida que el número de conjuntos de datos aumenta, disminuye la precisión.

- Cooperación inter-procesador: [Provost 1994], si un modelo local encuentra una regla, la comparte dinámicamente con el resto de los nodos, originando entonces una regla válida globalmente.

1.1.1 Propiedad Extendida de la Invarianza en Particiones de Datos

La estrategia de cooperación entre nodos se sustenta en la propiedad de invariancia de particiones de datos, que de forma resumida, establece que una regla local puede ser válida en un conjunto de datos global y que en consecuencia una regla global lo será también en un conjunto de datos local, siendo susceptible de ser localizada en éste. Este modelo posibilita la generación de un modelo global mediante la integración de reglas deducidas en distintos conjuntos de datos [Fayyad 1996] y la colaboración entre nodos [Provost 1994].

El aprendizaje de una regla de clasificación sobre un conjunto de datos, fracción de un conjunto global, es capaz de ocasionar reglas de alto valor, éstas, sin embargo, no son válidas una vez aplicadas al conjunto global. Por otro lado, una

regla globalmente válida lo será también en un subconjunto de los datos y por tanto, debe ser posible aprenderla de un subconjunto de datos. A continuación se desglosa el razonamiento matemático de dicha regla [Provost 1994],

Una regla *r*, sobre un conjunto de datos *E*, produce, por un lado, un conjunto de casos, cubiertos positivamente (TP = *True Positives*) y por otro lado, un conjunto de datos cubiertos, en los que no coincide la clase prevista (FP = *False Positives*).

Se evalúa una regla sobre el conjunto de datos, a través de una función que calibra la capacidad de predicción de la misma y un límite preestablecido *c*, de manera que se diga entonces que *r* es satisfactoria sobre *E* si $f(r,E) \varepsilon c$. La capacidad de predicción positiva será:

$$f(r,E) = ppv(r,E) = TP/(TP+FP) \quad (1.1)$$

[Quinlan 1994] establece la medida:

$$f(r,E) = cf(r,E) = (TP-0.5)/(TP+FP) \quad (1.2)$$

La estimación de Laplace [Webb, 1995]:

$$f(r,E) = le(r,E) = (TP+1)/(TP+FP+k) \quad (1.3)$$

Siendo k número de clases.

Si se fracciona el conjunto de datos *Ei, i=1..N*, la propiedad de invarianza de particiones (*invariant-partitioning property*) [Provost 1994], establece que si una regla *r* es satisfactoria para *E*, existe un *i* tal que *r* es satisfactoria en *Ei*. Según esta propiedad, toda regla que pueda ser hallada en un conjunto de datos, de igual manera podrá ser localizada en alguna de sus particiones, afín al mismo criterio de evaluación.

Resulta inmediato probar esta propiedad en los valores positivos de la regla, es decir, en aquellos en los que la clase de la regla se cumple en las instancias que posean el antecedente, pero no para los valores de (1.2) y (1.3), para los que es necesario un criterio menos restrictivo: f'(r,E,N), función que define como aceptable una regla si se concluye en la partición que $f'(r,Ei,N) \geq c$, dicho de otra forma, que una regla satisfactoria en un conjunto central, será aceptable en alguna de sus particiones.

Para Laplace se define

$f'(r,E,N) = le'(r,E,N) = (TP+1/N)/(TP+FP+k/N)$

Lo que significa que el criterio usado en el subconjunto es similar al empleado en el conjunto central

$$para\ N=1,\ le'(r,E,N) = le(r,E);$$

$$as\ N\rightarrow\infty,\ le'(r,E,N) \rightarrow ppv(r,E,N)$$

$$N>1,\ le(r,E) < le'(r,E,N) <= ppv(r,E) \quad (1.4)$$

Si se asume que para la regla

$$r: (TP+1)/(TP+FP+k)\ \varepsilon\ L \quad (1.5)$$

y no se cumple que en una partición de N subconjuntos de E

$$\forall i,\ (TP_i+1/N)/(TP_i+FP_i+k/N) > L \quad (1.6)$$

es decir, que no es aceptable en ninguna partición E_i, entonces

$1)\forall i\ \{TP_i + 1/N < L \cdot (TP_i + FP_i + k/N)\}$

$2)\Sigma(TP_i+1/N) < \Sigma\ (L \cdot (TP_i+FP_i + k/N))$

$3)\ \Sigma\ (TP_i+1/N) < L \cdot (TP_i+FP_i + k/N)$

$4)\ TP + 1 < L \cdot (TP + FP + k)$

5) $(TP+1)/(TP+FP+k) < L$ ==> Contradicción (1.7)

A través de una demostración similar se extiende la propiedad a otras funciones similares de evaluación.

Apoyándose en esta propiedad, este trabajo propone diseñar un algoritmo para conservar las mejores soluciones aportadas por cada modelo local, a fin de generar un modelo global, compuesto de las reglas de superior valor para el conjunto global de datos.

1.2 Características de los Algoritmos Genéticos Distribuidos

Uno de los principales problemas de diseño en los algoritmos genéticos paralelos y distribuidos es la colaboración de las distintas partes en la consecución de los objetivos finales [Alba 1999]. Dependiendo del rol que toman los nodos podemos encontrar dos enfoques.

- Algoritmos genéticos paralelos de control [Alba 1999], éstos están compuestos por uno o varios procesadores que dirigen las tareas sobre el resto de procesadores. Tanto a necesidad de sincronización como el flujo de la información limitan la escalabilidad, a medida que el número de elementos del sistema aumenta [Hockney 1988].
- Algoritmos genéticos descentralizados (tipo isla o *demes)* [Cantú-Paz 2000], éstos algoritmos carecen de un algoritmo supervisor que organice el conjunto de tareas a efectuar por los nodos. Están compuestos por un número de nodos que ejecutan un algoritmo genético, nodos que a su vez comparten individuos cada cierto número de iteraciones. La figura 1.1 muestra un algoritmo genético cuyas poblaciones se encuentran distribuidas siguiendo este modelo.

Las principales características del modelo de islas son:

- Estrategias de mantenimiento de la diversidad: Los AGs deben mantener un equilibrio entre la presión selectiva y la exploración [Goldberg 1987]. En un AG con población única, el mantenimiento de la diversidad depende del tamaño de la población y de técnicas de mantenimiento de la diversidad (mutación, re-start, etc...). Grosso [Grosso 1985] sostiene que n poblaciones aisladas tienen peor comportamiento que una población única, no obstante n poblaciones comunicadas muestran mejor comportamiento que una población única, en términos de mantenimiento de la diversidad genética de las mismas.

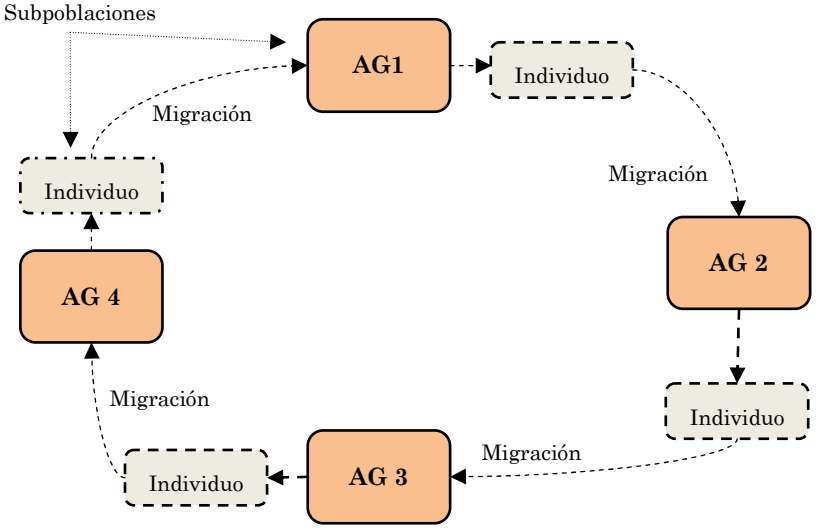

Figura 1.1 Migración de individuos en GA multi poblacional.

- Importancia de la migración: el ratio, la vecindad y la frecuencia de migración influyen en el mantenimiento de la diversidad. Un porcentaje de migración alto provoca una perdida rápida de diversidad. Sin embargo, un porcentaje de migración muy bajo puede ralentizar la convergencia hacia una solución de calidad [Cantú-Paz 2000].
- Heterogeneidad de los nodos: usualmente los algoritmos de cada nodo son idénticos, pero es factible tener diferentes operadores genéticos, o diferentes parámetros de cruce o mutación en cada nodo. Esta heterogeneidad en los nodos conlleva diferentes velocidades de convergencia, que permiten combinar la capacidad de explotación y el mantenimiento de la diversidad en el conjunto global de la población [Tanesse 1989].
- Aplicabilidad del modelo: Los algoritmos genéticos se pueden transformar en algoritmos distribuidos con un desempeño favorable, siempre que la función objetivo se pueda separar en partes evaluables

independientemente en cada nodo [Cantu-Paz 2000]. Por citar un ejemplo, la inducción de reglas mediante algoritmos genéticos puede ser expresada como una función separable. En este caso, una regla es una hipótesis acerca de los valores que pueden tomar los atributos en relación con la clase predicha. Un individuo en un AG de aprendizaje estará formado por una o varias reglas. La migración de individuos llevará a éste a una población receptora, donde generará nuevos descendientes, a través del cruce con los individuos más aptos de su población.

1.3 Algoritmos Evolutivos en el Aprendizaje de Reglas

Los algoritmos evolutivos para el aprendizaje de reglas de clasificación [Freitas 2002] son un tipo de algoritmos de aprendizaje automático (GBML del Inglés *Genetics-Based Machine* Learning), que utilizan reglas para la representación del conocimiento. Los algoritmos evolutivos inducen modelos de reglas en espacios de búsqueda complejos, sin conocimiento previo del dominio del problema [Eiben 2003]. Los algoritmos buscan soluciones en el espacio de las posibles reglas para construir sistemas clasificadores [Holland 1975], minimizando el error cometido en la clasificación.

Los espacios de búsqueda de los algoritmos de clasificación basados en reglas dependen del número de valores posibles que pueden tener las características. Un conjunto de datos D, con instancias de n características $C=c_1, c_2,..., c_n$ y d valores dentro de un dominio, donde cada característica *cumple* $c1 \in d1$, tendrá un espacio de búsqueda de tamaño $t = d_1 * d_2 * ..... * d_n$;

A continuación, vamos a describir las principales estrategias de codificación de los GBML, con el propósito de enmarcar el modelo propuesto y describir con mayor exactitud las características del mismo. Respecto a la representación del cromosoma, [Fernández 2010] establece la siguiente taxonomía:

Cromosoma = Conjunto de Reglas: un individuo es la representación completa de la solución propuesta o tipo *Pittsburg* [DeJong 1993]. Siguiendo el esquema

canónico, la población final estará compuesta por individuos con poca diferenciación genética, en la que cada uno de ellos es una solución. En este caso será un conjunto de reglas que componen un clasificador. Las reglas colaboran entre sí al formar parte del mismo esquema y se pueden introducir criterios de longitud en la evaluación, para priorizar una solución con menos reglas. En cambio tiene como principal inconveniente el alto coste computacional del cálculo de la función de evaluación. Por otro lado al tener una longitud variable el uso de operadores de cruce debe ser adaptado. Son ejemplos de este modelo GIL [Janikow 1993], GLPS [Leung 1995] y STEPS [Kennedy 1999], GA-MINER [Flockhart 1995], GASSIST [Bacardit 2003].

Cromosoma = regla: un individuo es la representación de una regla. El conjunto final de la población deberá ser procesado para extraer el conjunto de reglas que forme el clasificador. Dentro de esta representación hay tres enfoques para guiar la búsqueda:

- *Michigan* [Holland, 1986], El conjunto de reglas recibe de manera incremental ejemplos de entrenamiento y usa el resultado del clasificador actual como estrategia de aprendizaje reforzado. El GA se encarga de detectar nuevas reglas que reemplacen aquellas de peor calidad del clasificador actual.
- IRL (Iterative Rule Learning): Siguiendo la estrategia de nichos para múltiples soluciones, en cada ejecución se elimina la regla y los datos que cubre hasta que todos los datos están cubiertos. SIA01 [Augier 1995], es una propuesta que sigue esta estrategia.
- Enfoque GCCL (*Genetic Cooperative-Competitive Learning*): La población completa evoluciona y el clasificador estará compuesto por un subconjunto de los individuos. Los cromosomas compiten por los ejemplos a cubrir y cooperan en la formación del clasificador. REGAL [Giordana 1994], G-NET [Anglano 1998] y DOGMA [Hekanaho 1998], son algoritmos representativos de esta representación

Respecto a la estrategia de control de datos en los algoritmos distribuidos existen varias corrientes:

- Paralelización Global: El principal exponente de esta corriente es GA-PVMMINER [Araujo 1999], basado en un cálculo de la función de

evaluación en un clúster de maquinas heterogéneas bajo el *framework Parallel Virtual Machine*.

Paralelización de grano fino: GA-MINER [Flockhart 1995]: Es un algoritmo paralelo basado en algoritmos genéticos jerárquicos que evoluciona un conjunto de reglas tipo Pittsburgh.

Paralelización de grano grueso: los algoritmos más representativos de este enfoque son REGAL [Giordana 1994] y Now-GNET [Anglano 1998].

REGAL [Giordana 1994]: es un algoritmo híbrido de control y datos sobre el esquema CCGL. REGAL trata de generar reglas que cubren a los ejemplos presentes en los nodos locales, enviándolas posteriormente a un nodo central que hace la función de supervisor. Este nodo selecciona las mejores reglas junto con las instancias que cubre para ser optimizadas mediante un algoritmo genético en otro nodo. Es un algoritmo muy adaptado a este modelo de representación en el que se consiguen clasificadores muy compactos y de alta calidad. Tiene algunas limitaciones derivadas de su diseño que restringen su aplicabilidad:

- Manejo de datos nominales: Se necesita realizar una discretización previa de los datos para orígenes de carácter numérico continuo.
- Modelo de control síncrono centralizado: La escalabilidad depende del número de nodos y las condiciones de latencia en la red [Barrezta 2003]
- Numero de comunicaciones: el proceso de control centralizado realiza frecuentes movimientos de reglas y conjuntos de datos entre el supervisor y los nodos.

G-Net [Anglano 1998], es un algoritmo que hereda de REGAL la representación y alguno de sus operadores. Tiene un sistema de reparto de las tareas más desarrollado con nodos evaluadores y nodos aprendedores en un sistema híbrido de control y datos. Está fuertemente inspirado en el principio mínimo de longitud, por lo que genera reglas y clasificadores muy compactos. Comparte con REGAL las características de rendimiento paralelo con algunos matices. La principal diferencia entre ambos algoritmos reside en la exploración y validación de reglas. Mientras que REGAL procesa las reglas y sus ejemplos en los nodos aprendedores, G-net divide esta tarea en dos roles, los nodos genéticos básicos y los nodos

evaluadores encargados de la función de evaluación de cada regla. Así pues G-net realiza más operaciones entre procesadores que REGAL pero de menor peso, por lo que puede competir mejor en entornos donde la velocidad de comunicación es baja [Barrezta 2003].

1.4 Rendimiento de una Arquitectura Distribuida

Esta Sección describe brevemente las características que determinan el rendimiento de una arquitectura distribuida. Para concluir se analizan las condiciones para determinar la posible mejora de un modelo paralelo frente a un modelo compacto en base a las características descritas.

La principal diferencia entre un modelo compacto y un modelo sustentado en una arquitectura paralela reside en la necesaria colaboración de las partes del modelo paralelo. El rendimiento esperado de la colaboración entre componentes depende de la forma en que se sincronizan las tareas así como el impacto de la velocidad de comunicación [Grama 1994].

En relación de la sincronización entre procesadores, ésta puede ser síncrona o asíncrona. En los sistemas paralelos síncronos se establecen esperas entre procesadores que resultan en tiempos inactivos. Mientras que la comunicación asíncrona no tiene tiempos inactivos *per se*, los procesos pueden incurrir en esperas debido a la necesidad de precedencia de alguna tarea. Es por esto que este tipo de sincronización requiere de mecanismos de control para coordinar adecuadamente el orden de las comunicaciones.

En cuanto a la velocidad de la comunicación, el rendimiento de una arquitectura esta determinado sobre todo por la cantidad de información que debe moverse entre los procesadores y la frecuencia con que la información se transfiere de un procesador a otro.

En base a estos criterios, el cálculo del rendimiento máximo esperado para un modelo paralelo se expresa como:

El rendimiento o *speed up* (*Sp*) de un algoritmo paralelo se compara con la implementación secuencial más rápida del mismo. Sea T_1 el tiempo de la versión

secuencial en un procesador, y T_p el tiempo del algoritmo paralelo con p procesadores [Barrezta 2003], el speed up se define como (1.8):

$$Sp = \frac{T_1}{T_p} \qquad (1.8)$$

El hardware asociado a cada procesador, secuencial o paralelo, debe ser equivalente, exceptuando la memoria, en la que para los procesadores paralelos será [Quinn 1994]:

$$memoria\_nodo = 1/p\ memoria\_secuencial \qquad (1.9)$$

Si todas las tareas fuesen independientes y el coste de la comunicación fuese cero, el sistema paralelo presentaría un S_p lineal respecto al número de procesadores, en la práctica el Sp es inferior, siendo mayor la diferencia a medida que crece el número de procesadores. Los principales parámetros en el cálculo de la escalabilidad son:

- Tiempo inactividad (*idle time*): tiempo que un procesador no está realizando trabajo productivo a causa de la espera del resultado de otro procesador.

- Latencia de la red: en estructuras de memoria no compartida, los procesadores se envían información a través de un medio con un cierto ancho de banda y una latencia de respuesta.

- Balanceo de la carga**:** una incorrecta repartición de la carga de trabajo aumenta el tiempo de inactividad, en los procesos con dependencia de tareas (sincronización de tareas). Esta situación puede deberse a una desigual repartición de las tareas o a una arquitectura no homogénea (*grids* de ordenadores, computación distribuida sobre IP, etc...

1.5 Consideraciones Relativas a la Complejidad de los Datos

En el aprendizaje supervisado, el origen de datos puede tener características de complejidad que hagan más difícil la generalización de conceptos. En el Capítulo anterior, hemos enumerado alguna de ellas, como las clases no balanceadas o el manejo de atributos numéricos. Como ha sido anteriormente mencionado, desde el punto de vista del aprendizaje supervisado, hay dos estrategias para tratar con datos de estas características. La primera de ellas, a nivel de datos, realizando un preprocesamiento que permite utilizar un algoritmo de aprendizaje estándar; la segunda, a nivel algorítmico, mediante una codificación especifica que las considere.

Por ello, en primer lugar, en el Capítulo 2, se propone un modelo evolutivo distribuido para tratar conjuntos de datos de alta cardinalidad. Posteriormente, en el Capítulo 3, describimos una adaptación algorítmica del mismo para tratar con clases no balanceadas y finalmente, en el Capítulo 4, dada la relevancia del proceso de discretización en los algoritmos de aprendizaje, se propone una transformación del modelo para tratar atributos numéricos continuos.

En este contexto, pensamos que hay algunas características de los datos que dificultan el aprendizaje, como el solapamiento de clases [Chawla 2004], el ruido o la existencia de clases dispersas [López-Fernández 2013] entre otras. Estos problemas son de difícil solución a nivel de preprocesamiento de los datos. Aunque existen medidas para estimar la complejidad y por tanto conocer de antemano alguno de estas dificultades [Saez 2013], se requiere de una adaptación algorítmica que se adapte de manera dinámica durante el proceso de aprendizaje para mejorar la precisión en estos conjuntos de datos [Luengo 2011].

Una de estas características intrínsecas, la existencia de clases dispersas se define como la presencia de pequeños grupos de instancias de una clase rodeada por otra u otras clases. Estas instancias son de difícil clasificación y frecuentemente ignoradas por el algoritmo de aprendizaje como instancias ruidosas o poco representativas.

Capítulo 2

Modelo Distribuido para el aprendizaje de Reglas de Clasificación

Este Capítulo describe una propuesta de aprendizaje evolutivo distribuido de reglas de clasificación. En adelante nos referiremos a la implementación específica de este modelo como algoritmo genético distribuido eficiente para la extracción de reglas, EDGAR, acrónimo de su denominación en inglés: *Efficient Distributted Genetic Algorithm for Rule Extraction*.

El modelo propuesto pretende conseguir una sinergia positiva en el aprendizaje y cooperación de las diferentes partes del modelo:

- En cuanto a la distribución, utiliza un modelo de algoritmos genéticos paralelos de grano grueso [Cantú-Paz 2000] con una distribución de datos de entrenamiento parcial y una comunicación de reglas e instancias entre los nodos.
- El aprendizaje de las reglas se realiza mediante algoritmos genéticos con igual codificación en cada uno de los nodos. Denominamos a estos nodos AGL (Algoritmo Genético Local) por tratar con un conjunto local de datos asignados. Este algoritmo realiza el aprendizaje de reglas parciales las cuales son enviadas a un nodo central (supervisor).
- El clasificador global es generado por un nodo supervisor que recibe las mejores reglas, volviendo a efectuar la evaluación utilizando el conjunto

global de datos de entrenamiento hasta que éste alcanza la calidad necesaria para dar por terminada la búsqueda.

La descripción del modelo propuesto se organiza en las siguientes secciones:

- Modelo Evolutivo Distribuido, describe las partes del sistema y las técnicas de cooperación entre nodos para minimizar la posible pérdida de calidad del clasificador al realizar la distribución de datos entre los AGL.

- Diseño del algoritmo genético local, donde se justifican las decisiones de diseño del algoritmo genético utilizado para el aprendizaje de reglas y se revisan los componentes del mismo: el modelo de aprendizaje CCGL, el fenotipo , genotipo, función de evaluación , generación del clasificador y las adaptaciones realizadas a los operadores de cruce, mutación y selección para la codificación utilizada.

- Estudio Experimental, sección dedicada a desarrollar un conjunto de experimentos a fin de determinar la escalabilidad del modelo y su relación con la precisión de la solución generada. Se realiza asimismo una comparativa con conjuntos de datos de características variadas. Se ha considerado el uso de tests no paramétricos en la validación de los resultados obtenidos [Demšar 2006].

2.1 Modelo Evolutivo Distribuido

Como se ha indicado en el Capítulo anterior, los algoritmos de grano grueso pueden paralelizarse en base al paralelismo implícito de las poblaciones de individuos. A modo de resumen, la mayoría de los algoritmos genéticos distribuidos en el ámbito de la clasificación utilizan modelos centralizados que realizan frecuentes comunicaciones con los AGL [Giordana 1994] [Anglano 1998].

En este contexto se plantea mejorar la escalabilidad del modelo evolutivo propuesto en dos puntos del modelo distribuido:

- -Minimizar las esperas entre procesos

- -Reducir la cantidad de información que se transfiere entre procesos.

El modelo propuesto se basa en el modelo de islas en cuanto a la distribución de los AGL y la comunicación de individuos entre poblaciones. El susodicho modelo se complementa con técnicas de distribución de datos e integración de la información de los nodos como son: la distribución parcial del conjunto de datos, la comunicación de ejemplos de entrenamiento entre nodos y la integración global de las soluciones parciales por parte de un nodo supervisor. Todo esto será descrito en detalle en las siguientes secciones.

La organización del capítulo, por tanto, es la siguiente, donde cada subsección describe: la subsección 2.1.1, las funciones del supervisor en la integración de la información recibida; la subsección 2.1.2, la compartición de datos de entrenamiento; la subsección 2.1.3, la topología de comunicación de los AGL; la subsección 2.1.4, una técnica de reestructuración de la búsqueda; finalmente, la Sección 2.1.5, las estrategias de escalabilidad propuestas.

2.1.1 Supervisor

El modelo distribuido propuesto se apoya en una estrategia de búsqueda descentralizada sobre un conjunto parcial de los datos originales; para ello proponemos el fraccionamiento del conjunto de datos en particiones disjuntas en base a las instancias para aprender en cada una de ellas reglas de clasificación mediante un Algoritmo Genético (AGL).

Apoyándonos en la propiedad de invarianza (ver 1.1.1), especulamos con que alguno de los AGLs, encontrará una regla local que seguirá siendo aplicable en el conjunto completo de datos con una medida suficiente. No obstante, es necesario que el modelo cuente con una estrategia para integrar estas reglas en una solución global. El supervisor es la parte del sistema encargada de integrar la información proveniente de los nodos. Éste realiza las siguientes tareas.

- Integración de las reglas locales: se aplica la estrategia de integración [Fayyad 1996]. El supervisor recibe de cada AGL reglas válidas localmente y comprueba evalúa con el conjunto global de instancias. Este nodo conserva todas las reglas recibidas formando un conjunto

redundante sobre el que se genera un clasificador coherente globalmente. Este flujo se puede apreciar en la figura 2.1.

- Cálculo del criterio de parada global: genera periódicamente un clasificador partiendo del conjunto de reglas. La búsqueda finaliza cuando el clasificador generado no mejora durante un número de iteraciones predefinido.
- Generación del clasificador global: una vez terminada la búsqueda, el nodo supervisor genera un clasificador con un algoritmo voraz que tiene en cuenta las reglas con mayor número de casos positivos y bajo índice de error de clasificación (ver 2.2.8).

2.1.2 Compartición de Datos

La reducción de datos puede empeorar la labor de aprendizaje de las instancias pertenecientes a **clases dispersas** [Danyluk & Provost 93]. Éstas son también denominadas *small disjunts* [Quinlan 1991] [Weiss 1995], debido a que su representación en modelos de reglas corresponde a una disyunción de reglas. En este sentido el particionamiento de los datos en nodos ejerce el mismo efecto que una reducción de datos. Un *small disjunt* puede ser repartido en distintos AGLs, dificultando que una instancia local sea cubierta por una regla que defina el *small disjunt*.

Para tratar este problema EDGAR implementa una técnica original de compartición de datos que denominamos DLF (*Data Learning Flow)* la cual selecciona los datos de entrenamiento poco representados (con pocas reglas en su nodo que tengan a estos ejemplos como casos positivos) y los copia en otros nodos. La compartición de datos tiene como objetico reunificar los posibles ejemplos de un *small disjunt*. Por otro lado, los ejemplos que no lleguen a formar reglas de valor, formaran un contraejemplo efectivo y ayudarán a depurar las reglas presentes en dichas poblaciones.

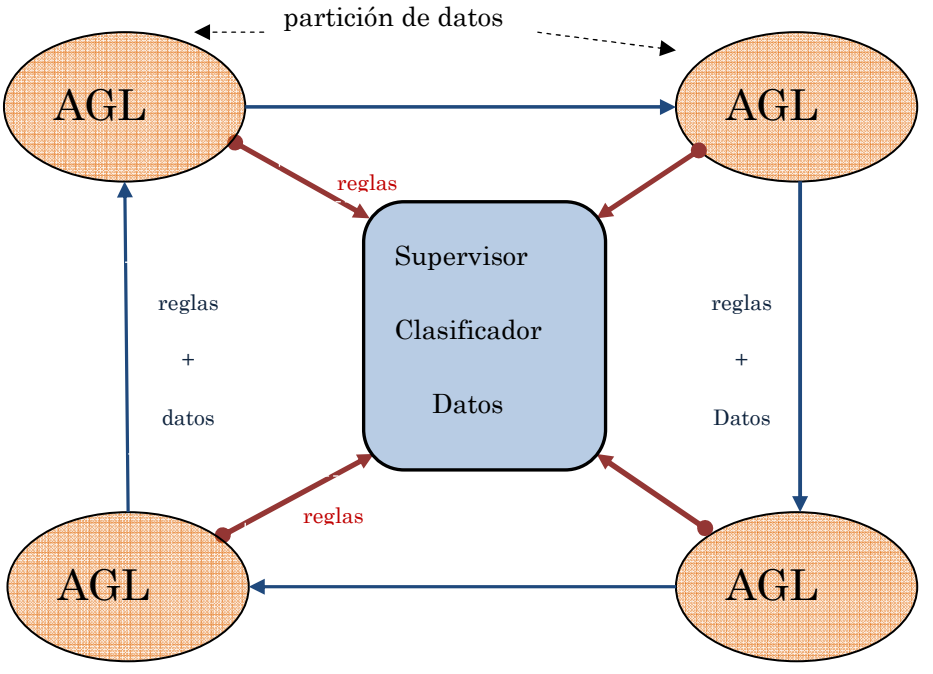

Figura 2.1 Esquema de distribución de datos e Individuos.

2.1.3 Topología

La topología del modelo distribuido establece como están conectados los AGLs ente sí para enviar y recibir instancias de entrenamiento y reglas de/hacia otros nodos. EDGAR implementa dos topologías dirigidas. De un lado, la topología en anillo, en la que cada nodo tiene un solo vecino del que recibe migrantes y otro al que los envía; y de otro lado, la topología en estrella, donde cualquier nodo es posible receptor/donante.

La topología en anillo permite un estudio del algoritmo con resultados más predecibles, adecuado para conjuntos de alta complejidad por su capacidad de mantener la diversidad. No obstante, cuando el número de nodos es grande (empíricamente a partir de 10/12 nodos), se puede producir una disminución de la

diversidad por el escaso refresco de nuevo material genético en las poblaciones más alejadas.

2.1.4 Reestructuración de la Búsqueda por Reducción de Datos

Cuando el algoritmo genético llega a un punto de equilibrio en la búsqueda en el que no se generan nuevas reglas, es posible que algunos ejemplos difíciles de cubrir sigan sin estar representados. Asumimos que las reglas de valor presentes en la evaluación no dejan evolucionar a la población y por tanto la eliminación de los datos que cubre redirigirá la búsqueda al resto de las instancias.

