MEJORAS EN EFICIENCIA Y EFICACIA DE ALGORITMOS EVOLUTIVOS PARA APRENDIZAJE SUPERVISADO

DEPARTAMENTO DE LENGUAJES Y SISTEMAS INFORMÁTICOS

Memoria de Tesis Doctoral para optar al grado de
Doctor en Informática por la Universidad de Sevilla
presentada por

D. Raúl Giráldez Rojo

Directores:

Dr. D. José C. Riquelme Santos
Dr. D. Jesús S. Aguilar Ruiz

Sevilla, Julio de 2004

Tesis Doctoral parcialmente subvencionada por la
Comisión Interministerial de Ciencia y Tecnología
con el proyecto TIC2001-1143-C03-02.

CICYT
TIC2001-1143-C03-02

D. José Cristóbal Riquelme Santos, Profesor Titular de Universidad, y D. Jesús Salvador Aguilar Ruiz, Profesor Titular de Universidad, ambos adscritos al área de Lenguajes y Sistemas Informáticos,

CERTIFICAN QUE

D. Raúl Giráldez Rojo, Ingeniero en Informática por la Universidad de Sevilla, ha realizado bajo nuestra supervisión el trabajo de investigación titulado:

Mejoras en Eficiencia y Eficacia de Algoritmos Evolutivos para Aprendizaje Supervisado

Una vez revisado, autorizan la presentación del mismo como Tesis Doctoral en la Universidad de Sevilla y estiman oportuna su presentación al tribunal que habrá de valorarlo.

Fdo. D. José C. Riquelme Santos
Profesor Titular de Universidad
Área de Lenguajes y Sistemas Informáticos

Fdo. D. Jesús S. Aguilar Ruiz
Profesor Titular de Universidad
Área de Lenguajes y Sistemas Informáticos

Sevilla, Julio de 2004

UNIVERSIDAD DE SEVILLA

DEPARTAMENTO DE LENGUAJES Y SISTEMAS INFORMÁTICOS

Nosotros, miembros del Tribunal de la Tesis Doctoral presentada por el candidato D. Raúl Giráldez Rojo para la obtención del grado de Doctor en Informática, recomendamos por la presente la aceptación de la misma.

Dr. D. José Miguel Toro Bonilla
Dpto. Lenguajes y Sistemas Informáticos
Universidad de Sevilla

Dr. D. Francisco Herrera Triguero
Dpto. C. de la Computación e I. Artificial
Universidad de Granada

Dr. D. Francesc Josep Ferri Rabasa
Dpto. de Informática
Universidad de Valencia

Dra. Dña. María José Ramírez Quintana
Dpto. Sist. Informáticos y Computación
Universidad Politécnica de Valencia

Dr. D. Carmelo del Valle Sevillano
Dpto. Lenguajes y Sistemas Informáticos
Universidad de Sevilla

Sevilla, Julio de 2004

a mis padres
a Alicia

Agradecimientos

Desde estas líneas, quiero mostrar mi más sincero agradecimiento a todas aquellas personas que de alguna manera han contribuido, tanto en el plano profesional como en el personal, al desarrollo de esta tesis doctoral.

A mis directores, D. José C. Riquelme Santos y D. Jesús S. Aguilar Ruiz, por iniciarme en el que hoy es mi trabajo y depositar en mí su confianza. A Pepe, por ofrecerme gratuitamente su experiencia, tanto en el campo de la investigación como en el de la docencia. A Jesús, por no dejar que me hundiera en los peores momentos. Su espíritu de constante superación me ha enriquecido en todos los aspectos.

A D. Miguel Toro Bonilla, porque sus palabras siempre fueron un estímulo para crecer intelectualmente.

A Paco, mi vecino más cercano en todos los sentidos, por su calidad como amigo y su ayuda en el transcurso de mi corta carrera docente e investigadora.

A mis compañeros de investigación, Roberto, Alicia, Daniel y Domingo, por su colaboración desinteresada en este trabajo.