En base a esta hipótesis, proponemos una técnica de reducción de datos una vez alcanzado el criterio de estancamiento de la búsqueda en el AGL con dos objetivos:

- Focalizar la búsqueda en ejemplos aún no clasificados
- Generar nuevas reglas que tengan en cuenta el carácter de lista ordenada del clasificador.

Esta estrategia se parece a la búsqueda genética multimodal IRL [Augier 1995], sin embargo a diferencia de éste, eliminación de la regla de mayor valor (y sus datos) no implica reinicializar la población. Otra diferencia con este modelo es que en IRL el modelo converge a un solo individuo, mientras que el GA propuesto es un CCGL y por tanto las reglas existentes se re-evaluarán respecto a los ejemplos restantes. Este hecho produce de facto una reestructuración de la búsqueda para adaptarse a la distribución de los datos existentes. Esta reducción de datos continúa hasta que no queden elementos de entrenamiento.

2.1.5 Taxonomía de la Arquitectura Distribuida

Esta Sección enmarca este modelo distribuido en base a la taxonomía realizada en el Capítulo 1 sobre paralelismo:

- Paralelismo de datos: Se particiona un conjunto de datos en procesadores distintos, por lo que se alcanza escalabilidad respecto al volumen de datos a tratar.

- El modelo de computación está basado en arquitectura distribuida (*sharednothing*), donde cada procesador tiene memoria local suficiente para tratar el conjunto de datos asignado y sólo se comunican por paso de mensajes. Por consiguiente no será un algoritmo paralelo equivalente a su codificación compacta.

Respecto a la escalabilidad de la arquitectura, el tiempo de ejecución de un algoritmo depende del tiempo de cada uno de los procesadores (T_{pi}), el número de comunicaciones *c,* el tiempo medio de comunicación (T_c), Tiempo de espera para la sincronización (T_i) y la probabilidad de tiempo inactivo de un nodo (ρ) .

$$T = \sum_{1}^{n} T_{pi} + c*(T_c + T_i*\rho) \qquad (2.1)$$

Si el término $c*(T_c + T_i*\rho)$ es inferior en proporción al tiempo entre comunicaciones consecutivas del algoritmo genético en cada nodo, el algoritmo tendrá un comportamiento independiente de la velocidad de la red. Con el fin de minimizar este término se han seguido las siguientes estrategias:

- Reducción de las tareas del supervisor: la mayor parte de la comunicación se produce de forma descentralizada. El supervisor realiza una función de integración de menor costo computacional que no necesita sincronización con el aprendizaje realizado en los AGLs.

- Modelo de comunicación asíncrono: la comunicación se produce a través de estructuras de datos independientes, los nodos almacenan las reglas y los ejemplos poco representados en un buffer recogiendo de él los individuos/ejemplos provenientes de otros nodos. La figura 2.2 muestra éste esquema de comunicaciones. Los AGLs envían y reciben individuos en tiempos de evolución diferentes, ya que no están sincronizados entre sí. Esta circunstancia no tiene efectos negativos en el comportamiento del algoritmo dado que cada uno explora un modelo local de datos e integra periódicamente la información que recibe.

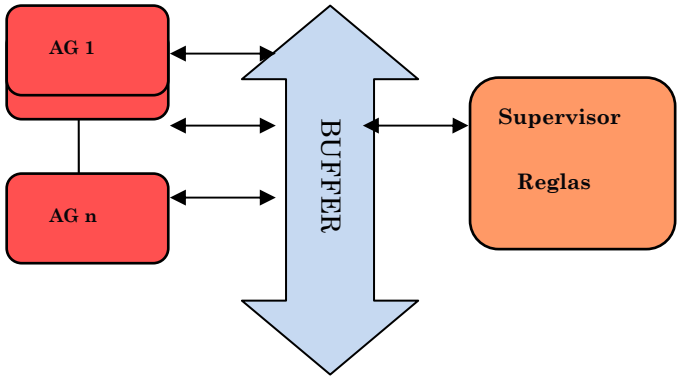

Figura 2.2 Arquitectura de comunicación no acoplada.

- Minimizar las comunicaciones: La información que se transfiere entre nodos o desde éstos al supervisor es relativamente pequeña. Cada cierto número de generaciones se envían las nuevas reglas y las instancias no representadas. El Tiempo de espera *(Ti)* y la probabilidad de estado inactivo (ρ) por cada procesador se puede minimizar, haciendo que el tiempo entre comunicaciones se adapte a las condiciones de la red.

2.2 Algoritmo Genético Local

En esta Sección se describe el algoritmo genético que en base al conjunto de datos recibidos, genera un conjunto de reglas. Se justifican la elección y la parametrización de los elementos característicos de un algoritmo genético: cromosoma, selección, cruce, mutación y generación de la población inicial.

El algoritmo cuenta con un conjunto de adaptaciones dependientes de la codificación *Un individuo = Una Regla*, entre ellas cabe destacar que la población final representa un conjunto redundante de reglas y que será necesario elegir el subconjunto que forme el clasificador final.

El funcionamiento general del algoritmo se muestra en el pseudocódigo 1. Se puede observar que corresponde con un sistema estacionario donde un porcentaje

de los cromosomas son reemplazados en cada iteración. El reemplazo se produce por *Crowding* determinístico. Este operador permite en combinación con el operador de selección una adecuada cobertura de los datos formando nichos.

```
Ejemplos = Ejemplos positivos y negativos de todas las clases
Población P = generar con Sembrado(Ejemplos)

Mientras (condición de parada)
   Cada M iteraciones
      Extraer clasificador
      Enviar/Recibir mejores individuos
      Enviar instancias poco representadas
   Seleccionar g individuos por Sufragio universal
      para cada individuo seleccionado:
        cruzar con otro individuo seleccionado
        mutar (probabilidad)
      fin para
      reemplazar g individuos de la población original
      Si condición de reducción
        Eliminar mejor regla y datos cubiertos
Fin Mientras
```

Pseudocódigo 1. AG para el aprendizaje de reglas

2.2.1 Fenotipo

La regla de clasificación elegida tiene una forma normal disyuntiva, en la que los antecedentes pueden tomar varios valores y el consecuente, representa la clase asignada a dicha regla.

Un *conjunto de datos* D , está compuesto por un número de *instancias* $I = i_1, i_2,..., i_n$, de *características* $C=c_1, c_2,..., c_j$, siendo la característica c_j la clase asignada a la instancia. Cada característica tiene valores dentro de un dominio que cumple

$c_i \in \{v_1 \cdots v_d\}$. D estará representado por un conjunto de reglas $R= r_1,r_2,....,r_k$ donde cada una de ellas puede tomar valores

$r_i : c_a (v_x,....,v_{ya}) \wedge ..... c_{j--1} (v_{xj-1},....,v_{yj-1}) \rightarrow c_j(v_i)$*(4.2)*

La regla r_i aplicada al conjunto D, se dice que cubre a una instancia I_j cuando para todas las características existentes en r_i el valor correspondiente de la característica en I_j está representado.

	Sexo	**Edad**	**Prueba1**	**Prueba2**	**Clase**
Regla	-	4,5,6	-	Negativo	Si
Ejemplo	V	5	Positivo	Negativo	Si

Figura 2.3 Ejemplo cubierto por regla

La regla de la figura 2.3, se lee como "todos los ejemplos en los que Edad es igual a 4 ó 5 ó 6 y Prueba2 es igual a Negativo", la misma expresión en forma compacta se puede expresar como *Edad(4,5,6)* $\wedge$ *Prueba(Negativo)* $\rightarrow$ *Clase(Si).*

La regla r_i aplicada al conjunto D, se dice *caso* positivo de dicha regla a una instancia I_j cuando para todas las características existentes en r_i el valor correspondiente de la característica en I_j está representado y la característica c_j es igual en I_i y en r_i..

Análogamente, la regla r_i aplicada al conjunto *D*, se dice caso negativo de dicha regla a una instancia I_j cuando para todas las características existentes en r_i, el valor correspondiente de la característica en I_j está representado y la característica c_j es distinta en I_i y en r_i..

El algoritmo de aprendizaje generará un conjunto R de reglas redundantes que cubren al conjunto de datos local D.

2.2.2 Genotipo

La regla se representa como una cadena binaria donde cada posible valor de un atributo está asociado a un bit. Esta representación puede tener varios valores activos en cada atributo, consiguiendo un lenguaje de descripción de conceptos

compacto. La clase también está representada en el cromosoma, con un único valor a la vez. Por lo tanto el clasificador generará reglas descriptivas para todas las clases presentes en el conjunto de datos.

Esta representación permite la evaluación de valores discretos y nominales. En el caso de atributos con valores continuos o discretos de alta cardinalidad, es necesario aplicar un preprocesamiento de discretización previo al conjunto de datos. Por ejemplo si una regla tiene 2 atributos, uno numérico y otro nominal, una vez discretizado el atributo numérico ésta queda como:

Atributo1=("**>=1y<2**" o **">5 y <5,**6") Y Atributo2 = (valor 1 o valor 7) → clase = 1.

Para el algoritmo los datos discretizados son a todos los efectos como los valores nominales no almacenando información de posición ordinal entre los distintos valores.

Tabla 2.1 Representación de reglas y ejemplos.

	Fenotipo	**Posición**	**Longitud**
Sexo {H, V}		1	2
Edad {1,2,3,4,5},		3	5
Prueba1 {Negativo, Positivo}		8	2
Prueba2 {Negativo, Positivo}		10	2
Clase[{Sí, No}		12	2

	Sexo	**Edad**	**Prueba1**	**Prueba2**	**Clase**
Regla	V	1,4,5	-	Negativo	Sí
	10	10011	01	10	10
Ej. Positivo	V	4	Positivo	Negativo	Si
	01	00010	01	10	10
Ej. Negativo	V	1	Positivo	Negativo	No
	10	10000	01	10	01
Ej. No Cubierto	H	5	Positivo	Negativo	Si
	01	00001	01	10	01

2.2.3 Operador de Cruce

Los operadores de cruce determinan en parte la velocidad de convergencia del algoritmo. Debido a este motivo se han elegido dos operadores de cruce con distintas características de exploración y explotación de la solución. Por un lado se ha elegido el cruce en dos puntos, más enfocado a la explotación de individuos de alto valor. Por otro lado, se ha elegido el cruce uniforme con una máscara de generación aleatoria, que permite una mayor exploración en los individuos poco evolucionados.

	Sexo	**Edad**	**Prueba1**	**Prueba2**	**Clase**
Regla 1	V	1,4,5	-	Negativo	Sí
	10	10011	00	00	01
Regla 2	H	4	Positivo	-	No
	01	01010	01	00	10
Cruce					
Resultado 1	V	2,4	Positivo	Negativo	Si
	10	10010	01	10	01
Resultado 2	H	2,4,5			No
	01	01011	00	00	10

Figura 2.4 Cruce en dos puntos.

Ambos operadores se aplican como en los algoritmos genéticos canónicos a nivel de bit. Después de la realización del cruce se comprueba que los hijos sean válidos. La única posibilidad de que esta circunstancia ocurra es la generación de un individuo con todos sus posiciones a 0 o a 1 (‘00000....0000’ o ‘111111111111’) en el antecedente ya que nunca evaluaría un individuo. La figura 2.4 muestra el efecto del operador de cruce sobre dos reglas en su fenotipo y su genotipo. La parte sombreada de cada regla genera el resultado 1 en los puntos de corte a nivel de bit y la parte no sombreada genera el resultado 2.

La capacidad de exploración del operador uniforme es mayor en el cruce en dos puntos, pero puede producir descendientes de bajo valor cuando los padres son individuos de alto valor. Por otro lado el operador de cruce por si sólo puede ser

muy lento en convergencia [Spears 1995]. Proponemos combinar la capacidad de exploración del operador de cruce uniforme, con la capacidad de explotación del cruce en dos puntos, mediante una función que determine en base a la calidad de los individuos, el tipo de cruce más indicado. En este sentido otros autores han usado técnicas similares para adaptar el tipo de cruce. Por ejemplo , [Spears 1995] codifica el tipo de cruce a utilizar en el propio cromosoma. De esta forma es el propio cromosoma el que determina cuales son los cruces más adecuados para un individuo. Por otro lado GABIL [DeJong 1993] usa un método basado en estimación de la calidad de los individuos similar al utilizado en este trabajo. Asimismo REGAL [Giordana 1994] también utiliza un equilibrio entre estos dos operadores para mantener la calidad de los descendientes.

Experimentalmente se ha observado que el porcentaje p de aplicación de los cruces se comporta adecuadamente para la formula

$$p = \frac{F.Evaluación(Padre1) + F.Evaluación(Padre2)}{2} \qquad (2.2)$$

Para un valor de p superior a la media del valor de la función de evaluación en la población actual se considera que la calidad de los padres es suficiente para evitar una posible disrupción de los cromosomas y por tanto se utiliza el operador de cruce en dos puntos. En otro caso se considera que es más productivo explorar mediante el uso del cruce uniforme.

2.2.4 Operador de Mutación

El operador de mutación es el habitual sobre codificación binaria, es decir el cambio de *uno* a *cero* o viceversa en un bit elegido aleatoriamente según un ratio de mutación preestablecido. Con todo se ha modificado el comportamiento del operador , favoreciendo la generalización o la especialización de la regla, según la calidad del individuo.

En [Srinivas 1994] se analizan varias opciones de variación dinámica de la mutación. Entre ellas se muestra como una opción efectiva la disminución del ratio de mutación a medida que la población evoluciona a soluciones de calidad. En el ámbito de la inducción de reglas, otros autores [De Jong, 1993] [Greene 1993] [Janikow 1993] utilizan operadores de mutación orientados a tareas especificas como añadir o eliminar condiciones.

En EDGAR, el criterio seguido para determinar el tipo de mutación a aplicar se basa en el valor de la función de evaluación (2.3). En el caso de reglas con un valor superior a la media se favorece (un 80% de las veces) la generalización (mutación a *cero*) para aumentar la cobertura de la regla. En caso contrario se favorece la especialización (mutación a *uno*) para reducir el número de casos cubiertos e intentar reducir por tanto los casos negativos.

2.2.5 Función de Evaluación

La función de evaluación en los algoritmos de aprendizaje está basada en los conceptos de

- Completitud: el clasificador debe cubrir la mayor parte posible del conjunto de datos.
- Precisión: debe clasificar positivamente las instancias a las que representan las reglas. A nivel de la regla significará maximizar el número de casos positivos y minimizar el de casos negativos que cubre.

- Comprensión: el conjunto de reglas debe ser comprensible. Hay muchas medidas posibles para ello. En general se acepta que el tamaño del lenguaje de descripción es una medida válida.

La función propiamente dicha se centra en la comprensión y precisión a través de los conceptos de equivalentes de consistencia y simplicidad:

- La consistencia se mide en base al número de casos negativos (c-).Los casos negativos serán todos aquellos cubiertos por la regla cuya clase es diferente a la predicha por la misma.
- La simplicidad de una regla es el número de antecedentes de la misma. Dado que la regla se codifica indicando en un cromosoma binario la existencia de un antecedente esta medida corresponde al número de *unos* presentes en el mismo.

La función de evaluación utiliza una función exponencial para minimizar el número de casos negativos de la clase. Una vez encontrada una regla de alta consistencia, el algoritmo optimizará la simplicidad, eliminando antecedentes no necesarios.

$$f(r) = \left(1 + \frac{ceros(r)}{longitud(r)}\right)^{-1*casos^-} \qquad (2.3)$$

No obstante el algoritmo debe cumplir el criterio de completitud y comprensión. Para ello se define *PI* como un valor derivado de la función de evaluación que mide la precisión, la comprensión y la cobertura de la regla (2.4).

$$PI = Casos^+ * F.Evaluación \qquad (2.4)$$

PI se tiene en cuenta en varios puntos del proceso de aprendizaje:

- Operador de Selección: pondera una ruleta de selección en base a *PI* para favorecer aquellas reglas con mayor capacidad de cobertura.
- Generación del clasificador: ordena las reglas en orden de *PI* reduciendo el número de reglas necesarias para cubrir todos los datos.

2.2.6 Operador de Sembrado y Población Inicial

Este operador desarrollado por [Giordana 1994] genera una regla válida que cubre un ejemplo. Este operador se usa en combinación con el operador de selección por votado universal *US o* en la generación de la población inicial.

Ejemplo	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**
Regla sembrada	**1**	0	0	**0**	**1**	0	**1**	**1**	**1**	0	0	**0**	**1**

Figura 2.5 Operador de Sembrado

Respecto del operador de selección, éste se aplica cuando un ejemplo no encuentra un representante en la población actual, motivo por el que se genera un individuo utilizando el ejemplo como patrón y generalizando alguna de sus características. La figura 2.5 muestra una regla sembrada de un hipotético ejemplo.

El operador de votado universal seleccionará un ejemplo con una probabilidad de 1/nº de instancias locales. Un ejemplo no representado tendrá al menos una regla activa cada:

$$Freq: \frac{\#n * iteraciones}{\%individuos\ por\ selección} \tag{2.5}$$

En cuanto a la generación de la población inicial, ésta es creada mediante la selección aleatoria de *n* instancias a las que se les aplica el operador de sembrado. El operador de sembrado establece una relación directa entre el número de ejemplos presentes y la posibilidad de que una regla cubra dicho ejemplo en la población inicial.

2.2.7 Operador de Selección

La estrategia GCCL requiere que los individuos compartan el espacio de búsqueda mediante la formación de nichos o especies. EDGAR mantiene la diversidad de soluciones en el AGL mediante el operador de Sufragio Universal (*US*), [Giordana 1994].

US es un operador de selección especializado para el aprendizaje de reglas con codificación CCGL. .En cada proceso de selección se elige un porcentaje de ejemplos**.** Cada uno de éstos se utiliza para seleccionar de la población actual los individuos que lo representan. Para mejorar la diversidad se introduce una selección por ruleta entre todos los individuos que cubren al ejemplo, teniendo más opciones de ser elegido aquel que tiene mayor valor de *PI (2.4)*.

El sufragio universal no está basado en medidas de similitud, pero tiene un carácter local que permite la formación de especies. Por otra parte permite el

mantenimiento de individuos poco representados por la población con el operador de sembrado, incluyendo nuevas reglas cuando un ejemplo no tiene ningún representante activo. La figura 2.6 muestra el proceso de votado y selección por ruleta para la elección de los padres.

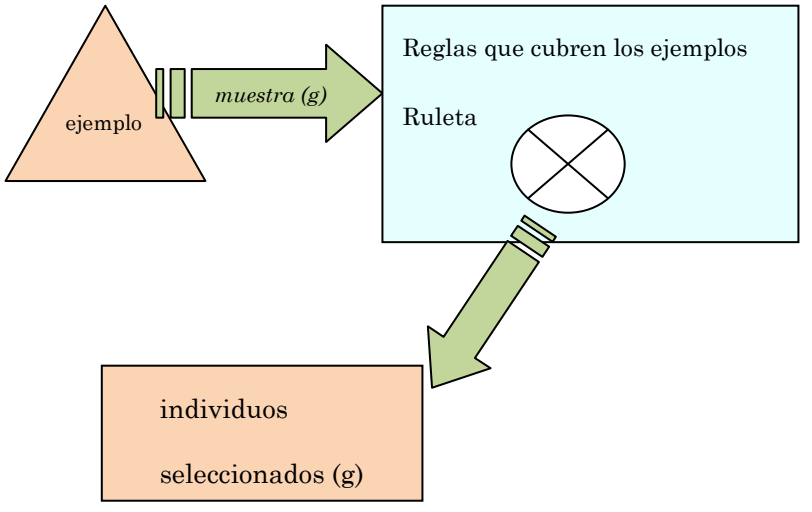

Figura 2.6 Operador de Sufragio Universal.

2.2.8 Generación del Clasificador

La población final del algoritmo está compuesta por un conjunto redundante de reglas que representan conceptos generalizados del conjunto de datos. Dada una población *M*, podemos extraer de este conjunto de individuos un clasificador no redundante, optimizando la longitud y la precisión con una cardinalidad en reglas menor o igual a *M*.

La selección de las reglas que se envían a otros nodos y el supervisor es la solución o *clasificador*. El clasificador es construido por un algoritmo voraz en base al conjunto redundante de reglas de la población actual. Éste algoritmo utiliza como

criterio de ordenación PI, basado en el número de casos positivos y el valor de la función de evaluación de la regla (2.4). De esta forma el algoritmo voraz minimiza el número de reglas necesarias para cubrir todos los ejemplos y elige entre las reglas las de mejor calidad y menor longitud.

```
Mientras (haya ejemplos)
   Seleccionar la primera regla
   para cada individuo seleccionado:
        Añadir la regla al clasificador
        Eliminar casos cubiertos positivos
        Actualizar el valor de las reglas
   fin para
fin Mientras
```

Pseudocódigo 2. Algoritmo voraz de generación de clasificador

El algoritmo voraz extrae la regla de mayor *PI* y elimina todos los ejemplos cubiertos positivamente por ella. Este proceso se repite hasta que no queden ejemplos a cubrir. Este proceso se muestra en el pseudocódigo 2.

```
   Ordenar reglas por Fitness*Ejemplos positivos
   Si todos los ejemplos están cubiertos →
      podado de las últimas reglas con igual clase c
      añadir regla ELSE → class = c
   Si no
      c = clase con mayor numero de ejemplos sin cubrir
   añadir regla ELSE → class = c
```

Pseudocódigo 3. Determinación de la clase por defecto

El algoritmo de extracción añade una regla por defecto al final del clasificador para asegurar la cobertura de todos los casos. Esta estrategia ha demostrado

reducir considerablemente el número de reglas necesarias [Bacardit 2007]. La regla por defecto se calcula según el pseudocódigo 3 inspirado en *C4.5Rules* [Quinlan 1993].

0-(gill-size=(b)) ∧ (ring-type=(e,l,p)) ∧ (spore-print-color=(k,n,o,w))-->(class=e)

1-(gill-size=(b)) ∧ (stalk-below-ring=(b,e,p,w)) ∧ (spore-print-color=(k,n,o,u,w,y))-->(class=e)

2-(odor=(a,l,n)) ∧ (spore-print-color=(k,n,o,u))-->(class=e)

3-(gill-spacing=(c,d)) ∧ (stalk-surface-above-ring=(f,k))-->(class=p)

4-(odor=(c,f,p)) ∧ (ring-type=(l,p))-->(class=p)

5-(odor=(c,f,m,p,s,y)) ∧ (stalk-below-ring=(b,g,n,p))-->(class=p)

6-(odor=(l,n))□(stalk-color-below-ring=(g,n,w)) ∧ (spore-print-color=(h,n,o,u))-->(class=e)

7-(odor=(c,m,p,s,y))-->(class=p)

8-(odor=(c,f,m,p,s,y)) ∧ (stalk-surface-below-ring=(s,y))-->(class=p)

9-(odor=(l,n)) ∧ (stalk-color-below-ring=(o,w))(spore-print-color=(b,h,n,o,u))-->(class=e)

10-(odor=(c,f,m,p,s,y))-->(class=p)

11-(ring-number=(t)) ∧ (spore-print-color=(h,n,r))-->(class=p)

12-(stalk-below-ring=(e,g,n)) ∧ (habitat=(l))-->(class=e)

13 –ELSE class=p

Figura 2.7 Clasificador generado para *Mushroom*

La figura 2.7 muestra un ejemplo de clasificador generado para el conjunto de datos *Mushroom*. Se puede observar el uso de reglas que combinan el operador disyuntivo dentro del atributo y conjuntivo entre éstos. Asimismo se puede observar la existencia de la regla por defecto al final del clasificador.

2.3 Estudio Experimental

En esta Sección se realiza un estudio experimental del modelo evolutivo en eficiencia y calidad, sobre un conjunto de datos con características variadas para observar el comportamiento del mismo.

Esta Sección se compone de 5 subsecciones, en las que se describirá respectivamente la metodología de comparación de tiempos, los algoritmos utilizados a efectos de comparación, los conjuntos de datos empleados en la experimentación, los tests estadísticos empleados en el estudio experimental y finalmente la comparativa en precisión y escalabilidad.

2.3.1 Metodología de Comparación de Tiempos

Para este estudio se han tenido en cuenta las siguientes consideraciones:

Arquitectura de experimentación: ésta se ha realizado sobre un clúster de 8 estaciones con 2 CPUs Intel Xeon de 3GHz cada una. Se ha desarrollado un modelo sobre una maquina virtual de Java para poder realizar una estimación del comportamiento en una maquina paralela distribuida (*Grid)* sobre una red con más nodos. Este modelo permite simular las diferentes estructuras de nodos en un número menor de procesadores. La máquina virtual de *java* garantiza equidad en la gestión de los procesos (hilos) con independencia del sistema operativo y el tipo de procesador. La comunicación entre diferentes procesos se realiza a través de objetos dedicados que simulan el comportamiento de la red.

Metodología de comparación: el tiempo total (T) del algoritmo distribuido se puede expresar como:

$$T = \sum_{1}^{n} T_{pi} + c*(T_c + T_i * \rho) \qquad (2.6)$$

Donde T_{pi} es el tiempo dedicado a computación por cada uno de los nodos, c es el número de comunicaciones realizadas, contabilizando una comunicación cada vez que una regla o ejemplo es enviado a otro nodo. T_c es el tiempo de espera medio necesario para enviar un ejemplo/regla a otro nodo. $T_{i\,\text{es}}$ el tiempo medio de espera (*idle time)* de un nodo y ρ el porcentaje de veces que se produce la espera.

Se ha calculado T_c (Tiempo de comunicación) en (2.6) como el tiempo necesario para la transmisión de una regla en una red de 10mbs. Los parámetros de red se han tomado como valores medios en la experimentación con el clúster real. Los valores asignados son de 512 bytes por paquete transmitido bajo protocolo Ethernet de 10mbs de ancho de banda. La longitud media de regla/instancias es de 30 bytes de longitud y el número de reglas/instancias medio transmitido es de 25. Esto supone un retraso de 0,0512/25 = 0,00015 segundos por regla.

2.3.2 Algoritmos Comparados

En esta Sección vamos a comparar EDGAR en calidad (número de reglas y precisión) con REGAL [Giordana 1994] como principal representante del ámbito de algoritmo genético distribuido para la clasificación. Este algoritmo se va a comparar en términos de escalabilidad y su relación con la pérdida de calidad de la solución.

REGAL es un algoritmo distribuido que comparte con EDGAR algunas características. Ambos son algoritmos distribuidos cuyos nodos contienen un algoritmo genético de aprendizaje, cromosoma binario y operador de selección US. De la misma forma comparte con EDGAR la filosofía de implementación CCGL.

2.3.3 Conjuntos de Datos

La legitimación de un proceso de experimentación requiere que los casos de prueba, en esta ocasión los conjuntos de datos, representen un margen amplio de posibilidades en varias categorías. Para la experimentación se han seleccionado 26 conjuntos de datos de UCI [Merz 1996]. El número de conjuntos de datos viene impuesto por la necesidad de establecer un mínimo de confianza en los tests no paramétricos [Demšar 2006]. Para los algoritmos de clasificación de carácter discreto se ha usado el método de discretización de igual frecuencia con 10 intervalos por atributo. En este estudio se ha utilizado este método para evaluar el comportamiento en tiempos de red del algoritmo con un número conocido de intervalos, ya que éste determina la longitud de la regla, y por tanto, el tamaño medio de los ejemplos y reglas. La tabla 2.2 muestra en la primera columna el nombre del conjunto de datos, en la segunda la cardinalidad, en la tercera el número de atributos y finalmente en la cuarta el número de clases.

Tabla 2.2 Datasets experimentación EDGAR

C. Datos	Instancias	Atributos	Clases
Car	1727	6	4
Cleveland	297	13	5
Credit	336	7	2
Ecoli	336	7	2
Glass	214	9	7
Haberman	305	3	2
House Votes	432	16	2
Hypothyroid	1920	9	4
Iris	150	4	3
krvskp	3198	37	2
Monks	432	6	2
Mushrooms	8124	22	2
New-Thyroid	215	5	3
Nursery	12960	6	2
Pima	768	8	2
Segment	2308	19	7
Soybean	307	35	19
Splice	3190	60	3
Tic-tac-toe	958	9	2
Vehicle	846	18	4
Waveform	5000	41	3
Wine	178	13	3
Wisconsin	683	9	2
Vote	435	16	2
Thyroid	7200	21	3
Zoo	100	16	7

Respecto a las características de los datasets elegidos se han seguido los siguientes criterios:

Complejidad: La muestra de conjuntos de datos contiene elementos con alto número de atributos y varias clases. Entre ellos cabe destacar Segment (19 atributos y 7 clases), Soybean (35 atributos y 19 clases) y Splice (60 atributos y 3 clases). La figura 2.11 muestra la distribución de atributos por conjunto de datos.

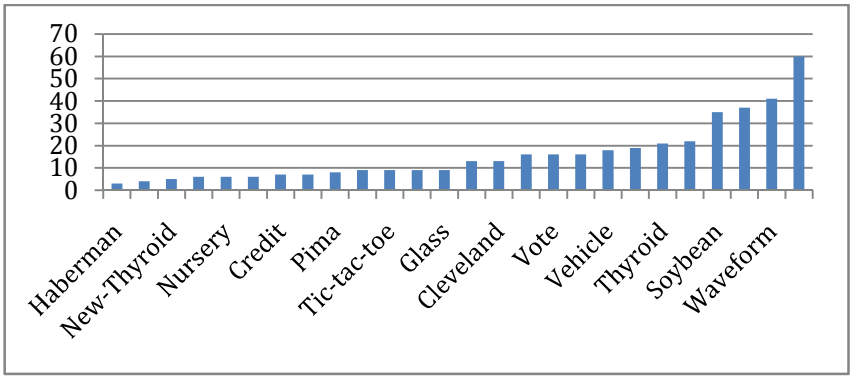

Figura 2.11 Distribución de atributos por conjunto de datos

Cardinalidad: se han seleccionado conjuntos de datos con baja cardinalidad como *Zoo* con 100 instancias o *Glass* con 214 instancias. Aunque no son el objetivo para una experimentación basada en escalabilidad en este tipo de algoritmos, muestran los límites de uso del algoritmo. Por otro lado para comprobar el comportamiento en alta cardinalidad se han elegido conjuntos de datos como *Nursery* con 12.960 instancias o *Mushroom* con 8.124 instancias. La figura 2.12 muestra la distribución de número de instancias por conjunto de datos.

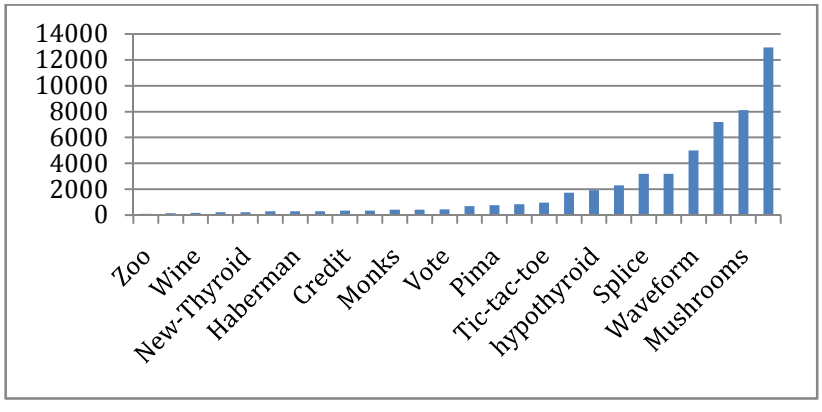

Figura 2.12 Distribución de número de instancias por conjunto de datos

Número de clases: se han incluido algunos conjuntos de datos que cumplen varios criterios de dificultad de aprendizaje como *Nursery,* con 6 clases y alta

cardinalidad o *Soybean* con 35 atributos 19 clases. La figura 2.13 muestra la distribución de clases por conjunto de datos.

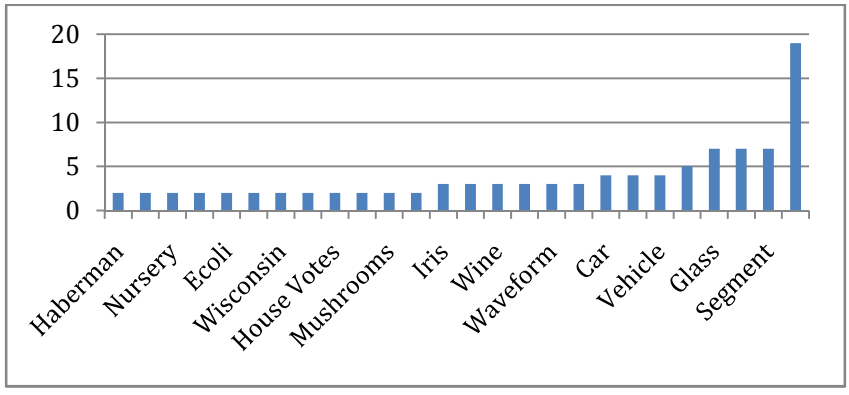

Figura 2.13 Distribución de clases por conjunto de datos

2.3.4 Análisis Estadístico

El uso de técnicas estadísticas paramétricas es adecuado sólo cuando se cumplen tres condiciones necesarias en los datos: independencias, normalidad y homocedasticidad [Demšar 2006]. En la experimentación realizada estas condiciones no se cumplen con los conjuntos de datos y algoritmos utilizados, por tanto se ha optado por utilizar tests no paramétricos siguiendo las recomendaciones realizadas por [García 2008] [García 2009]. Los tests no paramétricos han demostrado ser más seguros que los test paramétricos al no asumir una distribución normal o una homogeneidad de la varianza. Éstos pueden ser aplicados a la precisión en la clasificación, ratios de error u otra medida de evaluación en los clasificadores. Los resultados empíricos sugieren que son también más robustos [García 2009]. Demšar [Demšar 2006] recomienda un conjunto de tests simples y seguros para la comparación estadística de clasificadores, entre los que se encuentra *Wilcoxon signed-Rank test*. Éste es análogo a *t-test* en la estadística no paramétrica, permitiendo detectar diferencias significativas en el comportamiento de dos algoritmos. En nuestro estudio

mostramos el nivel de significación (p) al que se puede rechazar la hipótesis de igualdad (Ho). Un valor p de 0.05 significa que Ho puede ser rechazada con un nivel de confianza del 95%. Este test determina un mínimo de 24 conjuntos de datos. En este estudio se han utilizado 26 para tener un margen de confianza suficiente.

En el apéndice A se encuentra una explicación más detallada de la aplicación de los tests no paramétricos.

2.3.5 Comparación con Algoritmo de Referencia

Esta Sección está dedicada a analizar el efecto de la distribución de datos sobre la precisión y tamaño de la solución producida. Asimismo se detalla el tiempo necesario para construir la solución. Para analizar estas medidas se ha realizado una experimentación con distintas configuraciones de nodos siguiendo una serie exponencial desde 4 a 64 nodos.

Los parámetros de los algoritmos se han mantenido constantes en las diferentes configuraciones de nodos. Entre ellos hay que destacar la población de individuos. Se ha establecido una población global (la del conjunto de todos los nodos) de 1600 individuos, lo que corresponderá a diferentes poblaciones por nodo según la configuración, esto es, cada nodo tendrá una población igual a la población total entre el número de nodos. Todos los parámetros de REGAL han sido extraídos de la experimentación en su artículo original. La tabla 2.3 muestra los parámetros de ejecución de ambos algoritmos.

Tabla 2.3 Parámetros experimentación escalabilidad

	REGAL	***EDGAR***
Criterio de parada	*500 gen.*	*epo=20*
Porcentaje de Mutación	0,01%	1%
Porcentaje de Cruce	60%	90%
Porcentaje de selección	10%	10%
Ratio de Comunicación	10%	10% max
Reducción dataset	--	LSP =5
Numero de Reglas no representados	---	2

Se han extraído de esta experimentación los datos de dos conjuntos de datos representativos (Tablas 2.4 a 2.7) y un resumen de precisión para todos los datasets (Tabla 2.9). Para hacer el análisis más legible se ha llevado la experimentación detallada por nodos al apéndice B al final del documento

Las tablas 2.4 a 2.7 muestran las medias en 150 ejecuciones (5 para cada partición, con 6 semillas y configuraciones de nodos para 4, 8, 16, 32 y 64 nodos) para Mushroom y Nursery. Por un lado, Mushroom es un conjunto de datos de fácil aprendizaje, con dos clases, y suficientemente grande para medir el efecto de la distribución en diferentes configuraciones de nodos sobre la calidad del clasificador (precisión y número de reglas). Por otro lado, Nursery es un conjunto de datos de tamaño similar y mayor complejidad. Se han evaluado los dos algoritmos usando particiones 5x2 [Dietterich 1999] con 50% y 5 semillas.

La primera columna expresa el número de nodos, la segunda el número de comunicaciones de un nodo (una comunicación es un ejemplo o una regla enviada). La tercera columna contiene el tiempo de ejecución en minutos. La cuarta columna muestra el número de reglas del clasificador generado y las columnas quinta y sexta la precisión de test y entrenamiento respectivamente. La figura 2.14 muestra la evolución del tiempo de ejecución para el conjunto de datos Nursery en ambos algoritmos.

Podemos destacar algunos datos sobre las tablas 2.4 a 2.7. Respecto al tiempo de ejecución, observamos un considerable aumento de la escalabilidad (*speed up*) y un mejor comportamiento que el algoritmo comparado cuando el número de nodos aumenta. En Mushroom la precisión no es afectada por la distribución de los datos, sin embargo Nursery muestra una ligera tendencia de pérdida de precisión para valores altos de número de nodos en ambos algoritmos.

En la figura 2.14 se aprecia un mejor comportamiento de REGAL para la configuración de 4 nodos. Por otro lado la experimentación muestra que hay un punto de equilibrio dependiente de cada conjunto de datos en el que EDGAR mejora a REGAL en cuanto a tiempo de ejecución. La precisión del clasificador es similar en ambos algoritmos y no muestra ninguna tendencia en términos del número de procesadores.

Tabla 2.4 Resultados ejecución Mushroom - REGAL

Nodos	*#Com.*	*Tiempo*	*Reglas*	*%Test.*	*%Tra.*
4	318.092	0,79	16	99,93	99,98
8	558.501	0,68	14	99,95	99,99
16	681.782	0,57	15	99,95	99,99
32	1.085.929	0,54	15	99,94	99,99
64	1.267.774	0,43	16	99,95	99,99

Tabla 2.5 Resultados ejecución Mushroom – EDGAR

Nodos	*#Com.*	*Tiempo*	*Reglas*	*%Test*	*%Tra.*
4	2.129	1,2	14	99,92	99,94
8	3.907	0,55	13	99,94	99,96
16	8.232	0,35	15	99,96	99,99
32	17.160	0,23	16	99,95	99,9
64	20.211	0,18	16	99,93	99,95

Tabla 2.6 Resultados ejecución Nursery - REGAL

Nodos	*#Com.*	*Tiempo*	*Reglas*	*%Test*	*%Tra.*
4	784.870	1,78	290	98,6	98,9
8	2.319.559	1,97	251	99	99,3
16	6.304.130	2,32	250	98,9	99,2
32	18.595.490	2,81	268	98,5	98,7
64	50.664.658	3,08	316	97,9	98,1

Tabla 2.7 Resultados ejecución Nursery – EDGAR

Nodos	*#Com.*	*Tiempo*	*Reglas*	*%Test*	*%Tra.*
4	6.818	2,89	173	99,4	99,7
8	3.356	1,59	209	98,5	98,8
16	4.836	1,2	206	98,9	99,2
32	8.309	1,11	231	98,3	98,8
64	77.859	1,21	199	98,5	98,6

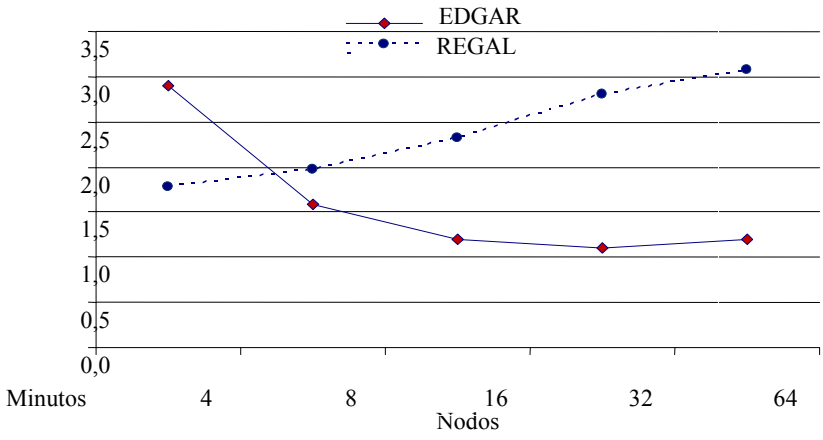

Figura 2.14 Evolución de tiempo de ejecución para Nursery

Respecto del comportamiento general del número de nodos en la calidad de la solución, la Tabla B.1 (anexo B) muestra el detalle de las ejecuciones por nodo y conjunto de datos para los dos algoritmos comparados. La tabla 2.8 muestra el test no paramétrico Wilcoxon Signed-Rank para las diferentes configuraciones de nodos en número de reglas y precisión. La tabla 2.9 muestra los resultados medios agregados de todas las ejecuciones para ambos algoritmos. Respectivamente para cada algoritmo la primera columna expresa el número de reglas, la segunda la precisión en test, la tercera la precisión en pruebas y por último el tiempo en minutos.

Podemos observar varios hechos basándonos en el contenido de las tablas 2.8 y 2.9 respecto de la influencia de la distribución de nodos sobre la calidad de la solución:

- El número de nodos en EDGAR no sigue ninguna tendencia respecto de la variación de la precisión.
- El tiempo de proceso disminuye con el número de nodos aunque no alcanza un *speed up* lineal.

- En la tabla 2.11 se aprecia que de media EDGAR supera a REGAL en 20 de los 25 datasets en precisión con una confianza del 99%.
- Respecto a los tiempos Edgar consigue una mejora estadísticamente significativa en todas las configuraciones.
- No se puede rechazar la hipótesis de igualdad para el número de reglas debido a que la confianza es de solo un 27% para el caso medio y menos de un 90% en el resto de las configuraciones de nodos.

Tabla 2.8 Resultados test Wilcoxon escalabilidad

Nodos	REGAL / EDGAR			R+	R-	Emp.	p-value
4	Rules	<	Rules	16	6	3	0,13
4	Acc.	<	Acc.	17	8	0	0,11
4	Time	<	Time	0	24	1	0,00
8	Rules	<	Rules	12	9	4	0,49
8	Acc.	<	Acc.	17	8	0	0,31
8	Time	<	Time	1	24	0	0,00
16	Rules	<	Rules	12	10	3	0,40
16	Acc.	<	Acc.	18	7	0	0,09
16	Time	<	Time	0	25	0	0,00
32	Rules	<	Rules	10	12	3	0,21
32	Acc.	<	Acc.	17	8	0	0,32
32	Time	<	Time	0	25	0	0,00
All	Rules	<	Rules	12	10	3	0,73
All	Acc.	<	Acc.	20	4	1	0,01
All	Time	<	Time	0	24	1	0,00

Tabla 2.9 Resumen de resultados comparados escalabilidad

	EDGAR				REGAL			
	Reglas	**Test**	**Train**	**Tiempo**	**Reglas**	**Test**	**Train**	**Tiempo**
Car	61	0.974	0.941	0.10	66	0.978	0.936	63.4
Cleveland	73	0.914	0.543	0.07	62	0.959	0.833	100.5
Credit	69	0.902	0.836	0.05	75	0.941	0.564	69.4
Ecoli	61	0.904	0.642	0.04	47	0.958	0.729	95.5
Glass	61	0.923	0.458	0.07	45	0.963	0.548	106.2
Haberman	29	0.816	0.687	0.04	52	0.921	0.694	125.4
House-votes	26	0.977	0.956	0.02	26	0.970	0.948	0.7
Hypothyroid	200	0.954	0.938	1.00	178	0.953	0.927	35.6
Iris	14	0.957	0.915	0.01	11	0.987	0.906	36.2
Krvskp	62	0.977	0.965	0.10	54	0.861	0.850	22.5
Monk	46	0.987	0.780	0.03	58	0.766	0.634	61.7
Mushroom	13	1.000	1.000	0.14	16	1.000	1.000	4.9
New-thyroid	14	0.993	0.944	0.81	14	0.992	0.914	51.9
Nursery	270	0.985	0.976	1.63	309	0.983	0.968	91.0
Pima	128	0.943	0.730	0.19	127	0.943	0.730	65.5
Segment	132	0.985	0.933	0.47	137	0.986	0.918	30.5
Soybean	72	0.981	0.942	0.41	81	0.973	0.914	21.8
Splice	77	0.999	0.975	9.71	72	0.946	0.968	29.1
Tic-tac-toe	87	0.992	0.943	0.06	50	0.911	0.835	61.4
Vehicle	181	0.953	0.620	0.64	161	0.949	0.610	39.5
Vote	24	0.972	0.946	0.02	8	0.530	0.508	11.8
Waveform	1026	0.948	0.720	15.15	1070	0.934	0.709	15.1
Wine	57	0.972	0.472	0.03	60	0.970	0.466	1.2
Wisconsin	25	0.976	0.949	0.02	25	0.995	0.936	14.7
Zoo	9	0.690	0.570	0.02	6	0.652	0.529	9.1

Como conclusión, incidir en que EDGAR muestra un *speed up considerable* y que aunque la distribución de los datos puede afectar al porcentaje de precisión, ésta no necesariamente empeora con la cardinalidad del modelo distribuido. Asimismo supera en precisión a REGAL, una de las referencias más representativas en el ámbito de los algoritmos genéticos distribuidos para la clasificación.

Capítulo 3

Modelo Distribuido de Aprendizaje para Clases No Balanceadas

Recientemente muchos estudios se han dedicado a este área de interés desarrollando nuevas estrategias [Hong 2007] [Chen 2007] [Lee 2008] [Su 2006]. En este sentido, este trabajo propone técnicas originales en el tratamiento de las clases no balanceadas.

Este capítulo muestra las adaptaciones realizadas al modelo expuesto en el capitulo anterior para mejorar la calidad de la clasificación con clases no balanceadas. Aunque es factible realizar un preprocesamiento para el tratamiento de las clases no balanceadas, este conlleva normalmente un aumento de la cardinalidad del conjunto de datos y podría producirse una pérdida de calidad tras el preprocesamiento por lo que resulta interesante la alternativa de tratamiento algorítmico de las clases no balanceadas.

El modelo de distribución propuesto en el Capítulo anterior (EDGAR), no tiene en cuenta la desproporción de ejemplos entre clases, motivo por el cual, en los nodos la proporción de desbalanceo se mantendrá en la misma proporción que el conjunto original, con todo el numero de instancias de la clase minoritaria será menor y puede llevar a un fraccionamiento de los conceptos de esta clase. Aunque el modelo propuesto dispone de una técnica de compartición de datos que mejora la distribución de los ejemplos poco representados, esta no está diseñada para rebalancear las clases y por consiguiente es necesario desarrollar otras técnicas que permitan utilizar ésta como una ventaja para el tratamiento de problemas de clasificación con clases no balanceadas [Fernández 2011]

3.1 Introducción

Un conjunto de datos es balanceado si tiene, aproximadamente, igual porcentaje de ejemplos positivos del concepto a clasificar como negativos. Se considera no balanceado cuando el ratio entre las instancias de las dos clases [Orriols 2009] (IR en adelante) se encuentra más allá de 1,5. Sin embargo no es extraño encontrar conjuntos de datos donde el IR se encuentra en proporciones superiores a 30. Una asignación de todas las instancias a la clase mayoritaria resulta en un falso clasificador con un ratio de acierto de un 90%. La figura 3.1 muestra un ejemplo de dos clases balanceadas claramente separables.

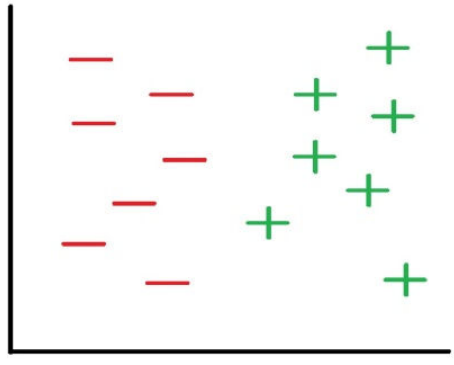

Figura 3.1 Datos balanceados

Actualmente existen muchos dominios de aplicación donde los conjuntos de datos son no balanceados y en los que el conocimiento más novedoso reside en las clases minoritarias [Cohen 2006]. La clase minoritaria se denomina habitualmente clase positiva y a la clase mayoritaria clase negativa.

Los algoritmos de clasificación cuando tratan con clases no balanceadas asignan más importancia a las reglas que cubren a la clase mayoritaria, puesto que el algoritmo de clasificación intenta reducir el error global usando medidas de precisión estándar. Por tanto estos algoritmos clasificarán con un alto porcentaje de aciertos los casos de la clase mayoritaria mientras la clase minoritaria tiende a estar mal clasificada. La figura 3.2 muestra un caso de clase no balanceada con poca separabilidad en la frontera entre clases. Este suele ser un problema añadido

dado que los ejemplos positivos rodeados de ejemplos negativos no generan reglas de alto valor y son difíciles de clasificar.

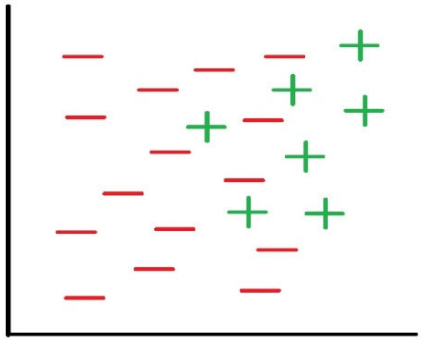

Figura 3.2 Datos no balanceados

La dificultad de aprendizaje en clases no balanceadas suele estar acompañada de los siguientes casos:

- Ejemplos ruidosos
- Solapamiento de clases
- Small Disjunts

Los ejemplos ruidosos interfieren con los ejemplos en las fronteras de las clases pudiendo hacer que la frontera de la clase se desplace para el algoritmo de clasificación. Asimismo las clases no balanceadas suelen crear small disjunts [Jo 2004] que pueden pasar desapercibidas para el algoritmo de clasificación.

La precisión es una medida de calidad de la clasificación basada en el conjunto de casos correctamente clasificados de cada una de las clases partido del total (3.1).

$$Precisión = \frac{TP+TN}{TP+TN+FP+TN} \qquad (4.1)$$

Tabla 3.1 Matriz de confusión en la clasificación binaria

	Predicción Positiva	Predicción Negativa
Clase positiva	*True Positive* (TP)	*False Positive* (FN)
Clase negativa	*False Positive*(FP)	*True Negative* (TN)

Otras medidas de calidad evalúan el comportamiento sobre una de las clases. La sensibilidad (3.2) mide la probabilidad de la clase positiva de ser clasificada y la especificidad la de la clase negativa. La especificidad mide la probabilidad de la clase negativa de ser clasificada (3.3)

$$Sensibilidad = \frac{TP}{TP+FN} \quad (3.2)$$

$$Especificidad = \frac{TN}{FP+TN} \quad (3.3)$$

La medida de un clasificador con precisión en las dos clases es una combinación de ambos términos, denominada media geométrica [Barandela 2003] (GM en adelante) (3.4)

$$GM = \sqrt{\frac{TP}{TP+FN} * \frac{TN}{FP+TN}} \quad (3.4)$$

Otra medida común en el área es AUC (Area under the ROC) [Bradley 1997]. ROC (Receiver Operating Characteristic) es una representación gráfica del funcionamiento de un clasificador para todos los contextos posibles. Se normaliza la matriz de costes de la tabla 3.1 para tener en cuenta el coste del error en la clasificación. AUC es un índice basado en ROC que mide la eficiencia en la clasificación para clasificadores discretos (en este contexto, que predice entre un número de clases determinado sin dar un valor de confianza a la predicción). Este índice se puede interpretar como la probabilidad de que un clasificador ordenará o puntuará una instancia positiva elegida aleatoriamente más alta que una negativa.

Los enfoques para tratar el problema de las clases no balanceadas se pueden dividir en nivel de datos y nivel algorítmico.

Nivel de datos, se realiza un preprocesamiento para balancear los datos de entrenamiento. El sobre-muestreo de la clase minoritaria equilibra el conjunto de datos y conserva los ejemplos importantes. Por otro lado existen algoritmos de muestreo de la clase mayoritaria que además del balanceo del conjunto puede realizar tareas de limpiado de la frontera entre clases o de instancias ruidosas. Existen enfoques híbridos que intentan delimitar las fronteras entre clases aplicando un sobre muestreo y un posterior limpiado de instancias de una o ambas clases.

Una de las técnicas de sobre-muestreo más utilizadas es SMOTE [Chawla 2002]. Este método genera sintéticamente ejemplos de la clase minoritaria basándose en puntos intermedios de los atributos de los ejemplo de la clase minoritaria. Un posible problema con SMOTE es que puede introducir ejemplos de la clase minoritaria en el área de la clase mayoritaria. También es un problema común en técnicas de sobre-muestreo la generación de ruido, pudiendo convertir instancias ruidosas en relaciones espurias que el algoritmo de clasificación interpretará como reglas de alto valor. Para paliar estos problemas SMOTE se puede combinar con ENN (Edited Nearest Neighbor) o Tomek links, métodos para la selección de instancias que mejora el comportamiento del conjunto generado. Estos métodos se pueden utilizar de manera independiente para reducir el número de instancias de la clase mayoritaria, sin embargo producen peores resultados que el sobre-muestreo [Batista 2004]. La figura 3.3 muestra el proceso de preprocesamiento, que partiendo de un conjunto de datos, genera otro permitiendo a un algoritmo tratar éste como si fuese balanceado. El procesamiento de problemas con múltiples clases implica una dificultad adicional para los algoritmos de MD, dado que las fronteras entre las clases pueden estar superpuestas, causando un decremento en el nivel de rendimiento. En esta situación, podemos proceder trasformando el problema multi-clase original en subconjuntos binarios, que son más fáciles de discriminar, a través de una técnica de binarización [Orlovsky1978] [Knerr 1990] (OVO: *One-vs ONE*) o de una clase contra las demás (OVA: *One-vs-All)* [Rifkin and Klautau, 2004].

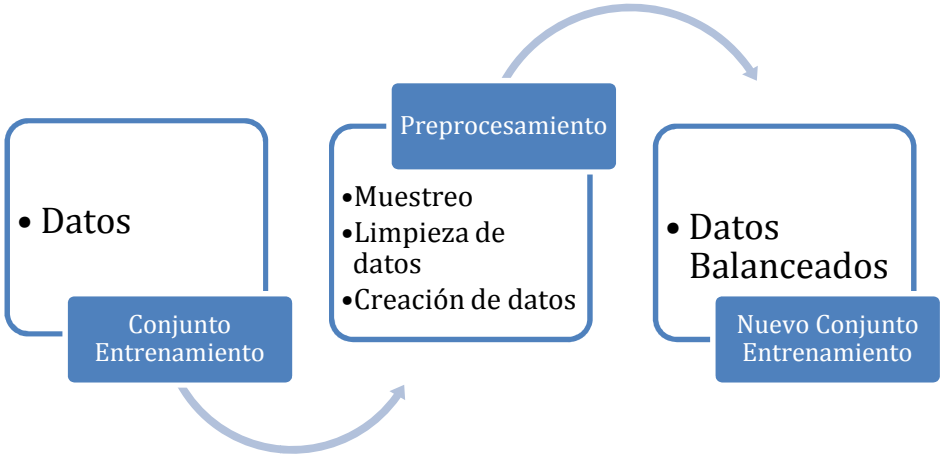

Figura 3.3 Estrategias de datos

En el **Nivel algorítmico**, las soluciones incluyen el ajuste de costes de las distintas clases del problema de tal forma que la clase menos representada es más costosa a efectos de clasificación, Por ejemplo en algoritmos basados en árboles de decisión resultaría en el ajuste de la estimación de probabilidad de las hojas de un árbol [Weiss 2003] y en el aprendizaje basado en reglas en el ajuste del coste del error de clasificación según la proporción de la clase [Domingos 1999][Sun 2007][Sen 2008][Weis 2004]. En [Chawla 2008] se establece una base empírica de comparación entre las estrategias de coste y de datos en problemas de clasificación no balanceados.

3.2 Consideraciones de Diseño

Esta subsección describe las modificaciones realizadas al modelo distribuido para tratar los conjuntos con clases no balanceadas. Se subdivide en varias subsecciones para explicar las estrategias utilizadas a distintos niveles. El algoritmo realiza una adaptación híbrida entre la adaptación del algoritmo para tener en cuenta el coste [Turney 1995] y el metalearning o modelo de datos producido por el rebalanceo de distribución de datos. La primera Sección está dedicada a explicar la redistribución desigual de las instancias según su clase y las

siguientes a las adaptaciones en el algoritmo para tener en cuenta el coste de error de clasificación sobre la clase positiva.