A todos mis compañeros del Departamento de Lenguajes y Sistemas Informáticos, por hacerme sentir cada día miembro de esta gran familia.

A mis padres, Antonio y María del Carmen, por su enorme sacrificio y constante estímulo a lo largo de toda mi vida.

Y por último, aunque no con menor importancia, a Alicia, por su paciencia e infinita generosidad. Por haber confiado en mí, a veces más que yo mismo.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Resumen

Los algoritmos evolutivos conforman una de las más importantes familias de modelos computacionales con aplicación en el campo del aprendizaje automático, cuya validez y efectividad han sido ampliamente estudiada en la bibliografía. Enmarcada dentro del área del aprendizaje supervisado, esta tesis doctoral tiene como objetivo fundamental el desarrollo de diversos métodos algorítmicos dirigidos hacia la mejora de este tipo de técnicas para la generación de reglas de decisión. Se pretende reducir el coste computacional asociado a los aspectos críticos de los algoritmos evolutivos, así como aumentar la calidad de los resultados mediante una búsqueda más eficiente y eficaz de las soluciones.

En este contexto, adquiere especial relevancia la codificación de los individuos de la población genética. La denominada *Codificación Natural* se presenta como una de las aportaciones principales de esta investigación, siendo la reducción del espacio de búsqueda, la disminución del tamaño de los individuos y la definición de los operadores genéticos, sus principales mejoras frente a otras codificaciones. Por otra parte, la evaluación de los individuos es un proceso esencial pero altamente costoso. En este sentido, proponemos la estructura de datos EES, que organiza e indexa el conjunto de datos evitando el recorrido completo del mismo, al tiempo que aprovecha la semántica de la reglas de decisión para llevar a cabo una evaluación más inteligente. La herramienta HIDER recoge estas mejoras y constituye la parte fundamental de nuestra investigación. HIDER genera reglas de decisión jerárquicas como modelo de comportamiento de un conjunto de datos etiquetados a la vez que clasifican nuevas instancias de los mismos, dando así soporte en la toma de decisiones.

Durante el desarrollo de HIDER, fueron surgiendo propuestas paralelas relacionadas con aspectos concretos de la herramienta. Destaca el algoritmo de discretización supervisada USD, el cual reduce sustancialmente la cardinalidad del conjunto de valores de los atributos continuos. La minimización del error introducido al transformar un dominio continuo y teóricamente infinito en discreto y finito, hace que USD tenga mejor comportamiento que otros métodos desde el punto de vista de la posterior aplicación de algoritmos evolutivos de generación de reglas.

Los resultados de las diversas pruebas empíricas realizadas reflejan la calidad y robustez de HIDER, alcanzando importantes mejoras frente a otras herramientas muy contrastadas en la comunidad científica. Asimismo, el estudio llevado a cabo en colaboración con la multinacional ATLANTIC COPPER S.A. como aplicación práctica de HIDER a un proceso industrial, muestra el importante potencial de esta investigación frente a problemas reales.

Capítulo 1

Introducción

Todo comienzo tiene su encanto.
JOHANN W. VON GOETHE.

1.1. Planteamiento

El desarrollo tecnológico alcanzado en nuestros días, unido al consecuente abaratamiento de los recursos, ha propiciado que cualquier entidad sea capaz de almacenar todos los datos generados por su actividad. Esta facilidad para generar y almacenar información ha fomentado en los últimos años el desarrollo y perfeccionamiento de técnicas para la extracción de conocimiento a partir de grandes conjuntos de datos para su posterior aplicación en la toma de decisiones.