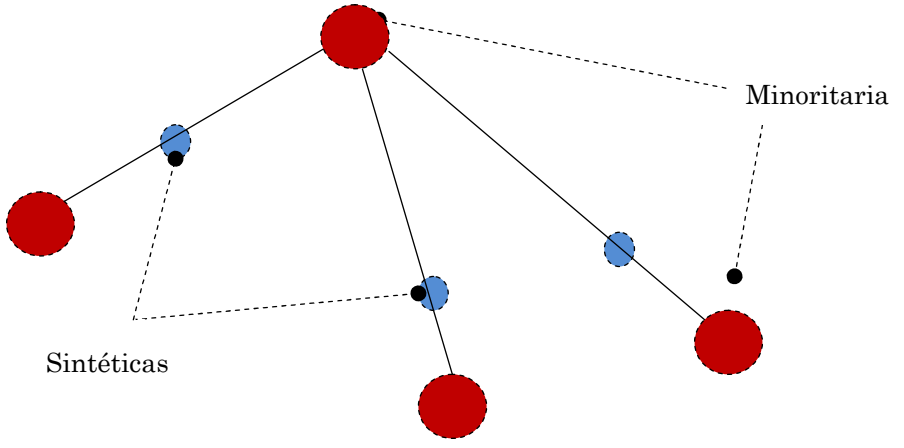

Figura 3.4 Sobre-muestreo por SMOTE

3.2.1 Balanceo de Clases Distribuido

La distribución de datos utilizada en el modelo propuesto en el Capítulo 1 tiene como fin mantener una escalabilidad horizontal respecto de los datos. Se ha modificado el algoritmo para repartir el conjunto de la clase minoritaria de manera integra a cada nodo y de esta manera rebalancear las clases a nivel de nodo. El número de nodos que componen el sistema se calcula de forma que ambas clases se encuentren balanceadas a nivel de los nodos AGL. Por ejemplo, un conjunto de datos con un IR de 10 generará una distribución de datos en 10 nodos, donde en cada nodo habrá un 10% representando a la clase mayoritaria y el total de las instancias de la clase minoritaria

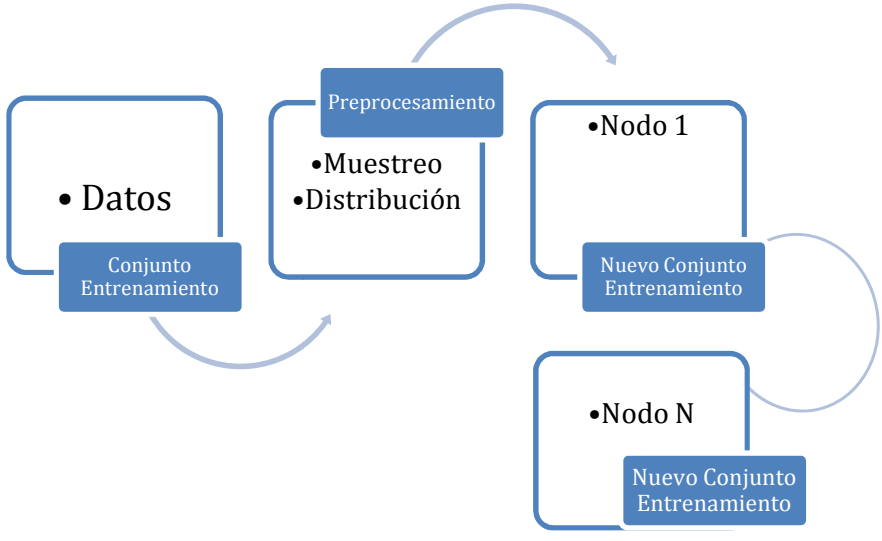

Figura 3.5 Distribución y copia de elementos no balanceados

Este procedimiento tiene el mismo efecto que el preprocesamiento de las bases de datos para crear copias de las clases no balanceadas. La duplicación de la clase positiva en cada nodo mejora al sobre-muestreo de la clase positiva en una población única porque no genera multiplicación de instancias ruidosas. Respecto del preprocesamiento por creación de instancias sintéticas (SMOTE), La copia de clases positivas en los nodos no produce sobre-generalización, ya que en la población local sólo hay una instancia y es un valor original. Asimismo se puede considerar un método de muestreo, ya que en cada nodo local se encontrará un subconjunto no disjunto de las instancias negativas existentes en el conjunto.

El muestreo de la clase mayoritaria se realiza mediante un muestreo aleatorio. Una vez repartidas todas las instancias en los nodos se realiza un nuevo muestreo hasta completar el balanceo de las clases en los nodos evitando que existan instancias duplicadas en los nodos.

La técnica de compartición de datos (ver 2.1.2) distribuye los datos mal representados a otros nodos. Este flujo de datos, cuando se trabaja en el modo no balanceado, crea un desbalanceo dado que los únicos casos que se pueden copiar pertenecen a la clase mayoritaria. El uso de otras técnicas en el algoritmo tiene en cuenta este efecto con medidas de coste adaptado a la desproporción entre clases.

3.2.2 Adaptación de la Función de Evaluación

A nivel algorítmico [Domingos 1999] una de las formas más efectivas de modificar un algoritmo es modificar el impacto que tiene un error de clasificación. La función de evaluación en el EDGAR (2.2.5), está diseñada para minimizar los casos negativos y la longitud de la regla. Inicialmente ambas clases tienen en los nodos el mismo número de instancias por lo que la función de evaluación sigue siendo equitativa con la medida de la cobertura sobre cada clase. Sin embargo a medida que las instancias negativas de difícil cobertura se copian en otros nodos al aplicar la técnica de compartición de datos, ésta producirá un desbalanceo progresivo. Es necesario por tanto tener en cuenta el desbalanceo a nivel algorítmico. Para ello, la formula de evaluación se modifica para incluir el IR de la clase sobre el total (3.5), calculado sobre la población de cada AGL. En este caso el IR de la clase minoritaria corresponderá con el IR en la población. Sin embargo para la clase mayoritaria el valor corresponderá a IR^{-1}.

$$f(r) = \left(1 + \frac{ceros(r)}{longitud(r)}\right)^{casos-} \tag{2.3}$$

$$f_{NB(r)} = \left(1 + \frac{ceros(r)}{longitud(r)}\right)^{IR*casos-} \tag{3.5}$$

3.2.3 Selección

El AGL usa un operador adaptado al aprendizaje de reglas denominado Sufragio Universal (ver 2.2.7 Operador de Selección). Este operador elige aleatoriamente elementos del conjunto de datos. Con estos ejemplos se seleccionan de la población las reglas que los cubren. Para cada ejemplo se elige mediante una ruleta proporcional a su función de evaluación, el mejor representante que será uno de

los padres elegidos. Al modificar la función de evaluación para tener en cuenta el IR, los individuos (reglas) elegidos que clasifiquen correctamente la clase positiva serán potenciados frente a los individuos que lo clasifiquen erróneamente, que tendrán una menor cuota en la ruleta proporcional.

Sin embargo cuando el IR aumenta el número de ejemplos positivos en la población local será menor y la probabilidad de generar reglas que los cubras será directamente proporcional a este desequilibrio. Se ha modificado el operador de selección para coger igual número de instancias de una y otra clase cuando se selecciona el conjunto de ejemplos que servirán para realizar el votado.

3.2.4 Generación del Clasificador

El AGL selecciona las mejores reglas para ser enviadas al supervisor, para ello genera un clasificador basado en la población local siguiendo un algoritmo voraz que prima las reglas con mejor valor y mayor cobertura (3.6). Esta fórmula (se ha modificado para que tenga en cuenta IR de cada clase, multiplicando la evaluación de orden por este factor relativo (4.6).

$$pi = Fitness * CasosPositivos \quad (2.5)$$

$$pi = Fitness * CasosPositivos * IR_{Clase} \quad (3.6)$$

Por otro lado el supervisor mantiene una población de todos los individuos recibidos y genera un clasificador contra el conjunto completo de datos cada cierto número de aportes de los AGLs. Éste clasificador está modificado de la misma forma para tener en cuenta el IR de la clase, pero al tener la población global, la composición global del clasificador variará sustancialmente por dos motivos:

El IR global será considerablemente mayor que en los AGLs y por tanto las reglas de la clase positiva con escasa cobertura podrán subir a los primeros puestos en el clasificador si la calidad (muy dependiente de la falta de errores) es suficiente al ser multiplicado por el IR.

La evaluación de la clase positiva SI será la misma que en los AGL, tanto para las reglas que clasifican a la clase positiva como a la clase negativa, porque en los AGL se encuentra la población total de la clase positiva. Esta circunstancia

favorece la correcta clasificación de la clase positiva y puede dificultar la de la clase negativa cuyas reglas seguramente van a tener una peor calidad.

El clasificador es una lista ordenada, en la que los casos cubiertos por las reglas precedentes son eliminados para evaluar las siguientes reglas. La ponderación de la clase positiva genera en principio clasificadores más largos al eliminar menos elementos en las reglas situadas en las primeras posiciones.

3.3 Estudio Experimental

En esta Sección se describe el estudio experimental del modelo adaptado sobre un conjunto de datos no balanceados.

En este estudio experimental se realiza la comparación con algoritmos de clasificación supervisados, distribuidos y no distribuidos con el modelo distribuido EDGAR para datos no balanceados. Se pretende comparar la efectividad del algoritmo modificado frente al preprocesamiento previo de los datos para balancear las clases. Los algoritmos de referencia también serán comparados con y sin preprocesamiento para mostrar el comportamiento del algoritmo modificado en ambas condiciones.

Esta Sección se compone de 5 subsecciones en las que se describirá respectivamente los conjuntos de datos empleados en la experimentación, los algoritmos de comparación, los tests estadísticos seleccionados y finalmente el procedimiento de validación.

3.3.1 Metodología de Comparación

Como se ha comentado en la subsección 3.1, la precisión no mide adecuadamente el grado de clasificación en los conjuntos no balanceados al no distinguir el coste del error relativo sobre una clase desbalanceada. En este estudio se utiliza la media geométrica [Barandela 2003] (GM), que se estima la precisión en cada una de las dos clases con el mismo peso según la fórmula:

$$GM = \sqrt{\frac{TP}{TP+FN} * \frac{TN}{FP+TN}} \quad (3.7)$$

3.3.2 Algoritmos Comparados

A efectos de comparación con EDGARNB se consideran los siguientes algoritmos genéticos distribuidos: REGAL-TC y REGAL. Se han elegido algoritmos de clasificación representativos del estado del arte según el comportamiento mostrado por [Fernández 2010]. Los algoritmos seleccionados son en primer lugar un conjunto de algoritmo evolutivos: Gassist (Pittsburgh) [Bacardit 2007], SIA (IRL) [Cantú-Paz 2003], UCS (Michigan) [Bernadó 2003], OCEC (CCGL) [Jiao 2006] y en segundo lugar dos referencias no evolutivas C4.5 [Quinlan q993] y RIPPER [Cohen 1995]. En las tablas 3.3 y 3.4 se resumen los parámetros utilizados según aparecen por defecto en la herramienta de comparación utilizada Keel [Alcalá 2009].

Alguno de los algoritmos utilizados para la comparación, a saber EDGAR, EDGARNB, REGAL, REGALTC, OCEC y COGIN necesitan un paso previo de discretización para tratar valores numéricos. Se ha discretizado aquellos conjuntos de datos con valores continuos mediante el discretizador Chi2Merge [Liu 1997]. Este discretizador tiene resultados competitivos para la mayoría de los datasets y algoritmos de comparación [García 2013].

Tabla 3.3 Algoritmos genéticos distribuidos de clasificación

Método	**Parámetros**
REGAL	Number of nodes = 6, Population Size per node = 133, Maximum number of generations = 500, Maximum number of iterations for freezing = 30, Generation Gap = 0.9, Cross Probability = 0.6, A = 0.5, B = 0.5, Mutation Probability = 0.001, Migration rate = 0.2
EDGAR	Nodos = 5, Tamaño Población = 50, Representados = 2, Topología = 1, GenSinCom = 50, épocas = 50
EDGARNB	Tamaño Población = 50, Representados = 2, Topología = 1, GenSinCom = 50, épocas = 50
REGAL-TC	Number of nodes = 6, Population Size per node = 133, Maximum number of generations = 500, Maximum number of iterations for freezing = 30, Generation Gap = 0.9, Cross Probability = 0.6, cross a = 0.5, cross b = 0.5, Mutation Probability = 0.001, Fitness A = 0.1, Migration rate = 0.2, Qws = 0.05, w1 = 0.5, w2 = 0.5

Tabla 3.4 Algoritmos de clasificación no distribuidos

Método	**Parámetros**
Gassist	Threshold in Hierarchical Selection = 0, Iteration of Activation for Rule Deletion Operator = 5, Iteration of Activation for Hierarchical Selection = 24, Minimum Number of Rules before Disabling the Deletion Operator = 12, Minimum Number of Rules before Disabling the Size Penalty Operator = 4, Number of Iterations = 750, Initial Rules = 20, Population Size = 400, Crossover Probability = 0.6, Probability of Individual Mutation = 0.6, Probability of Value 1 in Initialization = 0.90, Tournament Size = 3, Possible sizes of an attribute = {4, 5, 6, 7, 8, 10, 15, 20, 25}, Maximum N. of Intervals per Attribute = 5, psplit = 0.05, pmerge = 0.05, Probability of Reinitialize Begin = 0.03, Probability of Reinitialize End = 0, Use MDL = true, Iteration MDL = 25, Initial Theory Length Ratio = 0.075, Weight Relaxation Factor = 0.90, Class Initialization Method = cwinit, Default Class = auto
Oblique-DT	Number of Total Generations for the GA = 25, Population size = 20
SIA	Number of iterations = 200, α= 150, β= 0, Threshold Strength = 0
UCS	Number of explores =100000, population size =6400, δ= 0.1, acc_0 = 0.99, Pcross = 0.8,Pmut = 0.04, θGA = 50.0, θdel = 50.0, θsub = 50.0, doGASubsumption = true, r_0 = 0.6,type of selection = RWS, type of mutation = free, type of crossover = 2 point, m0 = 0.1
OCEC	Number of Total Generations = 500, Number of migrating/exchanging members = 1
COGIN	Misclassification error level=2, gen. limit=1,000, crossover rate=0.9, negation bit=yes
RIPPER	Size of growing subset = 66, Repetitions of the optimization stage = 2

3.3.3 Conjuntos de Datos

En este estudio hemos seleccionado 23 datasets de la UCI [Merz 1996] con diferentes ratios de desbalanceo (IR) entre las clases minoritarias y mayoritarias. Los datasets multiclase han sido modificados para obtener ficheros binarios con clases desbalanceadas en los que las clases positivas o negativas han sido agregadas para formar dos clases respectivamente positivas y negativas. EL nombre dado al conjunto de datos indica la unión de las clases que forman estos dos conjuntos. Por ejemplo Ecoli 0-3-4vs 5, indica que se han etiquetado las clases 0, 3 y 4 como una sola clase contra la clase 5.

Tabla 3.2 Conjuntos de datos para experimentación no balanceada

#n	C. Datos	At.	#Ej.	IR
1	pima	8	768	1.87
2	glass0	9	214	2.06
3	yeast1	8	1484	2.46
4	vehicle1	18	846	2.9
5	vehicle3	18	846	2.99
6	vehicle0	18	846	3.25
7	yeast3	8	1484	8.1
8	ecoli3	7	336	8.6
9	page-blocks0	10	5472	8.79
10	ecoli-0-3-4vs5	7	200	9
11	ecoli-0-6-7vs3-5	7	222	9.09
12	yeast-0-2-5-7-9vs3-6-8	8	1004	9.14
13	ecoli-0-4-6vs5	6	203	9.15
14	vowel0	13	988	9.98
15	ecoli-0-1-4-7vs5-6	6	332	12.28
16	glass4	9	214	15.47
17	page-blocks-1-3vs4	10	472	15.86
18	abalone9-18	8	731	16.4
19	glass-0-1-6vs5	9	184	19.44
20	glass5	9	214	22.78
21	yeast-2vs8	8	482	23.1
22	yeast4	8	1484	28.1
23	Ecoli-0-1-3-7vs2-6	7	281	39,14

La tabla 3.2 resume los conjuntos de datos utilizados indicando la primera columna un índice para referenciar los datasets en algunas figuras en vez de utilizar el nombre, la segunda el nombre del conjunto de datos, la tercera el número de atributos, la cuarta el número de instancias y la quinta el grado de desbalanceo (ratio IR.). La figura 3.6 muestra el grado de desbalanceo del conjunto de datos en orden de IR creciente.

Respecto a los conjuntos de test y prueba, se ha utilizado un particionamiento fivefold cross-validation.

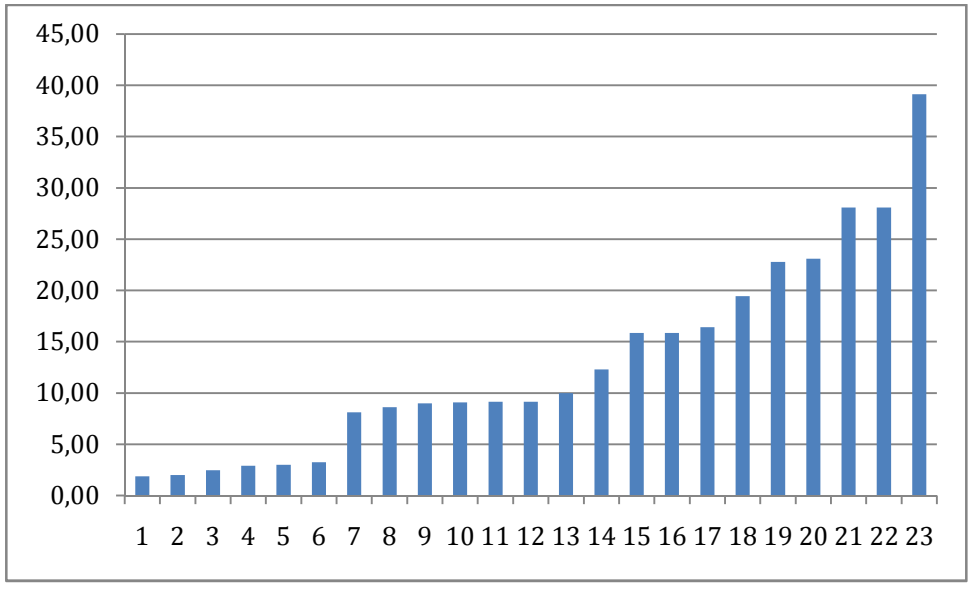

Figura 3.6 Representación de índice de desbalanceo de clases

3.3.4 Análisis Estadístico

Para validar el proceso se han utilizado pruebas no paramétricas [Demšar 2006] [García 2009]. En el apéndice A al final del documento se detallan las características de estos test y condiciones de uso.

Para la comparación múltiple el test seleccionado es Friedman 1: N y Wilcoxon Signed-Rank 1:N. La utilización de Wilcoxon permite comparar dos algoritmos entre sí cuando el algoritmo de referencia en Friedman no tiene suficiente diferencia para rechazar la hipótesis nula. Los test han sido ejecutados con la herramienta Keel [Alcalá 2009].

En caso de existir diferencia estadísticas se ha utilizado el test post-hoc de Finner para observar la diferencia de rendimiento ente los métodos y el mantenimiento o rechazo de la hipótesis con el nivel de significancia fijado.

3.3.5 Comparación con Algoritmos de Referencia

Hemos realizado una comparación de todos los algoritmos en igualdad de condiciones para medir el grado de clasificación en GM con el conjunto de datos no balanceado.

Para comprobar la eficacia del algoritmo propuesto se realiza un segundo experimento donde se preprocesan los datasets con la técnica SMOTE para generar un conjunto balanceado mediante generación de sintéticas,

La tabla 3.5 muestra las ejecuciones de los algoritmos seleccionados con los datasets en orden de IR de más bajo a más alto. La tabla 3.6 muestra las ejecuciones de los mismos algoritmos sobre los conjuntos de datos rebalanceados con SMOTE.

Tabla 3.5 Comparativa con algoritmos en promedio GM

C. Datos	EDGAR	ED_NB	Gassist	REGAL	RE_TC	ObliqueDT	SIA	UCS	OCEC	COGIN	C45	Ripper
pima	0.617	0.663	0.676	0.663	0.655	0.668	0.644	**0.702**	0.692	0.572	0.684	0.701
glass0	0.734	0.690	**0.798**	0.769	0.761	0.748	0.764	0.342	0.784	0.733	0.783	0.740
yeast1	0.507	0.622	0.603	0.630	0.615	0.628	0.589	0.657	0.654	0.542	0.686	**0.688**
vehicle1	0.587	0.677	0.549	0.648	0.624	0.668	0.600	0.658	0.670	0.562	0.647	**0.720**
vehicle3	0.608	**0.704**	0.574	0.662	0.640	0.654	0.525	0.654	0.676	0.601	0.680	0.703
vehicle0	0.833	0.899	0.913	0.914	0.921	0.907	0.780	0.925	0.905	0.833	0.934	**0.933**
yeast3	0.756	0.857	0.845	0.896	0.894	0.809	0.800	0.895	**0.910**	0.841	0.871	0.898
ecoli3	0.536	0.765	0.724	0.804	0.773	0.731	0.620	0.501	**0.830**	0.386	0.696	0.809
ecoli-0-3-4	0.777	0.832	0.837	0.669	0.881	0.808	0.719	0.523	**0.884**	0.693	0.842	0.867
ecoli-0-6-7	0.696	**0.900**	0.766	0.687	0.857	0.720	0.478	0.148	0.713	0.749	0.826	0.797
yeast-0-2-5-7-9	0.755	0.730	0.874	0.668	0.817	0.837	0.813	0.872	0.871	0.635	0.871	**0.888**
ecoli-0-4-6	0.673	0.845	0.801	0.829	0.837	0.800	0.763	0.661	0.791	0.744	0.828	**0.857**
vowel0	0.797	0.877	0.968	0.862	0.851	0.953	**1.000**	0.945	0.876	0.752	0.968	0.933
ecoli-0-1-3-7	0.483	0.730	0.630	0.000	0.000	0.555	0.080	0.705	0.200	**0.833**	0.736	0.663
Page-blocks0	0.890	0.890	0.800	0.936	0.931	0.912	0.859	0.854	**0.942**	0.900	0.925	0.933
page-blocks-1-3	0.700	0.840	0.969	0.597	0.937	0.969	0.660	0.963	0.941	0.693	**0.998**	0.995
abalone9-18	0.288	0.670	0.499	0.640	0.678	0.642	0.374	0.412	**0.710**	0.382	0.431	0.689
glass-0-1-6	0.493	0.831	0.439	0.840	0.840	0.704	0.586	0.742	**0.899**	0.159	0.794	0.675
glass5	0.131	0.755	0.243	0.187	0.708	0.837	0.614	0.500	**0.912**	0.094	0.880	0.744
yeast-2vs8	0.506	**0.739**	0.694	0.458	0.717	0.602	0.000	0.651	0.622	0.639	0.469	0.737
glass4	0.110	0.765	0.789	0.642	0.794	0.803	**0.889**	0.494	0.771	0.267	0.582	0.692
yeast4	0.249	0.690	0.175	0.670	0.665	0.556	0.340	0.612	**0.742**	0.142	0.553	0.661
ecoli-0-1-4-7	0.672	0.848	0.849	0.549	0.741	0.795	0.475	0.687	0.844	0.654	**0.927**	0.833

Tabla 3.6 Comparativa con algoritmos en Promedio GM + SMOTE

C. Datos	EDGAR	EDGARNB	REGAL	REGALTC	GASSIST	obliqueDT	SIA	UCS	OCEC	COGIN	C45	Ripper
pima	0.648	0.646	0.663	0.651	**0.727**	0.654	0.702	0.721	0.712	0.610	0.713	0.685
glass0	0.732	0.727	0.732	0.741	**0.820**	0.774	0.789	0.635	0.800	0.720	0.754	0.731
yeast1	0.519	0.584	0.647	0.417	**0.713**	0.657	0.704	0.709	0.651	0.600	0.699	0.682
vehicle1	0.546	0.672	0.674	0.551	**0.749**	0.680	0.651	0.728	0.685	0.640	0.696	0.713
vehicle3	0.554	0.693	0.667	0.587	0.731	0.675	0.650	0.732	0.697	0.620	0.679	**0.732**
vehicle0	0.890	0.899	0.912	0.907	0.932	0.923	0.818	**0.949**	0.911	0.900	0.924	0.931
yeast3	0.804	0.817	0.811	0.851	0.918	0.848	0.865	0.887	0.864	0.840	**0.922**	0.899
ecoli3	0.550	0.790	0.856	0.830	0.850	0.724	**0.859**	0.842	0.823	0.700	0.810	0.842
ecoli-0-3-4	0.748	0.863	**0.912**	0.904	0.880	0.868	0.832	0.882	0.869	0.860	0.857	0.863
ecoli-0-6-7	0.712	0.744	0.707	0.697	0.759	0.764	0.758	0.785	0.769	0.700	0.753	**0.802**
yeast-0-2-5-7-9	0.772	0.840	0.767	0.850	**0.922**	0.861	0.864	0.892	0.876	0.870	0.897	0.884
ecoli-0-4-6	0.663	0.859	0.831	0.896	0.824	0.830	0.809	0.870	**0.909**	0.830	0.866	0.871
vowel0	0.788	0.893	0.916	0.898	0.950	0.965	**0.996**	0.964	0.919	0.900	0.947	0.938
ecoli-0-1-3-7	0.676	0.512	0.532	0.505	0.670	0.610	0.556	0.708	0.712	0.530	**0.712**	0.707
page-blocks-1-3	0.719	0.948	0.810	0.924	0.978	0.990	0.884	0.982	0.949	0.930	**0.994**	0.989
Page-blocks0	0.880	0.888	0.934	0.829	0.888	0.921	0.829	0.920	0.936	0.940	**0.946**	0.944
abalone9-18	0.273	0.527	0.568	0.561	0.681	0.566	**0.684**	0.656	0.558	0.450	0.591	0.667
glass-0-1-6	0.492	0.852	0.844	0.826	0.766	0.797	0.835	0.799	0.886	0.890	0.721	**0.925**
glass5	0.736	0.921	0.869	0.947	0.744	0.852	0.701	0.900	0.884	0.890	0.872	**0.966**
yeast-2	0.524	0.771	0.767	0.782	0.775	0.708	0.524	0.726	**0.788**	0.690	0.750	0.756
glass4	0.553	**0.899**	0.862	0.862	0.862	0.847	0.892	0.788	0.852	0.860	0.878	0.851
yeast4	0.261	0.707	0.735	0.706	**0.770**	0.567	0.682	0.723	0.642	0.640	0.664	0.729
ecoli-0-1-4-7	0.707	0.844	0.875	0.864	0.856	0.830	0.824	**0.885**	0.813	0.760	0.880	0.847

Tabla 3.7 Ranking Friedman sin SMOTE

Algorithm	**Ranking**
EDGAR	10
EDGARNB	51.739
Gassist	64.783
REGAL	69.783
REGALTC	55.217
ObliqueDT	06
SIA	90.435
UCS	70.217
OCEC	39.783
COGIN	98.478
C45	45.217
Ripper	34.348

Tabla 3.8 Friedman p-Values ajustados sin SMOTE

i	Algoritmo	z=(R0−Ri)/SE	p	Finner
11	EDGAR	6.174.849	0	0.004652
10	COGIN	6.031.724	0	0.009283
9	SIA	5.275.202	0	0.013892
8	UCS	3.373.676	0.000742	0.018479
7	REGAL	3.332.783	0.00086	0.023045
6	Gassist	2.862.513	0.004203	0.02759
5	ObliqueDT	2.412.689	0.015835	0.032114
4	**REGALTC**	**1.962.866**	**0.049662**	**0.036617**
3	**EDGAR-NB**	**1.635.722**	**0.101898**	**0.041099**
2	**C45**	**1.022.326**	**0.306627**	**0.04556**
1	**OCEC**	**0.511163**	**0.609237**	**0.05**

Algoritmo de referencia Ripper

Finner rechaza Ho para p-value ≤0.036617

Las tablas 3.7 y 3.8 muestran los valores para el test de Friedman y los valores P ajustados para el test post-hoc. En ella se puede observar que EDGAR supera a la mayoría de los algoritmos pero no tiene diferencia estadística con REGALTC,

C4.5, OCEC y RIPPER. Respecto a los algoritmos genéticos distribuidos supera a EDGAR y a REGAL .La tabla 3.9 muestra los valores de Wilcoxon Signed-Rank, donde se aprecia que si bien no supera a REGALTC, lo supera por un pequeño margen. Respecto de C4.5 están en el mismo rango.

Tabla 3.9 Wilcoxon test 1:N

VS	R+	R-	Exact P	As. P
EDGAR	**250**	**3**	**0.0000024**	**0.000057**
Gassist	**196**	**80**	**0.0802000**	**0.075195**
REGAL	201.5	52	0.013411	0.013466
REGALTC	**143.5**	**133**	**≥ 0.2**	**0.848953**
ObliqueDT	175	101	≥ 0.2	0.254053
SIA	240	36	0.001118	0.001824
UCS	220	56	0.011156	0.012099
OCEC	92	184	≥ 0.2	1
COGIN	261.5	15	0.000029	0.000153
C45	136	140	≥ 0.2	1
Ripper	97	179	≥ 0.2	1

La figura 3.7 muestra en un gráfico radial el grado de clasificación medio de Edgar y Edgar NB, cada círculo concéntrico corresponde a 0,2 decimas de precisión y los números corresponden al índice de los conjuntos de datos en la tabla 4.3. En ella se aprecia como EDGARNB mejora sustancialmente a EDGAR en la zona con mayor desbalanceo de datos, con IR entre 15 y 39.mientra que en la zona de desbalanceo bajo, aunque de media mejora, el comportamiento es similar, probablemente por el efecto de las técnicas de envío de datos que permite cierto rebalanceo.

La figura 3.8 muestra el comportamiento de Ripper frente a EDGARNB, en esta figura se puede apreciar que si bien Ripper parece tener un mejor desempeño la diferencia en porcentaje es pequeña.