El proceso completo de extraer conocimiento a partir de bases de datos se conoce como *KDD* (*Knowledge Discovery in Databases*). Este proceso, descrito en profundidad en el Capítulo 2, comprende diversas etapas, que van desde la obtención de los datos hasta la aplicación del conocimiento adquirido en la toma de decisiones. Entre esas etapas, se encuentra la que puede considerarse como el núcleo del proceso *KDD* y que se denomina Minería de Datos (*Data mining*). Esta fase es crucial para la obtención de resultados apropiados, pues durante la misma se aplica el algoritmo de aprendizaje automático encargado de extraer el conocimiento inherente a los datos. Podemos definir la minería de datos como el proceso no trivial de inferir conocimiento, previamente desconocido, potencialmente útil y humanamente comprensible, a partir de grandes

cantidades de datos, con el propósito de predecir de manera automática comportamientos y tendencias. La elección del algoritmo de aprendizaje para llevar a cabo esta tarea es determinante.

Entre la gran cantidad de técnicas de aprendizaje automático existentes [122, 127], algunas de las cuales son comentadas en el Capítulo 2, destacan las basadas en *Algoritmos Evolutivos* [29]. Se trata de un tipo de técnicas bioinspiradas las cuales realizan una búsqueda probabilística en el espacio de soluciones. Los Algoritmos Evolutivos son usados para procesar problemas con un conjunto de soluciones muy grande, a menudo infinito, donde no es posible aplicar una búsqueda exhaustiva en la práctica. Éste es el caso de los problemas abordados en aprendizaje automático, donde pueden existir infinitos modelos que representen el comportamiento de un conjunto de datos, aunque sólo unos pocos resulten útiles. Cuando la optimización o búsqueda del mejor modelo se lleva a cabo mediante técnicas de computación evolutivas, suele denominarse *Aprendizaje Evolutivo*. El motivo del éxito de este tipo de aprendizaje reside en su buen comportamiento en un gran número de dominios respecto a la calidad de los modelos de conocimiento aprendidos. Sin embargo, su buen funcionamiento lleva asociado un elevado coste computacional, debido principalmente a la búsqueda estocástica de las soluciones y a la repetitiva evaluación de éstas.

En este contexto, el punto de partida de esta investigación es la herramienta denominada COGITO [1], cuyo objetivo es la generación de reglas de decisión mediante algoritmos evolutivos en aprendizaje supervisado. La calidad de este sistema ha sido contrastada en diversos trabajos, entre los que destacan [5, 6, 7, 8, 154, 156]. Sin embargo, el desarrollo de nuevas técnicas de preprocesado, codificación y evaluación pueden mejorar sustancialmente los resultados de COGITO.

1.2. Objetivos

Como se ha apuntado anteriormente, el objetivo de nuestra investigación es mejorar las técnicas de aprendizaje evolutivo, centrándonos inicialmente en la herramienta COGITO y extendiendo, posteriormente, el estudio a otras propuestas. Estas mejoras

son abordadas desde los dos puntos de vista fundamentales de un algoritmo: la *eficiencia*, reduciendo el coste computacional en tiempo y espacio de las tareas críticas del algoritmo y acelerando la convergencia en la búsqueda de soluciones; y la *eficacia*, aumentando la calidad de los resultados mediante una representación más adecuada del modelo y una búsqueda más efectiva de las soluciones.

Los algoritmos evolutivos son una familia de modelos computacionales inspirados en el concepto Darwiniano de evolución, los cuales emplean un método de búsqueda aleatoria para encontrar soluciones a un problema particular [80]. Partiendo de un conjunto de soluciones iniciales, genera nuevas soluciones mediante la selección de las mejores y recombinación de éstas, simulando así la evolución de una población de individuos. En general, un algoritmo evolutivo consta de los siguientes componentes básicos [121]:

1. Representación de las soluciones potenciales al problema (*codificación*), transformando éstas en *individuos genéticos*.

2. Método para generar las soluciones iniciales (*población inicial*).

3. Función de *evaluación* que juega el papel del ambiente, calificando las soluciones en términos de aptitud o bondad.

4. Método de *selección*, que emula la selección natural según la aptitud de los individuos.

5. Operadores genéticos (*cruce* y *mutación*), que simulan la reproducción de los individuos, alterando la composición de los descendientes para desencadenar la evolución.

Aunque todos estos aspectos tiene una influencia notable en la eficiencia y eficacia, las dos tareas principales en la aplicación de un algoritmo evolutivo son el diseño de la codificación y la evaluación de los individuos. Por ello, inicialmente enfocamos nuestros esfuerzos a la mejora de estos dos aspectos junto al de los operadores genéticos, estrechamente relacionados con la codificación. No obstante, durante el transcurso de la investigación, fueron necesarias diversas adaptaciones sobre el resto de factores.

La codificación influye directamente sobre el tamaño del espacio de búsqueda, el cual condiciona a su vez la convergencia del algoritmo, así como la probabilidad de encontrar buenas soluciones. En concreto, la codificación real aplicada habitualmente hace que el espacio de búsqueda sea teóricamente infinito para dominios continuos. Esto nos motivó a desarrollar la denominada *Codificación Natural*, con el objeto de minimizar, o al menos reducir, el número de soluciones potenciales, sin que se produjera pérdida en la precisión en las mismas. Conjuntamente, fueron diseñados los operadores genéticos específicos que contribuyeran a la aceleración de la búsqueda.

Respecto a la evaluación de los individuos hay que tener en cuenta dos aspectos importantes. En primer lugar, la función de evaluación es el elemento más influyente en la calidad del modelo final, puesto que precisamente mide la bondad de las soluciones respecto al conjunto de datos de entrada durante el proceso evolutivo. Debido a su importancia, existen innumerables propuestas sobre el diseño de la función de evaluación. En segundo lugar, el método evaluación afecta sustancialmente al tiempo de ejecución del algoritmo, ya que la evaluación de un solo individuo lleva consigo habitualmente un recorrido de los datos de entrenamiento. Tal aspecto, no menos importante que el anterior, ha sido quizá más descuidado. Este problema de eficiencia nos impulsó a plantear nuevos métodos de evaluación para mitigar el coste computacional asociado a esta fase del algoritmo evolutivo, los cuales confluyeron finalmente en el desarrollo la *Estructura de Evaluación Eficiente* (EES, *Efficient Evaluation Structure*).

Paralelamente al análisis anterior, surgieron algunas ideas interesantes que, no perteneciendo estrictamente al contexto del aprendizaje evolutivo, sí podrían contribuir a la mejora de este tipo de técnicas. La incorporación esas ideas a las necesidades que iban surgiendo en el núcleo de la investigación, dio como resultado el método de discretización supervisada denominado USD (*Unparametrized Supervised Discretization*), descrito en el Capítulo 4, y que, posteriormente, se convertiría en esencial para la aplicación de la codificación natural y la estructura EES.

Los resultados de esta investigación se integran en la herramienta HIDER (*Hierarchical Decision Rules*), cuyos fundamentos son expuestos en el Capítulo 5. Para medir la calidad de nuestra propuesta, se llevaron a cabo numerosos estudios experimentales,

presentándose en el Capítulo 6 los resultados más representativos. Es importante señalar que las pruebas realizadas son totalmente reproducibles, ya que se eligieron para ello bases de datos muy conocidas en el área, incluidas en el *UCI Machine Learning Repository* [24] y utilizadas por numerosos autores.

Actualmente, además de continuar profundizando en el desarrollo de la herramienta, se estudia la utilización de HIDER en un dominio industrial, en colaboración con la empresa multinacional ATLANTIC COPPER S.A. Concretamente, se está aplicando para la toma de decisiones en la producción de ácido sulfúrico durante el proceso de obtención del cobre con el objetivo de optimizar aprovechamiento de ambos productos. El Capítulo 7 describe esta aplicación práctica y presenta los resultados obtenidos hasta el momento.