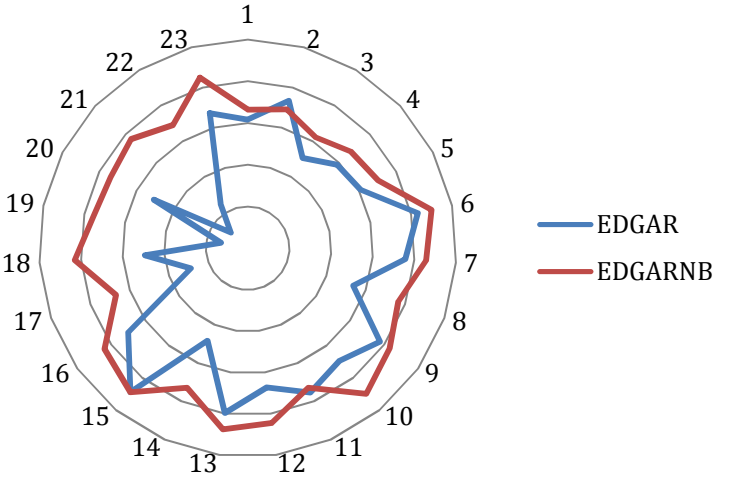

Figura 3.7. EDGAR vs EDGARNB

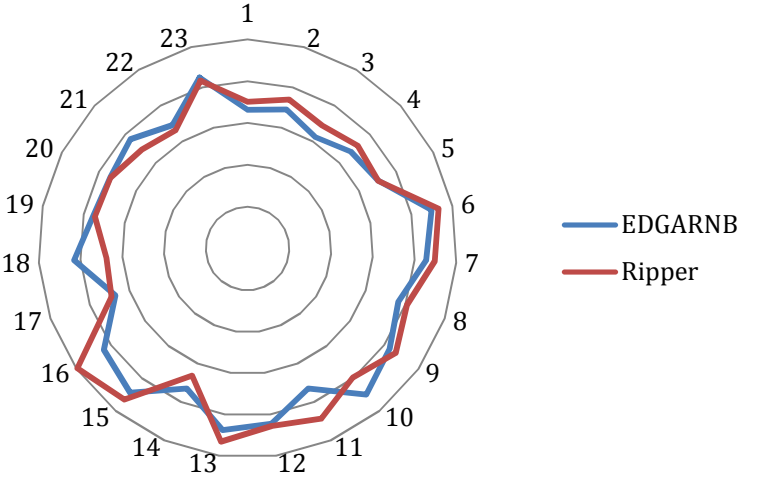

Figura 3.8. Ripper vs EDGARNB

Tabla 3.10 Ranking Friedman SMOTE

Algoritmo	Ranking
EDGARNB	6.6739
EDGARNB-S	6.7609
REGAL-S	7.1957
REGALTC-S	7.7609
Gassist-S	4.5652
ObliqueDT-S	7.7391
SIA-S	7.3043
UCS-S	4.7391
OCEC-S	5.9565
COGIN-S	9.1739
C45-S	5.3478
Ripper-S	4.7826

Tabla 3.11 Friedman p-Values ajustados con SMOTE

i	Algoritmo	z=(R0−Ri)/ SE	p	Finner
11	COGIN-S	4.334662	0.000015	0.004652
10	REGALTC-S	3.005639	0.00265	0.009283
9	ObliqueDT-S	2.985192	0.002834	0.013892
8	SIA-S	2.576262	0.009988	0.018479
7	REGAL-S	2.474029	0.01336	0.023045
6	EDGARNB-S	2.065099	0.038914	0.02759
5	**EDGARNB**	**1.983312**	**0.047333**	**0.032114**
4	**OCEC-S**	**1.308577**	**0.190678**	**0.036617**
3	**C45-S**	**0.736075**	**0.461685**	**0.041099**
2	**Ripper-S**	**0.204465**	**0.83799**	**0.04556**
1	**UCS-S**	**0.163572**	**0.870068**	**0.05**

Algoritmo de referencia Gassist-S

Finner rechaza Ho para un p-value ≤0.02759.

Respecto de la comparación con la combinación de conjuntos de datos preprocesados con SMOTE y los algoritmos de comparación, En la tabla 3.10, se han marcado con el sufijo -*S* los algoritmos con preprocesamiento SMOTE. Se ha comparado la versión del algoritmo modificado con y sin preprocesamiento.

Se puede observar que en la tabla 3.11 no hay grandes diferencias en cuanto a la comparación con el algoritmo comparado. EDGAR NB se sitúa en la franja de los mejores algoritmos aunque tras el preprocesamiento este grupo de algoritmos ha aumentado en 2: Gassist y UCS. Es reseñable que todos los métodos distribuidos se empeoran con SMOTE, REGALTC, EDGARNB y REGAL Obtienen peor rendimiento. En la figura 3.8 se aprecia que al igual que ocurría con Ripper la diferencia en porcentaje es pequeña. Tanto Ripper como Gassist trabajan directamente con datos numéricos. Pensamos que la discretización es un proceso que resta precisión a EDGAR_NB frente a algoritmos con capacidad de tratar datos numéricos.

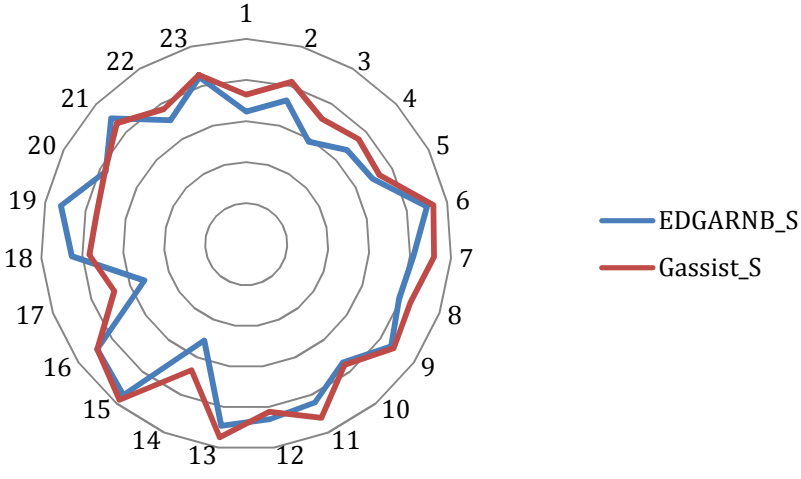

Figura 3.9. Ripper vs EDGARNB

Como conclusión remarcar que se ha utilizado la distribución de datos en nodos como una medida de balanceo dinámico de las clases. Esta medida junto con medidas de coste sobre el algoritmo permite una mejora importante en el tratamiento automático de las clases no balanceadas, siendo competitivo incluso con algoritmos de comparación incluyendo preprocesamiento SMOTE. Pensamos que una mejora en el tratamiento de los datos continuos puede mejorar el comportamiento con bases de datos no balanceadas.

Capitulo 4

Aprendizaje de Particiones para el Tratamiento de Valores Continuos

Una gran parte de los algoritmos de inducción de reglas supervisados necesita datos nominales. Por ello antes de utilizar un conjunto de datos numéricos es preciso realizar un preprocesamiento que genere un conjunto de datos discretizado. La elección del algoritmo de discretización adecuado no es simple y como se muestra en [García 2013], éste es un proceso que afecta a la precisión final, dependiendo de la combinación especifica de conjunto de datos/discretizador/algoritmo de aprendizaje.

El algoritmo EDGARNB no trata directamente con valores continuos en los atributos. En caso de necesitar tratar un conjunto de datos con atributos numéricos de carácter continuo, es necesario procesar el conjunto de datos para obtener intervalos fijos que el algoritmo tratará como valores nominales sin ninguna relación entre ellos ni información sobre el contenido de los mismos.

En el Capítulo 3 se ha visto que los algunos algoritmos como GASSIST, tienen un mejor comportamiento en el tratamiento de clases no balanceadas que EDGAR, aunque éste se comporta de manera adecuada respecto a otros algoritmos distribuidos y clásicos. En base a estos razonamientos proponemos adaptar EDGARNB para tratar directamente con datos continuos incluyendo un algoritmo de aprendizaje de las particiones. Pensamos que con esta mejora el algoritmo puede llegar a alcanzar el rendimiento de las mejores propuestas en el ámbito de la clasificación con datos no balanceados. .

Por otro lado la propia partición de los datos es en sí información útil sobre la estructura de los atributos, por lo que el algoritmo muestra la partición inicial y final una vez adaptada al conjunto de reglas del clasificador.

Este Capítulo se explica las técnicas implementadas sobre el algoritmo EDGARNB. En primer lugar la subsección 5.1 describe brevemente las teorías y trabajos tomados como referencia para adaptar el algoritmo al aprendizaje de particiones. En 5.2 mostramos las estrategias desarrolladas para adaptar el algoritmo al manejo de datos continuos, entre ellas cabe destacas el algoritmo CHC que realiza el aprendizaje de las particiones utilizando el clasificador generado por EDGAR como wrapper. Finalmente en 5.3 mostramos los resultados experimentales en condiciones similares a las del Capítulo anterior sobre clases no balanceadas, tanto en los conjunto de datos utilizados como en los algoritmos de comparación, manifestando un comportamiento competitivo con los mejores algoritmos de la muestra y sobrepasando a la mayoría de ellos en la clasificación de clases no balanceadas.

4.1 Introducción

El tratamiento directo de datos continuos es importante a la hora de aprender reglas de clasificación con precisión, ya que como se describe en [García 2013] el comportamiento de la discretización determina de manera substancial la calidad de los clasificadores obtenidos. Sin embargo como hemos comentado en el Capítulo 2, también es un medio eficaz de tratar con la complejidad de dimensional, reduciendo el espacio de búsqueda enormemente. Una gran cantidad de problemas de carácter continuo pueden ser tratados por algoritmos de clasificación con buenos resultados mediante la discretización. Pero hay problemas en los que por la composición intrínseca de los datos (multivariate), este preprocesamiento disminuye la calidad del clasificador producido. En estos casos los algoritmos que pueden aprender reglas directamente sobre los datos continuos tienen una ventaja competitiva al no depender de las bondades específicas del discretizador.

El aprendizaje de datos con atributos numéricos con algoritmos genéticos requiere tener en el cromosoma del mismo información sobre la parte del atributo que forma parte del antecedente de una regla, bien mediante una descripción

intervalar del mismo (atributo entre valor i y valor j) o mediante otras técnicas (lógica difusa, rough sets, etc...). EDGAR es un algoritmo fuertemente influenciado por su forma de representación. La representación de las reglas como cadenas binarias permite un lenguaje descriptivo rico al generar reglas en base a un conjunto de nombres (o particiones resultantes de la discretización) pero por otro lado la codificación de longitud fija de cromosomas determina los operadores genéticos utilizados para el aprendizaje. Proponemos en este trabajo, mantener intacta la estructura de aprendizaje, e decir mantener un cromosoma de longitud fija basado en una cadena binaria con una mejora de las particiones representadas por esta cadena en un proceso paralelo de aprendizaje. En este apartado vamos a revisar la estructura de reglas utilizada en el ámbito del aprendizaje intervalar de datos con datos continuos que servirán para encuadrar la estrategia de representación utilizada por el algoritmo propuesto.

Las reglas con antecedentes continuos se puede tratar de las siguientes formas según compartan o no los puntos de corte que definen las particiones a las que hace referencia la regla:

Reglas de tipo intervalar con intervalos locales. Cada regla utiliza particiones distintas de los datos y no hay particiones comunes para los datos. Aunque esta técnica a priori tiene más flexibilidad y por lo tanto podría dar como resultado una mejor clasificación, tiene el inconveniente de que es menos interpretable al no descubrir la estructura global de los atributos continuos.

Tratamiento intervalar global. En este caso el algoritmo tiene una base común de particiones que utilizan todas las reglas para que cuando se realiza una discretización, el algoritmo utilice los intervalos como una base de particiones común, sin embargo no tiene noción de la relación entre ellos y las particiones son fijas. Utilizar la información acerca de los límites de las particiones permite modificar éstos para conseguir hacer un aprendizaje de las particiones.

EDGAR implementa implícitamente (a través de los puntos de corte que ha realizado el discretizador previo) un tratamiento intervalar global, donde el algoritmo no tiene información sobre el carácter del atributo, es decir si éste numérico continuo, ordinal o nominal.

Respecto a las técnicas de discretización se pueden realizar taxonomías en varias dimensiones (*univariate* vs *multivariate*, dinámico vs estático, supervisado vs no supervisado, según medidas, ordinales vs nominales, etc.…). Vamos a repasar brevemente alguna de estas taxonomías:

Información ordinal: la inmensa mayoría de los discretizadores dan información nominal, no pudiendo realizarse posteriormente el aprendizaje de las reglas basadas en la contigüidad de valores ordinales presentes en la misma regla, y por tanto resultando en clasificadores menos compactos e interpretables.

Carácter estático vs dinámico: respecto a la integración con el clasificador, se refiere a la capacidad de interactuar con el algoritmo de aprendizaje para la generación de los intervalos. Un discretizador dinámico puede generar resultados más precisos y compactos. Prácticamente todos los algoritmo de discretización son de tipo estático, ya que los de tipo dinámico están normalmente embebidos dentro de un algoritmo de aprendizaje [García 2013].

Respecto al aprendizaje de las particiones respecto de la clase tenemos las siguientes corrientes:

Discretizadores no supervisados: se basan en medidas sobre los atributos sin tener en cuenta la clase a la que pertenece el ejemplo al llevar a cabo su discretización. Esto es, sólo tienen en cuenta los valores de los atributos, por lo que desechan parte de la información disponible en la relación con la clase. Ejemplos de estos son los de igualdad en frecuencia e igualdad en anchura. Estos dos discretizadores están ampliamente estudiados en numerosas publicaciones. [García 2013] muestra que el discretizador de igualdad en frecuencia da mejores resultados que el de igualdad en amplitud.

Discretizadores supervisados: tienen en cuenta a qué clase pertenece cada ejemplo para establecer los puntos de corte de las particiones. Dependiendo de la medida que determina si la partición es adecuada respecto a la clase:

Estadística: mide la correlación entre los atributos y la clase. ChiMerge [Kerber 1992] y Chi2Merge [Liu 1997] son algoritmos de este tipo que consiguen un rendimiento adecuado para la mayora de las tareas de clasificación.

Información: Maximiza la entropía del conjunto de particiones ID3 [Quinlan 1993].

Wrapper: Utiliza un método de aprendizaje con las particiones propuestas para dar una medida de calidad de la misma [Ventura 1994].

Rough Sets: utiliza las medidas propias de rough sets [Nguyen 1995].

En [García 2013] se aprecia que los mejores discretizadores para la el aprendizaje supervisado se encuentran en el grupo de los supervisados y entre ellos Chi2Merge es uno de los más representativos, mientras que los no supervisados no alcanza un rendimiento adecuado.

El uso de *wrappers* como técnica de medida no ha sido ampliamente utilizado en la discretización a pesar de que el uso del propio clasificador en la medida tiene características que *a priori* pueden ser una ventaja al ser inherentemente *multivariate* y dinámicos. Como hemos visto en la subsección de discretización estática/dinámica, el uso del discretizador integrado en el propio algoritmo de aprendizaje mejora la precisión del clasificador.

Por otro lado los algoritmos de clasificación basados en intervalos realizan normalmente un ajuste sobre los intervalos de manera independiente sobre cada regla con la consiguiente pérdida de interpretabilidad del clasificador.

Basándonos en estas consideraciones, proponemos integrar un discretizador dinámico basado en *wrapper* que realice un aprendizaje de las particiones globales de manera colaborativa con el proceso de aprendizaje de las reglas de clasificación. En este sentido [Ramírez-Gallego 2015] propone un discretizador que utiliza al igual que nuestra propuesta un sistema evolutivo para optimizar los puntos de corte utilizando un clasificador como *wrapper* consiguiendo un comportamiento mejorado frente a los principales discretizadores en ámbitos con fuerte relación entre atributos.

4.2 Consideraciones de Diseño

En este apartado describimos las principales decisiones de diseño necesarias para integrar un discretizador dinámico en el algoritmo de aprendizaje de reglas EDGARNB.

4.2.1 Codificación de las Particiones

EDGAR tiene un cromosoma de longitud fija determinado por una plantilla que determina que posiciones del cromosoma corresponden a cada valor de cada atributo (ver 2.2.2). Cuando se trata un valor continuo el discretizador da un conjunto de nombres a dichos intervalos pero la información numérica de ellos no es utilizada en ningún sentido por EDGAR. Sin embargo en EDGARNBC (Se añade *C* al nombre por el tratamiento Continuo de datos) se tiene constancia en la plantilla de los atributos continuos y nominales. Cada intervalo se entenderá como una comparación del tipo "valor > límite inferior" y "valor <= límite superior".

Tabla 4.1 Discretización de valor continuo

edad	Clase	Discretizado(10)	Binario	Discretizado(5)	Binario
29	Si	3 (20 a 30)	0010000000	2 (20 a 40)	01000
30	Sí	4 (30 a 40)	0010000000	2 (20 a 40)	01000
42	No	5 (40 a 50)	0000100000	2 (20 a 40)	00100
55	No	6 (50 a 60)	0000010000	3 (40 a 60)	00100
61	No	7 (60 a 70)	0000000100	4(60 a 80)	00010
75	Sí	8 (70 a 80)	0000000001	4(60 a 80)	00010

En la Tablas 4.1 y 4.2 se aprecia la codificación de una regla con 5 y 10 intervalos iniciales. Una vez creado el cromosoma, el número de intervalos para cada atributo es fijo. Este será definido por un proceso interno o bien establecido por parámetro. Este número de intervalos deberá entenderse como máximo, ya que con posterioridad el algoritmo realizará una reducción del número de intervalos.

Tabla 4.2 Ejemplo de reglas generadas para valor discretizado

	Regla	Edad	Clase	Casos Positivos	Casos Negativos
Discretizado 10	**r11**	**3,4,8**	**Si**	**3**	**0**
	r12	**5,6,7,**	**No**	**3**	**0**
Discretizado 5	**r21**	**2,4**	**Si**	**3**	**2**
	r22	**3**	**No**	**1**	**1**

Tabla 4.3 Cromosomas con 10 intervalos

	Regla	Edad	Clase
Discretizado 10	**r11**	**3,4,8**	**Si**
	r12	**5,6,7,**	**No**
Genotipo	**r11**	0011000100	**10**
	r22	0000111000	**01**

Con esta codificación, los operadores genéticos del algoritmo no cambian y siguen funcionando de manera similar a la versión para valores discretos, aun cuando los atributos conservan su naturaleza continua. Cuando se evalúa la cobertura de una regla sobre un individuo, el algoritmo comprobará los valores nominales con los presentes en la regla, y los numéricos serán comparados con los límites máximos y mínimos definidos por los puntos de corte para determinar que intervalo cubre al atributo del ejemplo.

Inspirados en el concepto de discretizador supervisado embebido (*wrapper*), se ha desarrollado un cromosoma intervalar que guarda por una parte un cromosoma para codificar la regla y otro cromosoma que gestiona los puntos de corte.

Se ha utilizado un discretizador supervisado Chi2Merge [Liu 1997] sobre la clase positiva para generar la primera plantilla que será optimizada posteriormente para adaptase a las reglas aprendidas. En [García 2013] este discretizador muestra un comportamiento general adecuado en el aprendizaje supervisado.

4.2.2 Modelo Distribuido de Aprendizaje

El modelo de aprendizaje tiene dos algoritmos genéticos que colaboran mutuamente, uno para las particiones y otro para las reglas:

Los AGLs distribuidos formulan reglas candidatas y las envían al nodo central donde son evaluadas contra la el conjunto de datos global.

Las particiones son aprendidas a través de un algoritmo genético que ajusta los límites de los intervalos usando el clasificador aprendido como función de evaluación de dicho cromosoma.

Respecto al aprendizaje de las particiones se pueden seguir dos orientaciones:

- Ajuste final de los puntos de corte: Se realiza un proceso de *tuning* de las particiones cuando se genera el clasificador final la Figura 4.1 muestra el proceso
- Coevolución del proceso de aprendizaje: Las particiones y el conjunto de reglas se aprenden alternativamente hasta que se alcanza un clasificador de calidad adecuada. La figura 4.2 muestra la interacción ente los componentes del sistema.

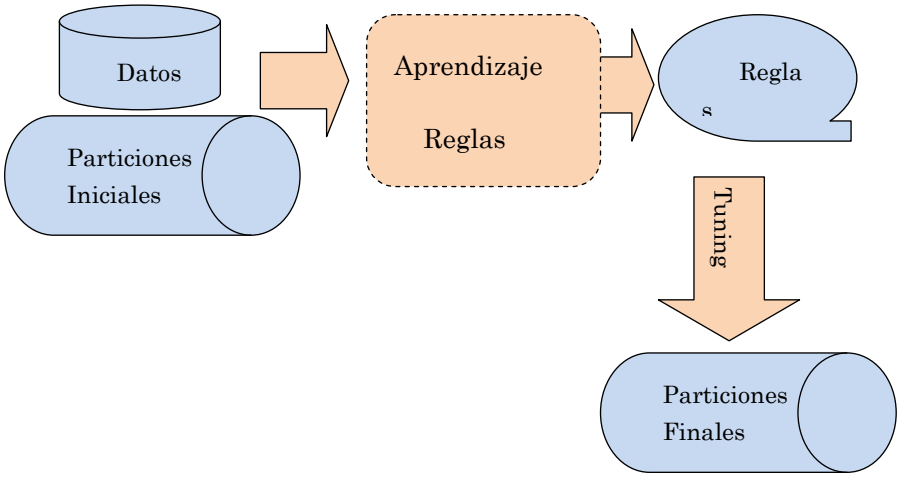

Figura 4.1 Tuning de particiones

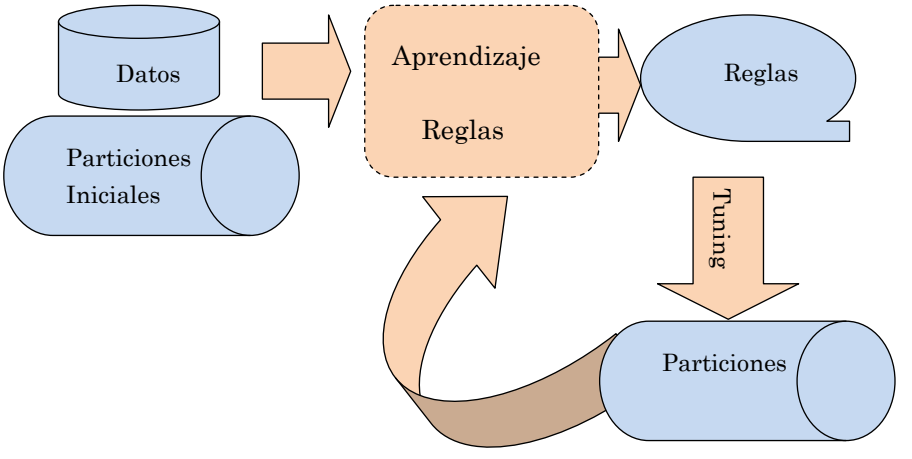

Figura 4.2 Aprendizaje de particiones

4.2.3 Aprendizaje de las Particiones

El aprendizaje de las particiones se realiza a través de un algoritmo de codificación real con mecanismos de diversidad y alta presión selectora. Se ha utilizado el algoritmo CHC [Eshelman 1991] para mejorar los intervalos inicialmente establecidos, exceptuando el primer y último punto de la partición que son fijos, ya que estarán definidos por el valor más bajo y más alto que puede tomar dicho atributo respectivamente. Como se ha comentado en la subsección anterior es posible tener dos aproximaciones al problema, el tuning final de las particiones o la coevolución continua de particiones.

4.2.3.1 Ajuste Final

El ajuste final de particiones es un proceso de optimización genética de las particiones iniciales basado en la utilización del clasificador generado por el modelo como wrapper.

Los AGL enviarán reglas a sus vecinos y al supervisor hasta que se produzca el criterio de parada (algoritmo 4), en dicho momento se genera el clasificador y se ejecuta el algoritmo CHC de optimización de las particiones para ajustarse a las reglas existentes en el clasificador final.

```
Mientras no se cumpla condición de parada
Los AGL generan nuevas reglas
Los AGL proporcionan reglas al pool
Cuando se cumple la condición de parada:
El Servidor genera un nuevo clasificador con esas reglas mediante un greedy
    CHC optimiza las particiones con el clasificador actual
```

Pseudocódigo 4. Esquema de tuning de particiones

Se impone un criterio de parada exigente al CHC para que la optimización sea la máxima posible ya que el algoritmo sólo se ejecutará una vez. Se establece como

criterio de parada que no haya ninguna mejora en ninguno de los individuos de la población (no en el mejor) en 10 iteraciones.

Cuando el algoritmo ha optimizado las particiones se realiza un algoritmo de podado de intervalos (pseudocódigo 4). Todos aquellos intervalos que no cubren ningún caso en las reglas del actual clasificador son fusionados. Asimismo es común encontrar en una regla varias particiones contiguas. El algoritmo de podado unifica estos intervalos si no hay perdida de precisión. Normalmente no es factible realizar esta operación dado que alguno de los valores de los atributos puede estar siendo usado por otra regla.

Este proceso se realiza regla a regla empezando por el orden de clasificación, ya que a medida que vamos procesando reglas los ejemplos cubiertos son eliminados de la base de datos y esta es cada vez más reducida. Cuando todas las reglas que quedan son de la misma clase se eliminan las reglas restantes y se sustituye por una regla de tipo *ELSE → Clase*.

```
Para cada Regla del Clasificador
    Para Attr i = atributo de la Regla
        Si Attr punto inicial = punto final →
                ClasificadorActual →Fusionar Regla(Attr i, Attr i+1)
        Si Attr_i y Attr_i+1 Son contiguos
            Si Evaluar( ClasificadorFusionado Regla(Attr_i, Attr_i+1) ) ==
                ClasificadorActual →Fusionar Regla(Attr i, Attr i+1)
```

Pseudocódigo 5. Esquema de eliminación de particiones

4.2.3.2 Coevolución de intervalos y reglas

Hemos denominado a este ajuste intermedio de las reglas *coevolución*, porque hay dos cromosomas que interactúan en paralelo en dos algoritmos genéticos que van alternándose en el control cooperando para la mejora de la solución global.

Como hemos comentado la principal adaptación para datos continuos consiste en la representación relativa sobre una plantilla que tiene la traducción de genotipo/fenotipo, representando los tipos de atributos (nominales/continuos), número de valores y valores numéricos de los intervalos.

Esta plantilla es el cromosoma que se utilizará alternativamente en un AG global cada vez que se genere un nuevo clasificador. El AG global CHC optimiza las particiones y evalúa el clasificador para pasar de nuevo el control a los nodos genéticos con una nueva plantilla ligeramente modificada. El proceso seguido se muestra en el pseudocódigo 5.

Se ha utilizado una variante del algoritmo CHC, que realiza una coevolución de los intervalos evaluando la clasificación con las reglas del clasificador actual. Se establece que el algoritmo se ejecute un número de veces fijo. Por un lado, este criterio, evita que el tiempo de ejecución sea excesivo, ya que en la estratégica de coevolución el algoritmo CHC se ejecutará múltiples veces. Por otro lado, un número elevado de iteraciones puede producir un sobre-aprendizaje, amplificado por el hecho de que al principio de la coevolución las reglas serán de baja calidad

```
Mientras no se cumpla condición de parada
        Los AGL generan nuevas reglas
        Los AGL proporcionan reglas al pool
        Pool genera un nuevo clasificador con esas reglas mediante un greedy
        CHC optimiza las particiones con el clasificador actual
        CHC aplica nueva partición a todos los AGL
```

Pseudocódigo 6. Esquema de coevolución

Mientras se ejecuta la optimización de la plantilla de particiones los AGL están trabajando en paralelo con la anterior versión de la misma y sólo se consolidará la nueva plantilla en nodos si la optimización CHC mejora la clasificación (calculada con GM) sobre el conjunto de datos de entrenamiento global.

La última ejecución de aprendizaje de las particiones realizará una eliminación de intervalos no utilizados en las particiones siguiendo la misma lógica implementada en la opción de optimización final (pseudocódigo 4).

4.2.4 Algoritmo CHC

La creación inicial de los puntos de corte se realiza con un discretizador, se ha utilizado Chi2Merge.

CHC [Eshelman 1991] es un tipo de AG que busca un equilibrio entre la exploración del espacio de búsqueda (realizar una búsqueda en amplitud para localizar zonas prometedoras) y la explotación (búsqueda en profundidad en esas zonas concretas para encontrar la mejor solución). Este algoritmo utiliza un sistema de reinicialización cuando los individuos presentes en la población son muy parecidos entre sí. CHC genera una nueva población manteniendo el mejor individuo encontrado hasta el momento. Este algoritmo utiliza un operador de cruce (BLX-Alpha [Eshelman 1993]) que proporciona una mayor diversidad, y en

los casos de convergencia prematura. El método de reinicialización permite salir de un posible mínimo local.

En la implementación presente trabajo, cada individuo de la población es un conjunto completo de particiones de todos los atributos presentes en la ejecución del algoritmo codificado como los puntos de corte de cada intervalo.

La función de evaluación de cada individuo es el resultado de evaluar las reglas ya generadas por el algoritmo con las particiones indicadas en dicho individuo. Las reglas se siembran inicialmente con el cromosoma con el que se han aprendido las reglas, por lo que el algoritmo producirá una población derivada de él con pequeñas diferencias en las particiones de cada atributo.

El valor mínimo y máximo posible de cada atributo no cambia nunca, por lo que el primer valor de la primera partición de un atributo siempre comenzará en el mínimo. De igual forma, el límite de la última partición será el valor máximo de dicho atributo.

Para poder cruzar dos padres, se define antes una distancia de Hamming que servirá para asegurar que la diferencia entre los padres (prevención de incesto) cumple el umbral establecido por el algoritmo (Umbral = (#Genes · BITSGEN)/4.0). Los cromosomas se pasan a una cadena de bits en codificación gray para estimar la distancia, ya que el CHC original utiliza una codificación binaria.

4.2.4.1 Operadores de cruce

Se han considerado dos posibles operadores de cruce BLX [Eshelman 1993] y PCBLX [Herrera 2003]. Como se puede ver en la figura 4.3, la diferencia entre ellos radica en la interpretación del parámetro alpha.