1.3. Aportaciones originales

Desde que comenzó esta investigación en el año 2000 hasta hoy, se han realizado diferentes propuestas con idea de ir perfeccionando el funcionamiento de la herramienta HIDER, principalmente en los aspectos anteriormente mencionados de codificación genética, evaluación de individuos y discretización de atributos continuos. Estos estudios, junto a los resultados obtenidos, han sido expuestos y publicados en diferentes foros especializados.

1.3.1. Relacionadas con la discretización

- Una Propuesta para Reducir el Coste de Evaluación en Aprendizaje Evolutivo. *Tercer Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'04)*, pp. 134–140. Córdoba, 2004. ISBN:84-688-4224-9

- Discretization Oriented to Decision Rules Generation. *Frontiers In Artificial Intelligence And Applications*, 82(1):275-279. IOS–Press, 2002. ISBN:1-58603-289-1, ISSN:0922-6389.

- Discretization by Maximal Global Goodness. *Proceedings of the International Conference on Fuzzy Systems and Knowledge Discovery, (FSKD'02)*, pp. 742-746. Singapore, 2002. ISBN:981-04-7520-9.

- Discretización Supervisada No Paramétrica Orientada a la Obtención de Reglas de Decisión. *Conferencia de la Asociación Española para la Inteligencia Artificial, CAEPIA'01*. Gijón, 2001. ISBN: 84-932297-1-7.

1.3.2. Relacionadas con la evaluación eficiente

- Knowledge-based Fast Evaluation for Evolutionary Learning. *IEEE Transactions on Systems, Man & Cybernetics – Part C*, (in press), 2004. ISSN: 84-688-0206-9.

- An Efficient Data Structure for Decision Rules Discovery. *Proceedings of 18th ACM Symposium on Applied Computing (SAC'03), Data Mining Track*. Melbourne, Florida, US, 2003. ISBN: 1-58113-624-2.

- Indexación de Datos para la Evaluación Rápida de Reglas de Decisión. *VII Jornadas de Ingeniería del Software y Bases de Datos (JISBD'02)*, pp. 35–44. El Escorial, Madrid, 2002. ISBN: 84-688-0206-9.

1.3.3. Relacionadas con la codificación genética

- Natural Coding: A More Efficient Representation for Evolutionary Learning. *Genetic and Evolutionary Computation Conference, (GECCO'03). Lecture Notes in Computer Science, vol. 2723*, pp. 979–990. Springer-Verlag. Chicago, US, 2003. ISBN: 3-540-40602-6, ISSN: 0302-9743.

- COGITO*: Aprendizaje Evolutivo de Reglas de Decisión con Codificación Natural. *Segundo Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, (MAEB'03)*, pp. 538–547. Gijón, 2003. ISBN: 84-607-65-26-1

1.3.4. Otras publicaciones

- Statistical Test-based Evolutionary Segmentation of Yeast Genome. *Genetic and Evolutionary Computation Conference (GECCO '04). Lecture Notes in Computer Science, vol. 3102*, pp. 493–494. Springer-Verlag. Seattle, Washington, EE.UU. 2004. ISBN: 3-540-22344-4.

- Separation Surfaces through Genetic Programming. *Engineering of Intelligent Systems (IEA-AIE). Lecture Notes in Artificial Intelligence, vol. 2070*, pp. 428–433, Springer-Verlag. Budapest, Hungary, 2001. ISBN: 3-540-42219-6.

- SNN: A Supervised Clustering Algorithm. *Engineering of Intelligent Systems (IEA-AIE). Lecture Notes in Artificial Intelligence, vol. 2070*, pp. 207–216, Springer-Verlag. Budapest, Hungary, 2001. ISBN: 3-540-42219-6.

- Local Nearest Neighbours by Competition. *Frontiers In Artificial Intelligence And Applications*, 82(1):260-264. IOS–Press, 2002. ISBN:1-58603-289-1, ISSN:0922-6389.

- Minería de Datos: Líneas de Investigación Actuales en la Universidad de Sevilla. *Workshop de Minería de Datos y Aprendizaje, VIII Iberoamerican Conference on Artificial Intelligence (IBERAMIA'02)*, Sevilla (Spain), 2002. ISBN: 84-95499-88-6.