BLX-ALPHA: este operador de cruce genera un hijo eligiendo un valor aleatorio entre los valores de los padres, ampliando el rango con un porcentaje por debajo del menor valor y por encima del valor mayor. Asumiendo la misma notación matemática anterior, su formulación es esta:

- O = (o1 … on), donde oi es aleatoriamente generado (de forma uniforme) en el intervalo [li, ui], con li = mini – Ii ·α, ui = maxi + Ii ·α, Ii =| xi−yi |, mini = min{xi,,yi} y maxi = max{xi,yi}

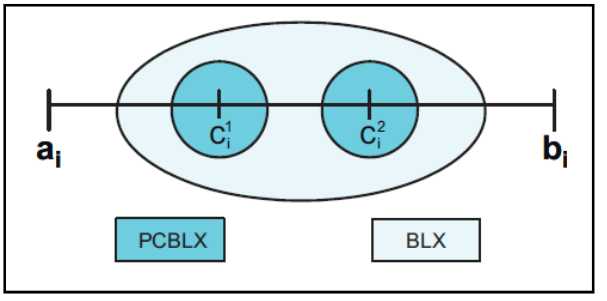

Figura 4.3 Operadores de cruce BLX-ALPHA y PC-BLX

Parent Centric BLX (PCBLX) intenta evitar que los descendientes sean demasiado diferentes a los padres. En determinados contextos cambios demasiado bruscos producen una búsqueda casi aleatoria. Cuando los padres tienen valores muy separados BLX-ALPHA puede producir descendientes con fenotipos muy diferentes a los valores de los padres. PCBLX generar el valor del hijo en un rango cercano al valor de uno u otro padre:

Se asume que X = (x1… xn) e Y = (y1…yn), son los dos cromosomas que queremos cruzar, con xi, yi∈ [ai, bi]⊂ R y i=1…n. El operador PCBLX genera dos hijos de la siguiente forma:

O1 = (o11 … o1n), donde o1i es aleatoriamente generado (de forma uniforme) en el intervalo [li, ui], con li = max{ai, xi−Ii}, ui = min{bi, xi+Ii}, e Ii =| xi−yi | ·α.

O2 = (o21 … o2n), donde o2i es aleatoriamente generado (de forma uniforme) en el intervalo [li, ui], con li = max{ai, yi−Ii} e u2i = min{bi, yi + Ii}, e Ii =| xi−yi | ·α.

4.2.4.2 Población inicial y reinicio

Un aspecto distintivo de CHC es el control del equilibrio exploración/explotación mediante generación de la población que se genera cuando se produce un reinicio. En este sentido se han considerado dos posibilidades. La primera de ellas, consistente en generar un conjunto de individuos aleatorios (válidos dentro de la amplitud de las variables consideradas) con la única consideración de conservar el mejor individuo encontrado hasta la fecha (si es un reinicio) o de conservar el individuo base recibido (si es la población inicial).

La otra opción es generar el resto de los individuos como una variación del individuo base o del mejor individuo encontrado hasta la fecha. Se asume que X = (x1… xn) es el cromosoma base del que se parte, con xi ∈ [ai]⊂ R y i=1…n. El nuevo individuo se genera:

$O = (o_1 \ldots o_n)$,

$o_i = x_1 + \text{Random}\{0, \alpha\}$ ó $o_i = x_1 - \text{Random}\{0, \alpha\}$…

4.2.4.3 Selección de los operadores

Según la literatura existente sobre el tipo de cruce [Herrera 2003] el cruce PCBLX da mejores resultados que el BLX-ALPHA. En el contexto de la optimización de particiones, PCBLX se adapta mejor, ya que el objetivo del *tuning* es mover en pequeñas proporciones los valores de las etiquetas ya consideradas en el algoritmo principal. BLX-ALPHA aplicado en este contexto puede hacer que las reglas que se basan en las nuevas particiones generadas dejen de tener validez.

En cuanto a la generación de individuos en el reinicio y en la población inicial, se ha elegido la generación aleatoria de individuos frente a la variación del mejor, dado que se necesita mantener la diversidad en la población. Al utilizar PCBLX la variación producida por el operador de cruce es inferior. y no existe un operador de mutación que genere diversidad.

4.3 Estudio Experimental

En esta Sección se describe el estudio experimental del modelo adaptado al tratamiento de atributos numéricos de carácter continuo sobre un conjunto de datos no balanceados .

En este estudio experimental se realiza la comparación con algoritmos de clasificación supervisados, distribuidos y no distribuidos con el modelo distribuido EDGAR para datos no balanceados. Se pretende comparar la efectividad del algoritmo modificado con medidas de rebalanceo internas más aprendizaje de los puntos de corte de las particiones frente al preprocesamiento previo de los datos para balancear las clases. Los algoritmos de referencia serán comparados con preprocesamiento para mostrar el comportamiento del algoritmo modificado frente a los mejores resultados obtenidos en el capitulo anterior.

Este estudio utiliza el mismo marco experimental del Capítulo 3 sobre aprendizaje para clases no balanceadas tanto en los algoritmos, los conjuntos de datos como en la metodología de comparación.

Esta Sección se compone de 2 en las que se analiza en primer lugar la elección del modelo de aprendizaje de particiones mediante un estudio del rendimiento de ambas alternativa y una segunda parte donde se realiza propiamente el procedimiento de validación con el conjunto de algoritmos de comparación y los conjuntos de datos no balanceados utilizados en el Capítulo 3.

4.3.1 Coevolución- Mejora final y EDGAR_NB

Esta subsección se dedica a analizar el rendimiento sobre la mejora en la clasificación del algoritmo intervalar con las versiones de aprendizaje final o coevolución de reglas e intervalos. Para ello se han ejecutado ambas versiones del algoritmo con los conjuntos de datos no balanceados.

Las precisiones mostradas están calculadas como la media geométrica de los porcentajes de clasificación de las dos clases.

Tabla 4.2 Test de Friedman 1xN Coevolución

Algoritmo	Ranking
EDGAR_NBC COEVOLUCIÓN	2.4215
EDGAR NB	2.8846
EDGAR_NBC -FINAL	3.9423

Según los resultados del test de Friedman 1xN de la tabla 4.2, l algoritmo ganador es EDGAR_NBC con coevolución. El parámetro p obtenido en esta tabla es 0.012093, con lo que al ser menor que 0.05 es posible hacer un tratamiento post-hoc con el método de Finner.

Tabla 4.3 p-values ajustados Mejora Final- Coevolución

Algoritmos	z=(R0-Ri)/SE	p	Finn
EDGAR_NBC-FINAL	3.376673	0.000734	0.012
EDGAR_NB	0.964764	0.334663	0.037
EDGAR_NBC-COEVOLUCION	0.833205	0.404729	0.05

La tabla 4.3 muestra la clasificación post hoc. Finner rechaza la Ho para p>= 0.005 .. Se puede asegurar que el algoritmo intervalar con coevolución es mejor que el algoritmo EDGAR_NB. Asimismo, según la clasificación propuesta por Friedman, también mejora al algoritmo EDGAR_NBC con mejora final, si bien es verdad que esta vez no de manera estadísticamente significativa según los resultados del método de Finner.

Por lo tanto, se puede concluir que EDGAR_NBC con coevolución de reglas supera en precisión a EDGAR_NB, siendo esta la opción elegida para la comparativa con los algoritmos de comparación.

4.3.2 Comparación con Algoritmos de Referencia

La tabla 4.3 muestra la ejecución de los algoritmos de referencia ya mostrados en el Capítulo 3 en el ámbito de los conjuntos de datos no balanceados de la

subsección 3.3.3. Las tres primeras columnas muestran las tres versiones del algoritmo, EDGAR, Edgar para Datos No balanceados y EDGAR para Datos No balanceados y atributos Continuos.

Las versiones de EDGAR con SMOTE que se han incluido en la comparación pertenecen a la tabla 3.6. Respecto del rendimiento comparado de EDGARNBC, podernos observar en la tabla 4.5 que el ranking muestra un comportamiento similar a los algoritmos Ripper+Smote, Gassist+Smote y UCS con SMOTE. En la tabla 3.7 se confirma que no existe diferencia estadística con estos algoritmos. Respecto del resto de algoritmo podemos afirmar que EDGARNBC es superior estadísticamente, incluyendo la versión del algoritmo sin tratamiento de datos continuos EDGAR_NB.

C. Datos	EDGARNBC	REGAL	REGALTC	Gassist	ObliqueDT	SIA	UCS	OCEC	COGIN	C45	Ripper
pima	0.69067	0.663	**0.651**	0.727	0.654	0.702	0.721	0.712	0.610	0.713	0.68534
glass0	0.772219	0.732	**0.741**	0.820	0.774	0.789	0.635	0.800	0.720	0.754	0.73113
yeast1	0.71123	0.647	**0.417**	0.713	0.657	0.704	0.709	0.651	0.600	0.699	0.68156
vehicle1	0.69	0.674	**0.551**	0.749	0.680	0.651	0.728	0.685	0.640	0.696	0.7133
vehicle3	0.72	0.667	0.587	0.731	0.675	0.650	0.732	0.697	0.620	**0.679**	0.73196
vehicle0	0.902	0.912	0.907	0.932	0.923	**0.818**	0.949	0.911	0.900	0.924	0.93055
yeast3	0.88	0.811	0.851	0.918	0.848	0.865	0.887	0.864	**0.840**	0.922	0.89936
ecoli3	0.820122	0.856	0.830	0.850	**0.724**	0.859	0.842	0.823	0.700	0.810	0.84169
ecoli-0-3-4vs5	0.863908	0.912	0.904	0.880	0.868	0.832	0.882	0.869	0.860	0.857	0.86288
ecoli-0-6-7vs3-5	0.917498	0.707	0.697	0.759	0.764	0.758	0.785	0.769	0.700	**0.753**	0.80203
yeast-0-2-5-7-9	0.89	0.767	**0.850**	0.922	0.861	0.864	0.892	0.876	0.870	0.897	0.88351
ecoli-0-4-6vs5	0.869492	0.831	0.896	0.824	0.830	0.809	**0.870**	0.909	0.830	0.866	0.87069
vowel0	0.9037	0.916	0.898	0.950	**0.965**	0.996	0.964	0.919	0.900	0.947	0.93757
ecoli-0-1-3-7vs2-	0.74	0.532	0.505	0.670	0.610	0.556	0.708	0.712	**0.530**	0.712	0.70717
page-blocks-1-3v	0.942412	0.810	0.924	0.978	0.990	0.884	0.982	0.949	**0.930**	0.994	0.98943
Page-blocks0	0.91	0.934	0.829	0.888	0.921	0.829	0.920	0.936	**0.940**	0.946	0.94431
abalone9-18	0.697342	0.568	0.561	0.681	**0.566**	0.684	0.656	0.558	0.450	0.591	0.66657
glass-0-1-6vs5	0.842293	0.844	0.826	0.766	0.797	0.835	0.799	0.886	0.890	**0.721**	0.92456
glass5	0.83	0.869	0.947	0.744	0.852	0.701	0.900	0.884	0.890	**0.872**	0.96613
yeast-2vs8	0.7458	0.767	0.782	0.775	0.708	0.524	**0.726**	0.788	0.690	0.750	0.75614
glass4	0.8732	0.862	0.862	0.862	0.847	0.892	0.788	0.852	0.860	0.878	0.85093
yeast4	0.705175	0.735	**0.706**	0.770	0.567	0.682	0.723	0.642	0.640	0.664	0.72931
ecoli-0-1-4-7vs5-6	0.879	0.875	0.864	0.856	0.830	**0.824**	0.885	0.813	0.760	0.880	0.8469

Tabla 4.5 Ranking Friedman vs. EDGARNBC

Algorithm	Ranking
EDGAR	15.6087
EDGARNB	10.3043
EDGARNBC	5.9565
EDGAR_S	14.9565
ED-NB_S	9.913
ED-NBC_S	8.2174
REGAL_S	8.7609
REGALTC_S	9.7174
Gassist	11.3478
Gassist_S	5.1739
ObliqueDT_S	9.1304
SIA_S	9.3478
UCS_S	5.2609
OCEC_S	6.4783
COGIN_S	11.6957
C45_S	5.9565
Ripper_S	5.1739

Respecto de la comparación de los algoritmos EDGAR en sus tres versiones, podemos afirmar que el preprocesamiento con SMOTE en cualquiera de sus versiones no mejora el comportamiento de clasificación sobre datos no balanceados.

Respecto de las mejoras de una versión respecto a las otras, en el capitulo anterior describíamos la mejora en casi todos los datasets de la versión para datos no balanceados respecto de la versión original. En la figura 4.4 se puede observar que la versión EDGARNBC, para datos no balanceados con aprendizaje de particiones mejora significativamente a ambas versiones. En el gráfico radial se aprecia como la mejora respecto de la versión sin tratamiento de datos continuos

es mínima, pero lo suficiente como para competir con Gassist+Smote y entrar en el ranking de los 5 algoritmos mejores de la comparativa sin diferencias estadísticas.

Tabla 4.6 Friedman p-values vs. EDGARNBC

i	algoritmo		*p*	Finner
16	EDGAR	7.007486	0	0.003201
15	EDGAR_S	6.569518	0	0.006391
14	COGIN_S	4.379679	0.000012	0.009571
13	Gassist	4.146096	0.000034	0.012741
12	EDGARNB	3.445347	0.00057	0.015901
11	EDGARNB_S	3.182567	0.00146	0.019051
10	REGALTC_S	3.051176	0.002279	0.022191
9	SIA_S	2.802994	0.005063	0.025321
8	ObliqueDT_S	2.657005	0.007884	0.02844
7	REGAL_S	2.408823	0.016004	0.03155
6	EDGARNBC_S	2.04385	0.040968	0.03465
5	**OCEC_S**	**0.875936**	**0.381065**	**0.037739**
4	**EDGAR-NBC**	**0.525561**	**0.599193**	**0.040819**
3	**C45_S**	**0.525561**	**0.599193**	**0.043889**
2	**UCS_S**	**0.058396**	**0.953433**	**0.04695**
1	**Ripper_S**	**0**	**1**	**0.05**

Algoritmo de Referencia Gassist_S

Se rechaza Ho para p-value ≤0.03465

En cuanto al comportamiento del preprocesamiento sobre la versión con tratamiento continuo en EDGAR, se puede ver como en la figura 4.5 la versión con tratamiento continuo y preprocesamiento tiene un rendimiento inferior aunque muy parecido al de la versión sin preprocesamiento. Parece un comportamiento ligado a la estructura interna de los datos, ya que hay 4 conjuntos de datos que tienen un rendimiento muy inferior a partir del desbalanceo medio.

Relativo al comportamiento respecto del algoritmo de referencia se puede observar como Gassist gana en precisión en la gran mayoría de los datasets de bajo nivel de balanceo, la parte central está equilibrada entre los dos y a partir de un Ir de 9, EDGARNBC gana en todas las comparativas a excepción de dos. Confirmando el buen comportamiento del algoritmo para niveles altos de IR sin necesidad de preprocesamiento.

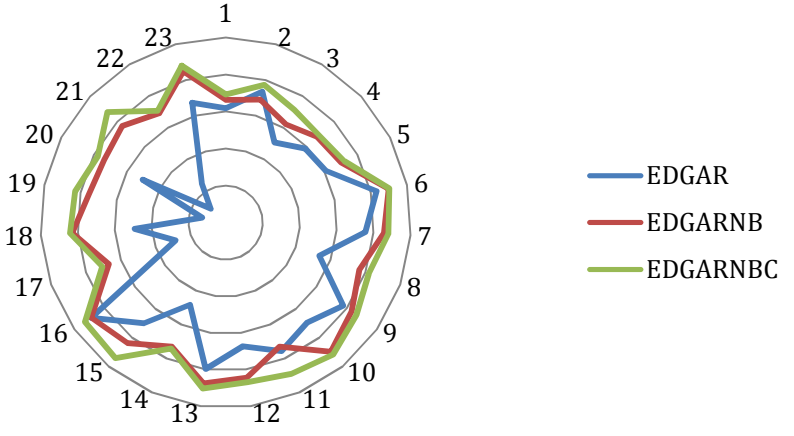

Figura 4.4 Comparación precisión EDGAR vs EDGARNB vs EDGARBNC

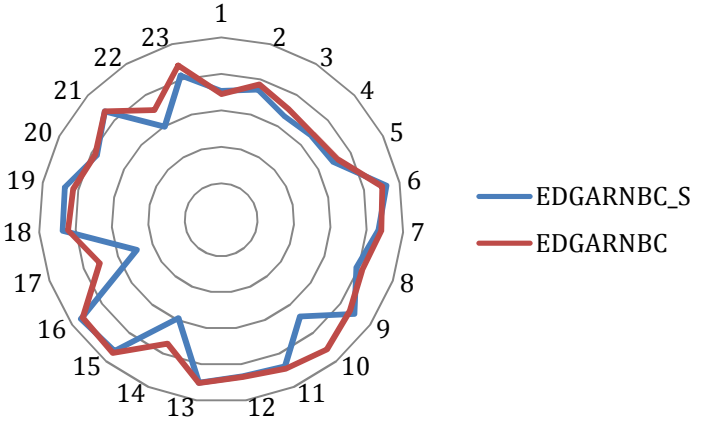

Figura 4.5 Comparación precisión EDGARNBC+ SMOTE vs EDGARBNC

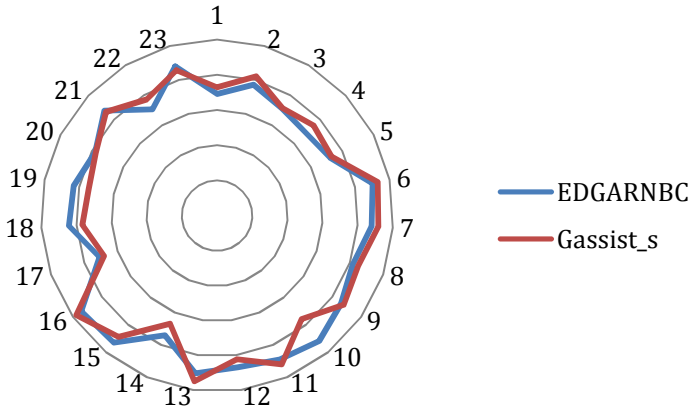

Figura 4.6 Comparación precisión EDGARNBC vs Gassist + SMOTE

Como conclusión, podemos afirmar que el aprendizaje de las particiones por parte del algoritmo permite ganar unas centésimas a la versión de EDGARNB que lo hacen estadísticamente mejor que éste y competitivo frente a los mejores algoritmos de clasificación con preprocesamiento.

Capitulo 5

Comentarios Finales

Dedicaremos esta Sección a la presentación de un resumen de los resultados obtenidos y conclusiones que esta memoria puede aportar. Presentaremos las publicaciones asociadas a esta tesis y comentaremos algunos aspectos sobre trabajos futuros que siguen la línea aquí expuesta y sobre otras líneas de investigación que se pueden derivar

5.1 Conclusiones

Uno de los puntos de interés principales en la actualidad en el ámbito del KDD es afrontar grandes volúmenes de información, dado que los resultados que se obtienen con las técnicas vigentes en bases de datos de pequeña o media dimensionalidad obtienen buenos resultados, pero cuando crece el volumen de los datos se vuelven ineficientes y los indicadores de calidad en su desempeño descienden notablemente.

Este trabajo afronta este problema utilizando computación evolutiva distribuida, proponiéndose un algoritmo genético distribuido para la extracción de reglas de clasificación apto para los citados espacios de búsqueda complejos por su dimensionalidad. Se ha obtenido un modelo computacional que empíricamente, presenta resultados similares a los algoritmos mejor considerados cuando el volumen de información es reducido, y los mejora, inclusive a otras propuestas evolutivas distribuidas, cuando la cantidad de datos es elevada, presentando además mejor rendimiento.

En el estudio descrito en este trabajo y que ha conducido a la propuesta de este modelo, se justifica el uso de algoritmos genéticos ya que éstos son capaces de integrar soluciones parciales basados en conjuntos de datos relacionados y aplicarlos al conjunto de datos local de forma natural, generando nuevas reglas de alto valor.

Respecto del tratamiento de clases no balanceadas, realizamos una propuesta novedosa al utilizar la distribución de los datos en los diferentes nodos como un sistema de rebalanceo que evita los inconvenientes de generación de ruido o solapamiento de clases asociados al sobre-muestreo. Por otro lado para conseguir cotas de competitividad, se han propuesto medidas de coste y de tratamiento de datos continuos. El conjunto de estas medidas ha permitido un algoritmo que está al nivel de los algoritmos clásicos y supera a los algoritmos genéticos distribuidos en el tratamiento de clases no balanceadas incluso con el preprocesamiento de los datos para producir un rebalanceo de las clases.

Para alcanzar una mayor escalabilidad se ha analizado las posibilidades de desarrollo futuro en el paradigma MapReduce. Este modelo aumentará en varios márgenes de magnitud la cantidad de datos que puede tratar el algoritmo sin asumir la limitación de tratar en memoria el conjunto total de datos.

5.2 Trabajos y líneas futuras

En lo relativo a la continuación de este trabajo, proponemos las siguientes líneas:

- Aprendizaje dinámico del número de intervalos
- Mejora de la escalabilidad con una implementación sobre modelo MapReduce.

Describimos a continuación las características de estas líneas y el impacto sobre el algoritmo propuesto.

5.3 Aprendizaje Dinámico del Número de Particiones

En el modelo propuesto el número de particiones de cada atributo es fijo en el cromosoma. El algoritmo de aprendizaje de las particiones CHC aprende los límites de las particiones y durante el proceso va a generar particiones de escaso valor (con muy pocos ejemplos cubiertos). Estos intervalos son mejorados en el proceso de eliminación de intervalos final.

Sin embargo es conveniente integrar este proceso de eliminación en el aprendizaje de las particiones ya que la función de evaluación del algoritmo de aprendizaje de reglas tiene en cuenta la longitud de la regla y la existencia de más particiones de las necesarias no permite favorecer las reglas que una vez procesadas tendrían una expresión más compacta.

5.4 Adaptación a Modelo MapReduce

En este apartado se propone la utilización del modelo MapReduce para eliminar la limitación del espacio de memoria en la validación del clasificador final y mejorar de esta forma la escalabilidad del algoritmo.

5.4.1 Introducción

En los últimos años la aparición de nuevas técnicas de almacenamiento distribuido ha proporcionado nuevas formas de desarrollar algoritmos paralelos y distribuidos que está permitiendo disminuir el tiempo necesario para tratar millones de datos con maquinas de bajo coste. Este paradigma está basado principalmente en las funciones MapReduce [Dean 2008] y el sistema de almacenamiento distribuido HDSF [White 2009].

En este sentido, aunque se han conseguido desarrollar versiones adaptadas a este modelo distribuido de buena parte de los algoritmos de aprendizaje a partir de datos, algunos de los modelos de aprendizaje, como por ejemplo los que utilizan algoritmos genéticos para la extracción de reglas, presentan ciertas dificultades para su implementación debido a la necesidad de acceder repetidamente al conjunto completo de datos para calcular la bondad de las reglas o conjuntos de

ellas en el alto número de iteraciones que éste tipo de algoritmos realiza durante su ejecución. El modelo MapReduce permite mejoras en éste ámbito pero no llega a hacer que la programación sea equivalente y es necesario por tanto establecer estrategias de implementación que conserven la calidad de los resultados y mantengan el tiempo de ejecución en niveles competitivos respecto a otras alternativas distribuidas.

Como trabajo futuro se proponen varias estrategias de aplicación de este modelo a algoritmos genéticos distribuidos para extracción de reglas de clasificación. El modelo resultante permitiría incrementar la escalabilidad con MapReduce y por otro lado mantener la calidad de la solución final mediante el uso de conjuntos de entrenamiento distribuidos obtenidos al aplicar diferentes estrategias de reducción de datos.

Si bien los algoritmos genéticos distribuidos para clasificación tienen en su esencia uno de los elementos apropiados para afrontar conjuntos de datos grandes en base a su escalabilidad, cuando los conjuntos de datos son del orden de magnitud de los considerados en Big Data, encuentran dos tipos de dificultades:

Suelen necesitar tener el conjunto completo de datos a tratar en la memoria principal (en el nodo principal o asignados a alguna regla de amplia cobertura), y por tanto, cuando el conjunto excede a ésta, no son aplicables.

En muchos casos también se suele observar a partir de cierto tamaño un deterioro progresivo de la calidad de los resultados proveniente probablemente del excesivo particionamiento del conjunto de datos en las fases iniciales de aprendizaje.

Estas limitaciones nos invitan a considerar modelos híbridos con MapReduce, que se discutirán en la Subsección 5.4.4.

5.4.2 MapReduce

El modelo MapReduce permite un modelo de diseño paralelo basado en la división de datos. Este modelo se basa en la existencia de conjunto de datos distribuidos que permiten una alta escalabilidad de datos y la codificación de

tareas sobre estos basados fundamentalmente en dos operaciones Map y Reduce. Estas dos operaciones ocultan la complejidad del hardware paralelo y permite simplificar enormemente el diseño de algoritmos paralelos.

Aunque inicialmente este modelo fue pensado para procesar grandes cantidades de datos distribuidas (Map) que luego tienen una agregación posterior para producir unos resultados (Reduce), la simplicidad de las operaciones permite utilizar este modelo de diseño incluso cuando no hay grandes volúmenes de datos, permitiendo estructurar algoritmos paralelos que se comunican a través de la agrupación de valores entorno a nuevos conjuntos de claves.

MapReduce no es siempre una alternativa a un desarrollo paralelo dado que hay que reformular el algoritmo en términos de estas dos operaciones y que la relación entre ellas no genere excesivos accesos a disco o malas configuraciones de comunicación que generen cuellos de botella.

Los algoritmos genéticos distribuidos para extracción de reglas permiten escalar el conjunto de datos de entrenamiento manteniendo el equilibrio entre exploración y calidad de la solución. Sin embargo cuando el conjunto de datos sobrepasa ciertos límites el modelo puede presentar problemas de rendimiento por los límites de memoria de los procesadores o bien de calidad de la solución por el excesivo particionamiento de los datos en procesos. En este apartado vamos a analizar brevemente el impacto del modelo MapReduce sobre la evaluación del fitness en un algoritmo genético Secuencial. Posteriormente propondremos alternativas para la mejora de las algunas de las propuestas existentes de algoritmos genéticos distribuidos para la clasificación que permitirían mantener las características del modelo distribuido con cualquier volumen de datos.

5.4.3 Modelo Secuencial Paralelizado sobre MapReduce

Como se ha comentado anteriormente, los algoritmos genéticos para la extracción de reglas son muy sensibles al tamaño del conjunto de entrenamiento, tanto en rendimiento como en capacidad de aprendizaje. La principal estrategia de paralelización de un algoritmo genético manteniendo la estructura secuencial del mismo es la de la paralelización del cálculo del fitness del individuo. Al aplicar esta estrategia a un algoritmo de aprendizaje de reglas en MapReduce el algoritmo

se beneficiaría de no tener un límite de datos determinado, dado que la operación de evaluación de una regla se puede realizar en un solo paso MapReduce.

Sin embargo cada operación Map tiene asociado un tiempo de latencia relativamente alto en el entorno de los milisegundos debido a la comunicación por red más el acceso al medio físico si accede a la base de datos. En este caso el tiempo de ejecución del algoritmo es relativo al número de iteraciones (k) y a la latencia de acceso al medio físico más la comunicación de red (l). Aún cuando el rendimiento pueda ser inferior al del modelo secuencial en un procesador, hay otros motivos por los que puede ser conveniente utilizar MapReduce:

A medida que la magnitud de datos de entrenamiento crece es más difícil almacenarlo de manera integra en un sólo sistema.

El coste económico de instanciar un conjunto de servidores virtuales que soporten una ejecución sobre el paradigma Hadoop es muy inferior al de una maquina (o servidor virtual) con la memoria equivalente para contener el conjunto total de los datos.

La evaluación de cada regla contra un conjunto elevado de elementos en un solo sistema requiere un algoritmo de orden n (cada elemento por las n reglas del conjunto de individuos que conforman la población del genético). Aún siendo orden n, el tiempo crecerá de forma pareja al aumento de los datos mientras que en MapReduce el tiempo se mantiene constante. En la formula (1), se expresa la relación entre el coste de mantener el conjunto de datos de manera compacta en un servidor o tener el *fitness* distribuido sobre MapReduce) siendo c coste en memoria principal en milisegundos, n número de datos y l la latencia:

$$c*n > l \text{ por tanto } n > l/c \qquad (5.1)$$

A partir de este valor n de datos, el sistema sobre MapReduce ofrecerá mejoras en el rendimiento.

Otra estrategia para disminuir el coste del aprendizaje asociado a la cardinalidad es la reducción de datos (discretización, selección de características, selección de instancias, etc...) para que el conjunto de datos resultante pueda ser tratado directamente en memoria principal.

La selección de instancias y la discretización sin repetición de elementos, tienen el efecto de representar en un solo ejemplo a varios elementos del conjunto original. Generalmente en el preprocesado de datos no se añade información en forma de metadatos para ayudar al proceso de aprendizaje a discernir qué datos son más representativos. Nosotros proponemos añadir metadatos a los datos de entrenamiento reducidos para ajustar el valor de clasificación de las reglas. El algoritmo puede entonces calcular la función de evaluación de un elemento teniendo en cuenta el número real de instancias representadas.

Fitness =casosTotales* e^(1/casos negativos*) (5.2)

En (5.2) se muestra una posible representación de *fitness* de una regla, donde el número casos totales y casos negativos, son un sumatorio ponderado de los casos reales a los que cubre y los casos negativos en los que la regla no predice la clase correcta. Esta fórmula permite mantener el coste de la clasificación errónea respecto a los datos iniciales representados manteniendo un conjunto reducido de datos. Cuando se trabaja con clases no balanceadas, se puede incluir un peso de fallo en la clasificación de las clases que represente la proporción real entre las reglas. Durante el proceso de disminución del modelo este peso puede ser calculado para formar parte de la ponderación de los casos representados respecto a los originales.

5.4.4 Algoritmos Genéticos Distribuidos de Clasificación

Los algoritmos distribuidos tienen como principal característica que la evaluación de los individuos y los datos asociados a ellos se realiza en procesadores paralelos. Este apartado recoge las posibles estrategias de implementación sobre las principales referencias de algoritmos distribuidos de aprendizaje de reglas en base a las operaciones básicas Map y Reduce.

Como se ha comentado anteriormente, algunas de las principales referencias de algoritmos genéticos distribuidos para la clasificación [Giordana 1994] [López 2011] [Anglano 1994] son implementables en MapReduce de manera equivalente. El modelo MPI de sincronismo puede ser implementado con Nodos Reduce y las

evaluaciones de los individuos con Map. Sin embargo ello conlleva una pérdida de rendimiento asociada al acceso físico para la evaluación del *fitness* de una regla, que según (7.1), tendrá una cota superior de datos a partir de la cual va a tener un comportamiento constante con el aumento de los datos a tratar.

EDGAR separa el proceso de aprendizaje en los nodos de la tarea de validación de las regla en un supervisor con el total de los datos de entrenamiento. Esta implementación podría realizarse en MapReduce sin pérdida de rendimiento respecto al modelo original en memoria principal, realizando la operación de validación global mientras que los nodos están realizando el aprendizaje de nuevas reglas, ya que éstos no necesitan esperar a otros nodos o al procesador maestro para generar reglas o compartir datos y reglas con nodos.

Por otro lado la reducción del conjunto de datos inicial podría permitir a estos algoritmos utilizar los nodos de aprendizaje directamente en memoria principal y utilizar los datos totales en la generación del clasificador final. Si bien esto es posible para todas las referencias, en [Giordana 1994] [López 2011] [Anglano 1994] la asignación de una regla y los datos globales asociados a la misma para su refinamiento no aportaría una gran mejora del rendimiento. Sin embargo EDGAR separa estas dos funciones por lo que planteamos dos alternativas en base a la función realizada por el nodo supervisor y los nodos de aprendizaje con los datos preprocesados.

Por un lado proponemos utilizar en los nodos aprendedores un conjunto de datos reducido (discretización, selección de instancias o características) que envían reglas al supervisor que valida y genera la solución global con MapReduce con el conjunto completo de entrenamiento. Esta validación minimiza la pérdida de calidad producida por el proceso de reducción de datos en los aprendedores.

Por otro lado esta alternativa se puede mejorar utilizando reducción de datos con distintas técnicas (distintos discretizadores, selección de instancias etc.). El clasificador final estará compuesto por la integración de las mejores reglas validadas de forma global minimizando de esta forma la pérdida de calidad que se pueda haber producido en el proceso de una sola técnica de reducción de datos.

5.4.5 Conclusiones

El modelo MapReduce permite abarcar conjuntos de datos que no son tratables directamente en memoria principal, sin embargo no tiene un rendimiento comparable a las implementaciones que trabajan con todos los datos en memoria principal. La hibridación de técnicas de disminución del conjunto de datos permite el tratamiento en memoria, pero a cambio puede afectar a la calidad del modelo generado. Se proponen alternativas basadas en la colaboración de modelos locales de menor magnitud con la validación global usando MapReduce que permitiría mantener los tiempos de ejecución abarcando un conjunto de datos mucho mayor sin que la calidad del modelo se vea afectada

Apéndice A

Tests no Paramétricos para el Análisis Estadístico de los Resultados

En este apéndice se explican con mayor detalle los distintos test estadísticos no paramétricos que se han usado en esta memoria para el análisis de los resultados obtenidos por los distintos métodos en estudio.

A.1. Test de Contraste de Hipótesis no Paramétricos

Un contraste o test de hipótesis es una técnica de inferencia estadística que permite, a partir de los datos obtenidos de una (o varias) muestra observada, decidir si se acepta o no una hipótesis formulada sobre una (o varias) población(es) o algoritmo(s).

Una hipótesis estadística es una asunción relativa a una o varias poblaciones, que puede ser cierta o no. Las hipótesis estadísticas se pueden contrastar con la información extraída de las muestras y se puede cometer un error, tanto si se aceptan como si se rechazan.

La hipótesis que se formula se denomina hipótesis de trabajo o nula y se denota H_0. La hipótesis contraria se denomina hipótesis alternativa, H_1. El test de hipótesis decidirá, basándose en la muestra observada, si se acepta o no la hipótesis nula formulada frente a la hipótesis alternativa.

En un test de hipótesis se pueden cometer dos tipos de errores:

Error tipo I, se rechaza la hipótesis H_0 cuando es cierta.

Error tipo II, se acepta la hipótesis H_0 cuando es falsa.

Sólo se puede cometer uno de los dos tipos de error y, en la mayoría de las situaciones, se desea controlar la probabilidad de cometer un error de tipo I. Se denomina *nivel de significación o confianza* α de un test a la probabilidad de cometer un error tipo I, es decir, la probabilidad de que el estadístico de contraste caiga en la región de rechazo:

$\alpha = P$ (rechazar H_0 | H_0 es cierta)

El nivel de confianza α se ha de decidir de antemano, de forma que se establece la probabilidad máxima que se está dispuesto a asumir de rechazar la hipótesis nula cuando es cierta. Dicho nivel lo elige el usuario. La selección de dicho nivel conduce a dividir en dos regiones el conjunto de posibles valores del estadístico de contraste: la región de rechazo, con probabilidad α, bajo H_0 y la región de aceptación, con probabilidad $1-\alpha$, bajo H_0.

Si el estadístico de contraste calculado por el test (también conocido como p-valor o p-value), toma un valor perteneciente a la región de aceptación (p-valor $\geq \alpha$) entonces no existen evidencias suficientes para rechazar la hipótesis nula con un nivel de significación α y se dice que el contraste estadísticamente no es significativo. Si, por el contrario, el estadístico cae en la región de rechazo (p-valor $< \alpha$) entonces se asume que los datos no son compatibles con la hipótesis nula y se rechaza a un nivel de significación α. En este supuesto se dice que el contraste es *estadísticamente significativo*.

Si p-valor $\geq \alpha$ ==> Se acepta la hipótesis nula H0

Si p-valor $< \alpha$ ==> Se rechaza la hipótesis nula H0

En nuestro caso los test de contraste los vamos a utilizar para comparar los resultados obtenidos por los diferentes algoritmos y ver si existen o no diferencias significativas entre ellos. Es decir:

La hipótesis nula H_0 es que los resultados de los diferentes métodos son similares, mientras que la hipótesis alternativa H_1 significa que los resultados de los diferentes métodos son distintos.

Al aplicar el test, si la hipótesis nula H_0 se rechaza significa que estadísticamente existen diferencias significativas entre los resultados obtenidos por los métodos y por el contrario si se acepta H_0, no existen diferencias significativas entre los resultados, es decir, los métodos obtienen resultados equivalentes y se puede afirmar que son estadísticamente iguales.

Los test estadísticos pueden ser paramétricos y no paramétricos. Para aplicar test paramétricos es necesario, entre otras cosas, conocer la forma de la distribución de donde proceden los datos, ya que si no siguen una distribución normal no se pueden utilizar. Como en la práctica esto es difícil de conocer, una alternativa es el uso de tests no paramétricos o de libre distribución, que no se basan en la hipótesis de que los datos siguen una determinada distribución de probabilidad.

Que las condiciones de aplicación de los tests no paramétricos sean menos restrictivas que la de los test paramétricos, unido al hecho de que la mayoría de las veces sean más fáciles de aplicar, han motivado que nos decantemos por este tipo de test para validar los resultados obtenidos en los experimentos.

A continuación se describe los distintos métodos estadísticos no paramétricos utilizados en esta memoria para el análisis de los resultados obtenidos por los distintos métodos en estudio.

A.2. Test de Friedman

El test de Friedman es un test no paramétrico utilizado para realizar comparaciones múltiples y es equivalente al test paramétrico ANOVA. El test de Friedman se utiliza para determinar si en un conjunto de k muestras, al menos dos de las muestras son suficientemente dispares para ser consideradas significativamente diferentes (se ejecuta por tanto para realizar comparaciones múltiples y determinar si hay diferencias entre las muestras).

El test de Friedman trabaja de la siguiente forma: Calcula un ranking de los resultados obtenidos por cada algoritmo (r_j para cada algoritmo j, habiendo k algoritmos) sobre cada uno de los conjunto de datos, y le asigna al mejor el ranking 1 y al peor el ranking k. Si se produce un empate entre dos algoritmos, el test asigna el ranking medio a ambos. Bajo la hipótesis nula, este test indica que todos los algoritmos son equivalentes (y por tanto sus rankings son similares). Por lo tanto, el rechazo de dicha hipótesis implica la existencia de diferencias en el rendimiento de todos los algoritmos estudiados.

Supongamos que r^i_j es el ranking del j-ésimo algoritmo (siendo k el número total de algoritmos) sobre el i-ésimo conjunto de datos (con un total de N). El test de Friedman lleva a cabo una comparación del ranking medio de los algoritmos, $R_j = 1/N \sum_i r_i^j$. Bajo la hipótesis nula, que como hemos dicho anteriormente afirma que todos los algoritmos son equivalentes y por lo tanto sus rankings R_j también deberían de ser igual, el estadístico de Friedman

$$X_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right] \qquad \text{(C.1)}$$

Se distribuye de acuerdo con una distribución X^2_F con k-1 grados de libertad, siempre que N y k sean lo suficientemente grandes (como regla general, N > 10 y K > 5).

A.3. Test de Finner

El rechazo de la hipótesis nula en el test de Friedman no implica que existan diferencias entre todos los algoritmos. Sólo indica que existen diferencias entre todas las muestras obtenidas por los algoritmos, pero no indica si dichas diferencias son estadísticamente significativas. Para determinar si los algoritmos son estadísticamente diferentes es necesario compararlos por pares mediante un procedimiento a posteriori (post-hoc). En este caso, se elige uno de los algoritmos (algoritmo de control) y el procedimiento a posterior) comparan el elegido con el resto de los algoritmos.

El propósito de los test post hoc (a posteriori), por tanto, es determinar exactamente cuál de las muestras difieren con las otras en términos de diferencias de medias. Estos test se realiza generalmente después del test de Friedman, si éste indica que las muestras no son idénticas (si el test de Friedman acepta la hipótesis nula, no tiene sentido aplicar los test a posteriori, al no indicar Friedman que hay evidencias de diferencias entre las muestras).

El método de Finner es un test a posteriori para el estadístico de Friedman. Se utiliza cuando el test de Friedman rechaza la hipótesis nula (indica que hay evidencias de diferencias entre las muestras) y queremos corroborar que dichas diferencias son estadísticamente significativas. Este método lleva a cabo un proceso de comparación por pares múltiple que trabaja con un algoritmo de control (normalmente, se elige el mejor de los algoritmos), el cual es comparado con el resto de algoritmos. El estadístico utilizado para comparar el i-ésimo y j-ésimo método es el siguiente:

$$z = (R_i - R_j) \Big/ \sqrt{\frac{k(k+1)}{6N}} \qquad \text{(C.2)}$$

El valor z se utiliza para encontrar la probabilidad correspondiente (p-valor) dentro de la tabla que represente la distribución normal, el cual se compara con un nivel de confianza α apropiado.

El test de Finner es un procedimiento incremental, que testea secuencialmente las hipótesis ordenadas por su valor de significancia. De esta forma, los procedimientos se ordenan secuencialmente por su importancia. Sean p_1, p_2, …, p_{k-1} los p-valores ordenados, de forma que $p_1 \leq p_2 \leq \ldots \leq p_{k-1}$ y sean H_1, H_2, …, H_{k-1} las hipótesis correspondientes. El test de Finner compara cada p_i con $1-(1-\alpha)^{(k-i)/k)}$, comenzando por el que tiene el valor p más significativo. Si p_1 es menor que $1-(1-\alpha)^{(k-1)/k)}$, la correspondiente hipótesis se rechaza y se pasa a comparar p_2 con $1-(1-\alpha)^{(k-2)/k)}$. Si se rechaza la segunda hipótesis, el test continúa con la tercera y así sucesivamente. Cuando una cierta hipótesis nula no se puede rechazar, el test acaba las comprobaciones aceptando la hipótesis nula para el resto de hipótesis que aun quedaban por comprobar.

A.4. Test de Wilcoxon

El test de ranking de signos de Wilcoxon [Wil45] es un test no paramétrico análogo al *t-test*. Por lo tanto, se trata de un test por parejas que tiene por objetivo detectar la existencia de diferencias significativas entre el comportamiento de dos medias muestrales o dos algoritmos (si las medias se refieren a las salidas de dos algoritmos, entonces el test prácticamente evalúa el comportamiento recíproco de los dos algoritmos).

Su funcionamiento se basa en calcular las diferencias entre los resultados de dos algoritmos y calcular un ranking utilizando dicho valor, ignorando signos. En este caso el ranking va desde 1 a N, en vez de hasta k.

Sea d_i la diferencia entre las medidas de rendimiento de los dos algoritmos sobre el i-ésimo conjunto de datos (de un total de N). Se ordenan las diferencias obtenidas en un ranking, según su valor absoluto (en caso de empates se asignan

los ranking medios). Se calcula R^+ como la suma de los rankings de los conjuntos de datos en los que el primer algoritmo supera al segundo, y R^- la suma de los rankings en el caso contrario. Los rankings con valor $d_i = 0$ se reparten de forma equitativa entre las dos sumas anteriores (si hay un número impar de rankings con $d_i = 0$ se ignora uno de ellos).

$$R^+ = \sum_{d_i>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i)$$
$$R^- = \sum_{d_i<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \qquad \text{(C.3)}$$

Sea T el valor menor de ambas sumas, $T = \min(R^+, R^-)$. Si T es menor o igual que el valor de la distribución de T de Wilcoxon para N grados de libertad (Tabla B.12 en [Wil45]), se rechaza la hipótesis nula de igualdad de las medias y el algoritmo asociado al mayor de los valores es el mejor.

El test de ranking de signos de Wilcoxon es más sensible que el t-test. Desde el punto de vista estadístico, el test de Wilcoxon es más seguro, ya que no se asume una distribución normal. Además, los valores atípicos (actuaciones excepcionalmente buenas o malas en unos pocos conjuntos de datos) tienen un efecto menor en el test de Wilcoxon que en el t-test. El test de Wilcoxon asume diferencias continuas d_i, por lo tanto, no se debería redondear a uno o dos decimales, ya que esto disminuiría el poder del test debido a un alto número de empates.

Apéndice B

Resultados Detallados de Estudio Experimental en Escalabilidad

En este apéndice se muestran con mayor detalle las tablas resultantes de la experimentación correspondientes al estudio de la escalabilidad del sistema. Debido a la extensión de las mismas se ha considerado oportuno separarlas de la subsección de experimentación para mejorar la legibilidad del documento.

5.4.6 B.1 Tablas Detalladas de Precisión y Escalabilidad

En la tabla B.1 se muestran los resultados detallados de ejecución de EDGAR y el algoritmo de comparación REGAL en las configuraciones descritas en la subsección 2.3.5. La tabla está dividida en dos secciones, una para cada algoritmo, donde la primera columna muestra el número de nodos, la segunda el número de reglas medias y la desviación estándar, la tercera la precisión del conjunto de test en media y desviación .La cuarta columna muestra la precisión en entrenamiento en media y desviación estándar. Finalmente la última columna de cada algoritmo representa el tiempo de ejecución en minutos. En las filas cada conjunto de datos agrupa sus resultados en base al número de nodos, incluyendo una fila final por cada conjunto de datos con la media de todas las ejecuciones. Se ha indicado la precisión en porcentaje para facilitar la diferenciación entre medias y desviación estándar en filas alternas.

Tabla B.1 Resultados detallados escalabilidad-precisión

		EDGAR				REGAL			
		Reglas	**Test**	**Train**	**Tiempo**	**Reglas**	**Test**	**Train**	**Tiempo**
Car									
media	4	57	96%	93%	0.09	58	96%	92%	15.8
ds		4.34	0.005	0.009	0.01	3.52	0.007	0.016	7.4
media	8	61	95%	92%	0.05	61	95%	90%	28.6
ds		6.85	0.005	0.010	0.00	5.99	0.006	0.009	20.4
media	16	61	94%	91%	0.08	67	94%	90%	75.8
ds		6.57	0.003	0.012	0.01	4.17	0.005	0.010	45.5
media	32	56	94%	90%	0.22	73	94%	90%	148.6
ds		4.72	0.004	0.011	0.08	7.03	0.006	0.016	87.6
media	Promedio	61	97%	94%	0.10	66	98%	94%	63.4
ds		6.06	0.006	0.011	0.07	8.13	0.006	0.014	71.6
Cleveland									
media	4	69	93%	54%	0.09	54	96%	82%	16.6
ds		7.90	0.054	0.041	0.10	4.04	0.053	0.048	2.6
media	8	65	84%	55%	0.06	55	92%	81%	42.6
ds		22.06	0.116	0.019	0.09	4.66	0.007	0.013	7.5
media	16	70	87%	49%	0.05	62	91%	82%	89.6
ds		7.24	0.029	0.024	0.02	4.24	0.010	0.012	17.6
media	32	78	89%	51%	0.05	68	90%	77%	234.9
ds		7.48	0.026	0.029	0.01	6.41	0.013	0.125	35.0
media	Promedio	73	91%	54%	0.07	62	96%	83%	100.5
ds		14.30	0.070	0.031	0.07	8.33	0.016	0.067	91.6
Credit									
media	4	62	91%	84%	0.08	73	94%	56%	15.2
ds		6.39	0.027	0.030	0.03	5.56	0.059	0.040	2.7
media	8	63	88%	81%	0.04	71	89%	54%	32.5
ds		7.36	0.010	0.021	0.01	5.88	0.022	0.021	11.5
media	16	68	87%	81%	0.03	73	90%	53%	62.3
ds		7.51	0.014	0.023	0.01	7.53	0.025	0.019	27.8
media	32	79	88%	82%	0.04	73	89%	54%	147.4
ds		9.62	0.013	0.024	0.03	6.75	0.028	0.024	56.3
media	Promedio	69	90%	84%	0.05	75	94%	56%	69.4
ds		10.62	0.015	0.024	0.03	6.31	0.028	0.024	63.3

		EDGAR				REGAL			
		Reglas	**Test**	**Train**	**Tiempo**	**Reglas**	**Test**	**Train**	**Tiempo**
Ecoli									
media	4	52	92%	67%	0.04	41	95%	75%	21.2
ds		4.88	0.053	0.051	0.03	5.23	0.056	0.060	5.4
media	8	57	84%	59%	0.04	41	91%	71%	48.2
ds		6.19	0.035	0.042	0.04	4.01	0.016	0.039	16.2
media	16	55	89%	64%	0.03	43	91%	69%	89.3
ds		7.15	0.016	0.055	0.01	4.94	0.019	0.047	53.9
media	32	70	84%	58%	0.06	53	94%	69%	158.0
ds		6.87	0.029	0.065	0.03	6.52	0.019	0.036	97.8
media	Promedio	61	90%	64%	0.04	47	96%	73%	95.5
ds		9.89	0.036	0.059	0.03	7.84	0.022	0.044	86.7
Glass									
media	4	55	96%	47%	0.11	35	97%	56%	18.1
ds		10.22	0.115	0.088	0.09	3.68	0.042	0.075	3.4
media	8	53	85%	44%	0.04	40	93%	50%	42.0
ds		6.21	0.019	0.060	0.02	3.64	0.024	0.048	5.9
media	16	51	77%	37%	0.04	45	93%	55%	100.8
ds		23.40	0.216	0.074	0.02	4.14	0.020	0.051	16.2
media	32	65	85%	41%	0.05	56	93%	53%	250.9
ds		9.63	0.013	0.059	0.03	7.96	0.024	0.065	53.0
media	Promedio	61	92%	46%	0.07	45	96%	55%	106.2
ds		14.24	0.105	0.069	0.03	9.57	0.025	0.062	98.1
Haberman									
media	4	31	83%	69%	0.04	53	91%	72%	15.6
ds		5.73	0.110	0.098	0.01	4.50	0.036	0.032	8.7
media	8	25	74%	62%	0.03	50	88%	65%	37.3
ds		5.33	0.030	0.036	0.01	4.91	0.014	0.026	21.6
media	16	24	73%	62%	0.03	51	89%	69%	122.9
ds		5.14	0.023	0.025	0.04	4.22	0.016	0.021	52.4
media	32	21	54%	46%	0.05	49	91%	66%	228.6
ds		3.61	0.009	0.018	0.09	5.14	0.015	0.015	106.9
media	Promedio	29	82%	69%	0.04	52	92%	69%	125.4
ds		4.84	0.038	0.037	0.03	4.83	0.018	0.029	115.5

		EDGAR				REGAL			
		Reglas	**Test**	**Train**	**Tiempo**	**Reglas**	**Test**	**Train**	**Tiempo**
House-votes									
media	4	24	98%	96%	0.04	24	96%	95%	12.5
ds		4.21	0.115	0.114	0.01	4.21	0.112	0.112	0.6
media	8	21	89%	87%	0.02	21	86%	86%	8.0
ds		3.11	0.006	0.009	0.00	3.11	0.006	0.008	0.3
media	16	23	89%	87%	0.01	23	87%	86%	9.6
ds		3.92	0.006	0.010	0.00	3.92	0.006	0.010	0.4
media	32	30	89%	86%	0.01	30	87%	85%	9.3
ds		4.83	0.005	0.008	0.00	4.83	0.005	0.008	0.7
media	Promedio	26	98%	96%	0.02	26	97%	95%	0.7
ds		5.79	0.026	0.027	0.01	5.79	0.025	0.027	9.5
Hypothyroid									
media	4	170	95%	93%	0.52	158	95%	93%	9.6
ds		28.72	0.146	0.144	0.08	26.63	0.147	0.144	4.0
media	8	173	85%	83%	0.40	147	85%	82%	13.6
ds		11.98	0.002	0.003	0.07	17.50	0.005	0.003	2.8
media	16	197	85%	82%	0.99	150	71%	70%	21.9
ds		13.67	0.003	0.003	0.06	14.11	0.001	0.002	6.4
media	32	180	92%	89%	2.57	180	90%	87%	56.5
ds		6.40	0.001	0.008	0.36	10.71	0.004	0.004	20.5
media	Promedio	200	95%	94%	1.00	178	95%	93%	35.6
ds		28.08	0.038	0.037	0.94	20.42	0.029	0.028	27.1
Iris									
media	4	13	97%	93%	0.02	10	98%	94%	11.8
ds		4.06	0.117	0.103	0.00	2.32	0.035	0.038	5.2
media	8	14	88%	85%	0.01	10	95%	88%	30.2
ds		2.77	0.024	0.061	0.00	2.34	0.013	0.033	11.4
media	16	12	90%	85%	0.01	12	96%	84%	66.1
ds		4.87	0.030	0.086	0.01	2.74	0.008	0.038	29.0
media	32	15	99%	95%	0.01	14	97%	93%	28.8
ds		3.47	0.030	0.086	0.01	3.23	0.064	0.084	12.5
media	Promedio	14	96%	92%	0.01	11	99%	91%	36.2
ds		5.55	0.038	0.075	0.01	2.55	0.017	0.044	9.6

		EDGAR				REGAL			
		Reglas	**Test**	**Train**	**Tiempo**	**Reglas**	**Test**	**Train**	**Tiempo**
Krvskp									
media	4	56	98%	97%	0.15	56	90%	89%	13.3
ds		8.14	0.035	0.034	0.07	8.99	0.004	0.009	4.8
media	8	60	93%	92%	0.07	58	90%	89%	33.9
ds		8.42	0.005	0.008	0.05	6.68	0.003	0.006	11.3
media	16	63	93%	92%	0.08	47	66%	66%	23.0
ds		6.59	0.006	0.009	0.03	1.41	0.004	0.001	1.6
media	32	61	88%	87%	0.07	59	65%	65%	29.0
ds		7.99	0.005	0.011	0.03	1.41	0.004	0.001	2.7
media	Promedio	62	98%	96%	0.10	54	86%	85%	22.5
ds		8.78	0.014	0.015	0.06	10.40	0.092	0.092	11.1
Monk									
media	4	52	99%	79%	0.05	51	82%	64%	5.6
ds		6.40	0.000	0.000	0.02	0.00	0.068	0.058	0.0
media	8	43	88%	68%	0.02	48	70%	59%	39.6
ds		3.90	0.007	0.025	0.01	2.73	0.024	0.031	27.1
media	16	32	88%	70%	0.03	51	61%	56%	43.1
ds		6.42	0.005	0.061	0.05	6.59	0.018	0.024	19.4
media	32	36	88%	70%	0.02	56	64%	56%	88.2
ds		3.12	0.009	0.031	0.01	5.93	0.021	0.032	48.7
media	Promedio	46	99%	78%	0.03	58	77%	63%	61.7
ds		7.25	0.043	0.054	0.03	6.51	0.055	0.037	41.3
Mushroom									
media	4	14	100%	100%	0.24	13	100%	100%	3.2
ds		3.39	0.035	0.035	0.07	2.80	0.054	0.054	2.4
media	8	14	98%	97%	0.07	15	95%	95%	4.1
ds		3.36	0.000	0.001	0.01	4.54	0.000	0.000	2.2
media	16	11	95%	95%	0.07	16	95%	95%	4.1
ds		1.78	0.000	0.001	0.02	2.97	0.000	0.000	1.0
media	32	11	95%	95%	0.08	16	95%	95%	7.4
ds		3.05	0.000	0.001	0.03	5.25	0.000	0.001	8.8
media	Promedio	13	100%	100%	0.14	16	100%	100%	4.9
ds		3.23	0.012	0.012	0.08	4.19	0.013	0.013	5.0

		EDGAR				REGAL			
		Reglas	**Test**	**Train**	**Tiempo**	**Reglas**	**Test**	**Train**	**Tiempo**
New-thyroid									
media	4	12	98%	91%	0.72	12	99%	90%	12.3
ds		2.50	0.128	0.110	0.12	2.50	0.117	0.109	6.7
media	8	12	90%	88%	0.66	12	90%	83%	25.0
ds		2.13	0.008	0.033	0.56	2.13	0.008	0.033	13.8
media	16	14	90%	84%	0.80	14	90%	84%	49.5
ds		2.56	0.009	0.033	0.62	2.56	0.008	0.033	23.4
media	32	15	90%	84%	0.58	15	90%	84%	103.3
ds		2.52	0.007	0.020	0.73	2.52	0.007	0.019	55.9
media	Promedio	14	99%	94%	0.81	14	99%	91%	51.9
ds		2.97	0.027	0.036	0.42	2.97	0.027	0.036	49.7
Nursery									
media	4	259	97%	96%	0.80	290	98%	97%	13.7
ds		26.59	0.116	0.115	0.54	25.11	0.053	0.053	8.0
media	8	261	88%	87%	0.75	292	94%	92%	40.8
ds		14.27	0.006	0.006	1.23	23.64	0.003	0.003	20.9
media	16	262	92%	91%	0.98	297	94%	92%	105.3
ds		24.03	0.002	0.004	0.09	11.86	0.003	0.004	30.9
media	32	224	91%	90%	3.56	311	94%	92%	187.5
ds		34.43	0.001	0.003	0.34	15.18	0.004	0.005	49.3
media	Promedio	270	99%	98%	1.63	309	98%	97%	91.0
ds		28.41	0.027	0.027	1.50	22.41	0.013	0.013	77.2
Pima									
media	4	125	94%	73%	0.23	120	90%	69%	12.5
ds		11.27	0.051	0.046	0.31	10.49	0.052	0.041	6.2
media	8	126	90%	69%	0.15	125	83%	66%	27.1
ds		9.27	0.011	0.018	0.27	8.74	0.022	0.028	12.6
media	16	132	90%	70%	0.14	122	82%	64%	51.7
ds		8.06	0.013	0.024	0.21	8.43	0.014	0.029	23.0
media	32	132	90%	70%	0.18	124	89%	69%	159.0
ds		11.53	0.013	0.023	0.26	8.22	0.028	0.028	57.8
media	Promedio	128	94%	73%	0.19	127	94%	73%	65.5
ds		9.34	0.017	0.024	0.22	9.49	0.017	0.024	68.5

		EDGAR				REGAL			
		Reglas	**Test**	**Train**	**Tiempo**	**Reglas**	**Test**	**Train**	**Tiempo**
Segment									
media	4	132	98%	93%	0.26	124	99%	92%	8.0
ds		13.86	0.034	0.033	0.20	12.46	0.054	0.055	4.2
media	8	130	93%	89%	0.17	117	94%	87%	15.6
ds		13.55	0.006	0.007	0.02	11.91	0.004	0.020	6.9
media	16	131	95%	90%	0.37	128	94%	87%	27.9
ds		11.89	0.003	0.008	0.02	13.99	0.005	0.015	12.0
media	32	119	96%	91%	0.95	160	94%	88%	65.2
ds		11.54	0.003	0.010	0.16	12.84	0.005	0.018	33.6
media	Promedio	132	99%	93%	0.47	137	99%	92%	30.5
ds		13.51	0.012	0.013	0.35	22.00	0.013	0.022	29.5
Soybean									
media	4	66	98%	94%	0.17	72	98%	93%	5.2
ds		7.32	0.116	0.111	0.02	8.73	0.118	0.108	1.3
media	8	65	89%	86%	0.12	66	89%	83%	9.6
ds		7.62	0.003	0.010	0.01	3.89	0.015	0.021	1.0
media	16	67	89%	86%	0.33	78	89%	82%	19.4
ds		4.08	0.003	0.009	0.04	7.10	0.017	0.021	2.7
media	32	71	90%	86%	0.90	87	88%	82%	45.3
ds		6.93	0.003	0.011	0.14	9.03	0.022	0.032	3.8
media	Promedio	72	98%	94%	0.41	81	97%	91%	21.8
ds		6.97	0.026	0.026	0.36	11.94	0.031	0.034	17.6
Splice									
media	4	84	100%	97%	11.90	83	100%	93%	24.4
ds		10.26	0.095	0.093	5.81	9.37	0.088	0.090	5.5
media	8	73	91%	89%	6.52	86	83%	87%	36.5
ds		7.97	0.001	0.008	2.74	8.24	0.001	0.007	2.5
media	16	66	91%	89%	10.22	68	86%	88%	39.7
ds		6.91	0.001	0.006	4.28	6.42	0.001	0.006	4.0
media	32	53	86%	84%	5.99	65	84%	77%	25.6
ds		8.59	0.000	0.009	1.19	8.77	0.000	0.009	1.2
media	Promedio	77	100%	97%	9.71	72	95%	97%	29.1
ds		10.91	0.027	0.028	4.53	12.41	0.026	0.027	4.5

		EDGAR				REGAL			
		Reglas	**Test**	**Train**	**Tiempo**	**Reglas**	**Test**	**Train**	**Tiempo**
Tic-tac-toe									
media	4	70	99%	95%	0.12	63	97%	90%	5.2
ds		10.38	0.119	0.117	0.03	11.28	0.113	0.108	2.8
media	8	74	90%	85%	0.03	39	85%	78%	42.2
ds		6.83	0.006	0.028	0.00	13.25	0.014	0.025	17.8
media	16	83	90%	86%	0.04	44	82%	74%	69.4
ds		4.68	0.007	0.025	0.01	11.31	0.011	0.024	56.9
media	32	95	91%	86%	0.03	40	77%	70%	106.8
ds		7.97	0.002	0.021	0.01	7.09	0.021	0.022	24.1
media	Promedio	87	99%	94%	0.06	50	91%	84%	61.4
ds		14.45	0.028	0.035	0.04	13.12	0.055	0.058	51.8
Vehicle									
media	4	190	92%	60%	1.23	143	95%	62%	6.9
ds		13.06	0.049	0.044	2.40	17.44	0.115	0.085	1.5
media	8	171	89%	58%	0.46	139	86%	54%	13.3
ds		10.66	0.010	0.022	0.92	8.89	0.013	0.030	3.2
media	16	162	90%	60%	0.23	150	86%	56%	23.1
ds		7.32	0.012	0.023	0.19	10.96	0.014	0.031	5.2
media	32	147	63%	35%	0.13	168	86%	55%	101.0
ds		5.00	0.018	0.025	0.02	10.98	0.011	0.042	36.1
media	Promedio	181	95%	62%	0.64	161	95%	61%	39.5
ds		14.11	0.019	0.033	1.42	18.32	0.028	0.039	46.2
Vote									
media	4	18	97%	96%	0.03	13	66%	63%	12.8
ds		3.54	0.115	0.112	0.00	1.00	0.000	0.003	14.9
media	8	17	88%	86%	0.01	13	66%	64%	16.6
ds		4.53	0.006	0.009	0.00	2.83	0.007	0.000	0.4
media	16	19	89%	86%	0.01	13	66%	63%	12.6
ds		3.48	0.006	0.006	0.00	1.00	0.003	0.006	13.9
media	32	29	93%	89%	0.02	14	67%	64%	17.1
ds		5.56	0.008	0.009	0.01	1.41	0.004	0.006	0.1
media	Promedio	24	97%	95%	0.02	8	53%	51%	11.8
ds		7.13	0.021	0.022	0.01	9.36	0.442	0.424	11.7

		EDGAR				REGAL			
		Reglas	**Test**	**Train**	**Tiempo**	**Reglas**	**Test**	**Train**	**Tiempo**
Waveform									
media	4	1064	93%	71%	9.74	1032	92%	70%	9.7
ds		96.97	0.075	0.060	4.58	96.97	0.074	0.058	4.6
media	8	530	47%	36%	19.93	1040	93%	71%	9.8
ds		54.27	0.075	0.061	4.77	97.84	0.073	0.058	4.6
media	16	880	88%	67%	18.41	843	46%	35%	19.9
ds		45.21	0.005	0.014	6.03	121.09	0.059	0.068	4.7
media	32	894	85%	64%	17.52	1069	86%	65%	18.4
ds		35.87	0.004	0.009	5.98	45.21	0.005	0.014	6.0
media	Promedio	1026	95%	72%	15.15	1070	93%	71%	15.1
ds		77.79	0.044	0.035	8.81	77.79	0.044	0.035	8.8
Wine									
media	4	51	97%	49%	0.04	60	94%	48%	1.1
ds		9.97	0.055	0.084	0.02	11.30	0.054	0.083	0.5
media	8	52	88%	47%	0.02	53	86%	46%	0.7
ds		5.99	0.013	0.041	0.00	5.99	0.013	0.040	0.1
media	16	65	89%	35%	0.02	63	88%	34%	1.1
ds		15.12	0.016	0.079	0.01	14.14	0.016	0.077	0.3
media	32	68	89%	41%	0.03	64	88%	38%	1.2
ds		14.55	0.017	0.094	0.01	17.20	0.019	0.089	0.3
media	Promedio	57	97%	47%	0.03	60	97%	47%	1.2
ds		13.51	0.031	0.087	0.01	14.52	0.031	0.099	0.3
Wisconsin									
media	4	24	98%	94%	0.03	23	99%	93%	4.9
ds		4.74	0.115	0.113	0.00	4.40	0.117	0.113	2.8
media	8	22	87%	85%	0.01	22	90%	86%	7.0
ds		3.93	0.007	0.007	0.00	2.56	0.004	0.012	4.4
media	16	15	48%	47%	0.05	24	91%	86%	16.0
ds		3.93	0.027	0.004	0.03	4.37	0.004	0.014	10.1
media	32	18	87%	85%	0.01	25	91%	84%	25.8
ds		4.29	0.047	0.032	0.00	2.65	0.004	0.009	8.2
media	Promedio	25	98%	95%	0.02	25	100%	94%	14.7
ds		4.10	0.056	0.056	0.01	3.63	0.026	0.029	5.4

		Edgar				Regal			
		Reglas	**Test**	**Train**	**Tiempo**	**Reglas**	**Test**	**Train**	**Tiempo**
Zoo									
media	4	10	99%	83%	0.04	9	98%	81%	10.2
ds		2.21	0.125	0.114	0.01	1.50	0.120	0.109	3.3
media	8	9	90%	81%	0.02	8	89%	72%	15.6
ds		2.52	0.017	0.057	0.01	1.49	0.015	0.073	3.8
media	16	8	90%	75%	0.01	9	82%	66%	28.8
ds		4.80	0.016	0.079	0.00	1.70	0.034	0.058	3.7
media	32	12	66%	58%	0.01	13	90%	68%	32.9
ds		3.63	0.314	0.269	0.00	4.59	0.024	0.073	4.6
media	Promedio	9	69%	57%	0.02	6	65%	53%	9.1
ds		5.04	0.314	0.269	0.01	4.10	0.311	0.243	14.5

Bibliografía

[Alba 1999] Alba E., Troya JM. (1999). A survey of parallel distributed genetic algorithms. Complexity 4 (4), pp. 31-52

[Alba 2002] Alba E., Nebro A.J., Troya J.M. (2002) Heterogeneous computing and
parallel genetic algorithms, J. Parallel Distrib. Comput. 62, pp. 1362–1385.

[Alcalá 2009] Alcalá-Fdez J., Sánchez L., García S., Del Jesus M.J., Ventura S., Garrell J.M., Otero J., Romero C., Bacardit J., Rivas V.M., Fernández J.C, Herrera F. (2009). KEEL: A software tool to assess evolutionary algorithms to data mining problems. Soft Computing, vol. 13, no. 3, pp. 307-318

[Anglano 1998] Anglano, C., Giordana, A., Bello, G. L., Saitta,L. (1998). An experimental evaluation of coevolutive concept learning. Proc. 15th International Conf. on Machine Learning. pp. 19-27

[Araujo 1999] Araujo, D.L.A.; Lopes, H.S., Freitas.(1999). A.A. A parallel genetic algorithm for rule discovery in large databases. Proc. 1999 IEEE Systems, Man and Cybernetics Conf., III, pp. 940-945

[Augier 1995] Augier, S., Venturini, G., Kodrato, Y. (1995). Learning first order logic rules with a genetic algorithm. The First International Conference on Knowledge Discovery and Data Mining. pp. 21-26

[Bacardit 2003] Bacardit, J., Garrell, J. M. Bloat (2003). Control and Generalization pressure using the minimum description length principle for a pittsburgh approach learning classifier system. In Proceedings of the 6th International Workshop on Learning Classifier Systems (in press), LNAI, Springer.

[Bacardit 2007] Bacardit J., Goldberg D.E., and. Butz M.V. (2007). Improving the performance of a pittsburgh learning classifier system using a default rule. International Workshop on Learning Classifier Systems, 439, pp. 291–307.

[Barandela 2003] Barandela R, Sánchez J.S., García V., Rangel E.(2003) Strategies for learning in class imbalance problems. Pattern Recognition, 36:3 849-851.

[Barrezta 2005] Barrezta V., Cuma M., Ferraro M., Facellia C.(2005). A general framework to understand parallel performance in heterogeneous clusters: analysis of a new adaptive parallel genetic algorithm J. Parallel and Distributed Computing. 65: pp. 48 – 57.

[Batista 2004] Batista G., Prati R.C., Monard M.C.(2004) A study of the behavior of several methods for balancing machine learning training data. SIGKDD Explorations 6:1, pp. 20-29.

[Beasley 1993] Beasley, D.; Bull, D.R., Martin, R.R. (1993). "A sequential niche technique for multimodal function optimization". Evolutionary Computation, 1(2): pp. 101-125.

[Bradley 1997] A.P. Bradley, The use of the area under the ROC curve in the evaluation of machine learning algorithms, Pattern Recognition 30(7) (1997), pp.1145-1159.

[Bernadó 2003] Bernadó-Mansilla E., Garrell J.M. (2003). Accuracy–based learning classifier systems: models, analysis and applications to classification tasks. Evolutionary Computation, vol. 11, no. 3, pp. 209–238

[Bekkerman 2011] Bekkerman, R., Bilenko, M., & Langford, J. (Eds.). (2011). Scaling up machine learning: Parallel and distributed approaches. Cambridge University Press.

[Bu 2012] Bu Y., Howe B., Balazinska M., Ernst M.D.(2012). The Haloop Approach to Large-Scale Iterative Data Analysis. PVLDB 21: pp. 169–190.

[Chawla 2002] Chawla N. V., Bowyer K. W., Hall L. O., y Kegelmeyer W. P. (2002) Smote: Synthetic minority over-sampling technique. Journal of Articial Intelligence Research 16: pp. 321–357.

[Chawla 2004] Chawla, N. V., Japkowicz, N., & Kotcz, A. (2004). Editorial: special issue on learning from imbalanced data sets. ACM Sigkdd Explorations Newsletter, 6(1), pp. 1-6.

[Chawla 2008] Chawla N. V., Cieslak D. A., Hall L. O., y Joshi A. (2008) Automatically countering imbalance and its empirical relationship to cost. Data Mining and Knowledge Discovery 17(2): pp. 225–252.

[Cantu-Paz 2000] Cantu-Paz E.(2000).Efficient and accurate parallel genetic algorithms, Kluwer Academic publishers.

[Cantú-Paz 2003] Cantú-Paz E., Kamath C. (2003). Inducing oblique decision trees with evolutionary algorithms. IEEE Transactions on Evolutionary Computation, vol. 7, no. 1, pp. 54–6.

[Cohen 1995] Cohen W.W. (1995). Fast effective rule induction. Proceedings of the Twelfth International Conference on Machine Learning Morgan Kaufmann, vol. 3, pp. 115-123.

[Cohen 2006] Cohen G., Hilario M., Sax H., Hugonnet S., Geisbuhler A. (2006). Learning from Imbalanced Data in Surveillance of Nosocomial Infection. Artificial Intelligence in Medicine 37: pp. 7-18

[Danyluk 1993] Danyluk A.P., Provost P.J.(1993). Small disjoints in action: learning to diagnose errors in the local loop of the telephone network. Proc 10th Int Conf. machine Learning, pp. 81-88.

[Davidor 1991] Davidor Y.(1991). A Naturally Occurring Niche & Species Phenomenon: The Model and First Results. Procs. of the 4th ICGA, R. K Belew, L. B. Booker (eds.). pp. 257-263.

[Dean 2008] Dean J, Ghemawat (2008) S. MapReduce: simplified data processing on large clusters. Commun ACM,51:107–113.

[Deb 1989] Deb K., Goldberg D.E. (1989). An investigation of niche and species formation in genetic function optimization. Proc. 3rd Int. Conference Genetic Algorithms, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 42–50.

[Dejong 1975] DeJong K.A.(1975). An analysis of the behavior of a class of genetic adaptative systems. Ph.D. dissertation, Univ. of Michigan, Ann Arbor.

[Dejong 1993] De Jong, K. A., Spears W. M. and Gordon F. D. (1993). ÒUsing Genetic Algorithms for Concept LearningÓ, Machine Learning, 13 , pp. 161-188, Kluwer Academic Publishers

[Demšar 2006] J. Demšar (2006) Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research. 7: pp. 1–30.

[De Jong 1975] De Jong K.A.(1975). An analysis of the behaviour of a class of genetic adaptive systems. Tesis doctoral, University of Michigan.

[De Jong 1991] De Jong K.A. and. Spears W.M. (1991). Learning concept classification rules using genetic algorithms. Proceedings International Joint Conference Artificial Intelligence. pp. 651–656.

[De Jong 1993] DeJong K.A., Spears W.M., D.F. Gordon.(1993). Using genetic algorithms for concept learning. Machine Learning, 1(13):161–188.

[Del Río 2014] del Río, S., López, V., Benítez, J. M., & Herrera, F. (2014). On the use of MapReduce for imbalanced big data using Random Forest. Information Sciences, 285, pp. 112-137.

[Dietterich 1999] T.G. Dietterich, "Approximate statistical tests for comparing supervised classification", learning algorithms Neural Computation 10 (7), pp. 1895–1924

[Domingos 1999] Domingos P (1999) Metacost: a general method for making classifiers cost sensitive. In: Advances in Neural Networks, International Journal of Pattern Recognition and Artificial Intelligence, pp. 155–164

[Dougherty 1995] Dougherty J., Kohavi R, Sahami.(1995). Supervised and unsupervised discretization of continuous features. In Proceedings of the 12th International Conference on Machine Learning, pp. 194–202, Los Altos, CA, Morgan Kaufmann.

[Eiben 2003] Eiben A.E., Smith J.E. (2003). Introduction to evolutionary computing. Springer-Verlag

[Eshelman 1991] Eshelman J.L.(1991). The CHC adaptive search algorithm: how to safe search when engaging in non traditional genetic recombination, In Foundations of Genetic Algorithms,G.J.E. Rawlins (Ed.), Morgan Kaufmann, San Mateo, pp. 265-283.

[Eshelman] Eshelman L.J. (1993). Real-coded genetic algorithms and interval schemata, In Foundations of Genetic Algorithms 2, L.D. Whitley (Ed.), Morgan Kaufmann Publishers,San Mateo, pp. 187-202.

[Forgy 1965] Forgy E. (1965).Cluster analysis of multivariate data: Efficiency vs. interpretability of classifications, Biometrics. pp. 21- 76.

[Fayyad 1996] Fayyad U.M., S. George Djorgovski, Nicholas W. (1996). Automating the Analysis and Cataloging of Sky Surveys. Advances in Knowledge Discovery and Data Mining. pp. 471-493

[Fayyad & Piatetsky 1996] Fayyad, U., Piatetsky-Shapiro, G., Smyth, P. (1996). From data mining to knowledge discovery in databases. AI magazine, 17(3), 37.

[Fernández 2010] Fernández A., García S., Luengo J., Bernadó-Mansilla E., Herrera F. (2010). Genetics-based machine learning for rule induction: state of the art, taxonomy and comparative study. IEEE Transaction Evolutionary Computation, 14(6) pp. 913-941

[Fernández 2011] Fernández, A., García, S., & Herrera, F. (2011). Addressing the classification with imbalanced data: open problems and new challenges on class distribution. In Hybrid Artificial Intelligent Systems (pp. 1-10). Springer Berlin Heidelberg.

[Flockhart 1995] Flockhart, I.W. and Radcliffe, N.J. (1995).GA-MINER: parallel data mining with hierarchical genetic algorithms – final report. EPCC-AIKMS-GA-MINER Report1.0. University of Edinburgh, UK..

[Freitas 1998] Freitas, A.A. and Lavington, S.H. (1998) Mining Very Large Databases with Parallel Processing. Kluwer,1998

[Freitas 2002] Freitas A.A. (2002). Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer-Verlag.

[Garcia 2008] García F., Herrera F.(2008). An Extension on 'Statistical Comparisons of Classifiers over Multiple Data Sets' for all Pairwise Comparisons". Journal of Machine Learning Research 9: pp. 2677-2694.

[Garcia 2009] García S., Fernández A., Luengo J., Herrera F. (2009). A Study of Statistical Techniques and Performance Measures for Genetics-Based Machine Learning: Accuracy and Interpretability. Soft Computing 13(10): pp. 959-977.

[Garcia 2013] García S, Luengo J., Sáez J.A., López V., Herrera F. (2013). A Survey of Discretization Techniques: Taxonomy and Empirical Analysis in Supervised Learning. IEEE Transactions On Knowledge And Data Engineering. 25(4) : pp. 734-740.

[Giordana 1994] Giordana A., Saitta L. (1994). Learning Disjunctive Concepts by Means of Genetic Algorithms. Proc. Int. Conf. on Machine Learning, Morgan Kaufmann (New Brunswick, NJ), 96-104.

[Goldberg 1987] Goldberg, D. E., J. Richardson (1987), "Genetic Algorithms with Sharing for Multimodal Function Optimization," in J.

Grefensette (ed.), Proceedings of the 2nd International Conference on Genetic Algorithms, Lawrence Erlbaum. pp. 41-49.

[Goldberg 1989] Goldberg, D. E. (1989).Genetic algorithms in search, optimization and machine learning. Addison-Wesley, New York,.

[Goldberg 1991] Goldberg, D. E.(1991). A tutorial on genetic algorithms theory. Fourth international Conference on genetic Algorithms, University of California at San Diego, La Jolla, CA.

[Gordon 1993] Gordon V.S., Whitley D.(1993) "Serial and Parallel Genetic Algorithms as Function Optimizers". Procs.of the 5th ICGA, S. Forrest (ed.), Morgan Kaufmann, pp. 177-183.

[Grama 1994] Grama, A., Gupta, A., Karypis, G. (1994). Introduction to parallel computing: design and analysis of algorithms. Redwood City, CA: Benjamin/Cummings Publishing Company.

[Greene 1993] Greene D.P., Smith S.F. (1993). ÒCompetition-Based Induction of Decision Models from ExamplesÓ, Machine Learning, 13 , 229-258, Kluwer Academic Publishers

[Grosso 1985] Paul Bryant Grosso. (1985). Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model. Ph.D. Dissertation. University of Michigan, Ann Arbor, MI, USA. AAI8520908.

[Grzymala 2003]. Grzymala-Busse Jw. (2003).L. K. Goodwin,.(2003). X. Zhang, Increasing sensitivity of preterm birth by changing rule strengths, Pattern Recognition Letters 24 (6): pp. 903-910

[Hekanaho 1998] Hekanaho, J. (1998). DOGMA: a GA based relational learner. Proceedings of the 8th International Conference on Inductive Logic Programming. 205-214.

[Herrera 2003] Herrera H., Lozano M, Sánchez A. (2003). A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study.International Journal of Intelligent Systems. 18. pp. 309–338.

[Hockney 1988] Hockney, R. W., Jesshope, C. R. (1988). Parallel Computers 2: architecture, programming and algorithms (Vol. 2). CRC Press.

[Holland, 1975] Holland J.H..(1975). Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI.

[Holland, 1986] Holland J.H.(1986). Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. Machine Learning: An Artificial Intelligence Approach. Volume II, Morgan Kaufman Publishers.

[Hong 2007] Hong, X., Chen, S., Harris, C. J. (2007). A kernel-based two-class classifier for imbalanced data sets. IEEE Transactions on Neural Networks, 18(1): pp. 28–41.

[Hwang 1993] Hwang K.(1993). Advanced Computer Architectures: Parallelism, Scalability, Programmability. McGraw-Hill.

[Janikow 1993] Janikow, C. (1993). A knowledge intensive genetic algorithm for supervised learning. Machine Learning,13: pp. 198-228.

[Jiao 2006] Jiao L., Liu J., Zhong W. (2006). An organizational coevolutionary algorithm for classification. IEEE Transactions on Evolutionary Computation, vol. 10, no. 1, pp. 67–80.

[Jo 2004] Jo T., Japkowicz N. (2004]. Class imbalances versus small disjuncts. SIGKDD Explorations 6:1, pp.40-49

[Kennedy 1999] Kennedy C. J., Giraud-Carrier C. (1999). A depth controlling strategy for strongly typed evolutionary programming. GECCO 1999: Proceedings of the First Annual Conference 1-6. Morgan Kauman.

[Kerber 1992] Kerber R. (1992). ChiMerge: Discretization of Numeric Attributes. Proccedings Nat'l Conf. Artifical Intelligence Am. Assoc. for Artificial Intelligence (AAAI), pp. 123-128.

[Knerr 1990] Knerr S., Personnaz L., Dreyfus G. (1990). Single-layer learning revisited: A stepwise procedure for building and training a neural network Neurocomputing

[Kubat 1998] Kubat M., Holte R, Matwin S.. (1998). Machine learning for the detection of oil spills in satellite radar images, Machine Learning 30 (2-3): pp. 195-215.

[Lee 2008] Lee, C.-I., Tsai, C.-J., Wu, T.-Q., Yang, W.-P. (2008). An approach to mining the multi-relational imbalanced database. Expert Systems with Applications, 34:3021–3032

[Leung 1995] Leung, K., Wong, M. (1995). Genetic logic programming and applications. IEEE Expert, 10(5), 68-76.

[Lin 1994] Lin S., Punch S.J., Goodman E.D.(1994). Coarse-Grain Parallel Genetic Algorithms: Categorization New Approach. Parallel & Distributed Processing.

[Liu 1997] Liu H., R. Setiono (1997).Feature Selection via Discretization. IEEE Transactions on Knowledge and Data Engineering 9(4): pp. 642-645

[López 2011] Lopez L.I., Bardallo J.M., De Vega M.A., Peregrin A.(2011). Regaltc: A Distributed Genetic Algorithm for Concept learning based on regal and the treatment of counterexamples. SoftComputing 15(7), pp 1389–1403.

[López-Fernández 2013] López, V., Fernández, A., García, S., Palade, V., & Herrera, F. (2013). An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. Information Sciences, 250,pp 113-141.

[López-delRío 2015] López, V., del Río, S., Benítez, J. M., & Herrera, F. (2015). Cost-sensitive linguistic fuzzy rule based classification systems under the MapReduce framework for imbalanced big data. Fuzzy Sets and Systems, 258, 5-38.

[Luengo 2011] Luengo, J., Fernández, A., García, S., & Herrera, F. (2011). Addressing data complexity for imbalanced data sets: analysis of SMOTE-based oversampling and evolutionary undersampling. Soft Computing, 15(10), pp. 1909-1936.

[McQueen 1967] McQueen. (1997). Some methods for classication and analysis of multivariate observations, Proceeding of the Fifth Berkeley Symposium on Mathematical Statistics and Probability 1, 281-297.

[Michalski 1986] Michalski R.S., Mozetic I., Hong J., Lavrac N.(1986) The multipurpose incremental learning system AQ15 and its testing application to three medical domains. Proc. 5th Nat. Conf. AAAI, pp. 1041-1045. Philadelphia, PA.

[Nickolls 2008] Nickolls, J., Buck, I., Garland, M., Skadron, K. (2008). Scalable parallel programming with CUDA. Queue, 6(2): pp. 40-53.

[Nguyen 1995] Nguyen S.H., Skowron A. (1995). Quantization of Real Value Attributes - Rough Set and Boolean Reasoning Approach. Proceedings' Second Joint Ann. Conference Information Sciences (JCIS), pp. 34-37.

[Orriols 2009] Orriols-Puig, A., Bernadó-Mansilla, E. (2009). Evolutionary rule-based systems for imbalanced datasets.

Soft Computing, 13(3): pp. 213–225.

[Orlovsky 1978] Orlovsky S. A. (1978) Decision-making with a fuzzy preference relation. Fuzzy Sets and Systems 1: pp. 155–167

[Park 2002] Park, B. H., Kargupta, H. (2002). Distributed data mining: Algorithms, systems, and applications.

[Pétrowski 1996] Pétrowski, (1996).A. Clearing Procedure As A Niching Method For Genetic Algorithms. In Proc. IEEE Int. Conf. Evolutionary Computation, Nagoya, Japan, pp. 798-803.

[Provost 1994] Provost, F. and Hennessy D. (1994) Distributed Machine Learning: Scaling up with Coarse-grained Parallelism. In Proc of the Second International Conference on Intelligent Systems for Molecular Biology (ISMB-94).

[Provost 2001] Provost F., Fawcett H. (2001). Robust classification for imprecise environments, Machine Learning 42 (3): pp. 203-231.

[Quinlan 1991] Quinlan J.R.(1991). Improved Estimates for the Sccuracy of Small Disjunts. Machine Learning 6(1): pp. 93-98.

[Quinlan 1993] Quinlan, J. R. (1993). C4.5, "Programs for Machine Learning", Morgan Kaufmann Publishers, 1993.

[Quinn 1994] Quinn M.J. (1994). Parallel Computing: theory and practice. McGraw-Hill.

[Ramirez-Gallego 2015] Ramírez-Gallego, S., Garcia, S. ; Benitez, J.M. ; Herrera, F..(2015). Multivariate Discretization Based on Evolutionary Cut Points Selection for Classification. Cybernetics, IEEE Transactions on PP(99.DOI 10.1109/ TCYB.2015.2410143

[Raskutt 2004] Raskutti B., Kowalczyk A.(2004) Extreme rebalancing for SVMs: a case study,SIGKDD Explorations 6 (1) pp. 60-69.

[Rifkin, 2004] Rifkin R., Klautau A. (2004) In defense of one-vs-all classification. Journal of Machine Learning Research 5: 101–141.

[Saez 2013] Saez, J. A., Luengo, J., & Herrera, F. (2013). Predicting noise filtering efficacy with data complexity measures for nearest neighbor classification. Pattern Recognition, 46(1), pp. 355-364.

[Sen 2008] Sen P, Getoor L (2008) Cost-sensitive learning with conditional markov networks. DatavMining and Knowledge Discovery 17(2): pp. 136–163

[Spears 1995] Spears W. M., De Jong, K. D. (1995). On the virtues of parameterized uniform crossover. NAVAL RESEARCH LAB WASHINGTON DC.

[Srinivas 1994] Srinivas, M., & Patnaik, L. M. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. Systems, Man and Cybernetics, IEEE Transactions on, 24(4), 656-667.

[Su 2006] Su, C.-T., Chen, L.-S., Yih, Y. (2006). Knowledge acquisition through information granulation for imbalanced data. Expert Systems with Applications, 31: pp. 531–541.

[Sun 2007] Sun Y, Kamel MS, Wong AKC, Wang Y (2007) Cost–sensitive boosting for classification of imbalanced data. Pattern Recognition 40: pp. 3358–3378

[Tan 2005] Tan P.-N., Steinbach M., y Kumar V. (2005) Introduction to Data Mining. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

[Tanesse 1989] Tanese, R.(1989). Distributed genetic algorithms for function optimization. PhD thesis, The University of Michigan, Ann Arbor, MI.

[Turney 1995] Turney, P.D. 1995. Cost-Sensitive Classification: Empirical Evaluation of a Hybrid Genetic Decision Tree Induction Algorithm. Journal of Artificial Intelligence Research 2: pp. 369-409.

[Ventura 1994] Ventura D., Martinez T.R. (1994). BRACE: A Paradigm for the Discretization of Continuously Valued Data Proceedings Seventh Ann. Florida AI Research Symposium. (FLAIRS), pp. 117-121

[Venturini 1993] G Venturini. (1993). SIA: a supervised inductive algorithm with genetic search for learning attributes based concepts Machine Learning: ECML, pp. 280-296.

[Webb 1995] Webb, G. I. (1995). OPUS: An Efficient Admissible Algorithm for Unordered Search. Journal of Artificial Intelligence Research, 3: pp. 431-465.

[Weiss 1995] Weiss G.M.(1995). Learning with rare cases and small disjunts Proc. 12th Int. Conf. Machine Learning (ML-95), pp. 558-565.

[Weis 2003] Weiss G. M. y Provost F. J. (2003) Learning when training data are costly: The effect of class distribution on tree induction. Journal of Articial Intelligence Research 19: pp. 315–354.

[Weis 2004] Weiss GM (2004) Mining with rarity: a unifying framework. SIGKDD Explorations 6(1): pp. 7–19

[White 2009] White T. (2009).Hadoop: The Definitive Guide. 1st ed.Sebastopol, CA: O'Reilly.

[Wilson 1972] Wilson D.L.(1972). Asymptotic properties of nearest neighbor rules using edited data, IEEE Transactions on Systems, Man, and Communications 2 (3): pp. 408–421.

[Wilson 1995] Wilson, S. W. (1995). Classifier fitness based on accuracy. Evolutionary Computation, 3(2): pp. 149–175.

[Wolfgang 2009] Wolfgang B., Harding S..(2009) Accelerating evolutionary computation with graphics processing units. In GECCO '09: Proceedings of the 11th annual conference companion on Genetic and evolutionary computation conference, pp. 3237–3286.

[Yang 2006] Yang Q., Wu X. (2006). "10 Challenging problems in MD research" del International Journal of Information Technology, Decision Making.

[Zaharia 2010] Zaharia M., Chowdhury M., Franklin M.J., Shenker S., Stoica I. (2010) Spark: Cluster Computing with Working Sets. HotCloud, Boston, MA 1–7