1.4. Organización

El contenido de este documento se organiza en los siguientes capítulos:

Capítulo 2: Minería de Datos y $\mathcal{KDD}$. En este apartado se presenta una visión general del el proceso $\mathcal{KDD}$, haciendo mayor hincapié en la fase de Minería de Datos. Asimismo, son descritos los métodos de aprendizaje supervisado más utilizados en el área, así como las formas de representación del conocimiento más eficientes.

Capítulo 3: Aprendizaje Evolutivo. En este capítulo se resumen los conceptos básicos de la Computación Evolutiva, centrándose principalmente en el aprendizaje de reglas mediante algoritmos genéticos y evolutivos. También son descritos los métodos de aprendizaje evolutivo más afines a nuestra investigación.

Capítulo 4: Discretización Supervisada No Paramétrica. Como primera parte de nuestra investigación proponemos un algoritmo de discretización supervisada, que denominamos USD. Además de su descripción, este capítulo contempla diversas pruebas empíricas y aplicaciones del mismo en el ámbito de la clasificación y la edición de ejemplos.

Capítulo 5: HIDER. La descripción detallada de nuestra propuesta principal, a la que hemos llamado HIDER, se desarrolla en este capítulo, donde se lleva a cabo un estudio de las diferentes partes que la conforman, teniendo mayor relevancia la Codificación Natural de individuos y la Estructura de Evaluación Eficiente EES.

Capítulo 6: Pruebas. Este apartado contiene los experimentos realizados para medir la eficiencia y eficacia de nuestra herramienta frente a otros métodos. En concreto, se han contrastado los resultados de HIDER con los obtenidos por C4.5, C4.5Rules y COGITO respecto a la clasificación de algunas de las bases de datos del almacén UCI.

Capítulo 7: Aplicación de HIDER a un Proceso Industrial. Para mostrar la utilidad de esta investigación frente a un problema real, este capítulo detalla una aplicación práctica de HIDER a la industria de la química. A partir de los datos cuantitativos proporcionados por la empresa ATLANTIC COPPER S.A., HIDER genera modelos de conocimiento para facilitar toma de decisiones de los expertos en la producción de ácido sulfúrico a partir de los desechos del proceso de obtención del cobre. Así, no sólo se ayuda a la optimización de la producción desde el punto de vista empresarial, sino que también reduce el impacto medioambiental que este proceso puede llevar consigo si no son correctamente resueltos los estados críticos del sistema de producción.

Capítulo 8: Conclusiones y Trabajos Futuros. Finalmente, este capítulo resume las conclusiones obtenidas durante esta investigación, las cuales nos impulsan a considerar ciertas alternativas para obtener mayor rendimiento en propuestas futuras.

Apéndice A: Datos Experimentales. Este apéndice contiene algunos datos complementarios al estudio realizado en el Capítulo 7.

Capítulo 2

Minería de Datos y $\mathcal{KDD}$

Propongo considerar la siguiente cuestión:
¿Pueden pensar las máquinas?
ALAN M. TURING.

Aunque la idea de Minería de Datos parece estar generalmente aceptada en la comunidad científica, no existe una definición clara de este término. Informalmente, podríamos definir la minería de datos como el análisis de bases de datos con el fin de descubrir o extraer información inherente a los datos objeto de análisis, de modo que sea de utilidad en la toma de decisiones. Tal información puede venir representada como patrones, relaciones, reglas, asociaciones, dependencias o incluso excepciones entre los datos analizados.

El desarrollo tecnológico ha permitido que la creciente cantidad de información que diariamente se genera en el mundo pueda ser trasferida de forma casi instantánea, así como almacenada en grandes bases de datos. La competitividad existente hoy en día en campos como la economía, el comercio, la industria y la propia ciencia entre otros, ha fomentado el aprovechamiento de esta información para la toma de decisiones. Tradicionalmente, la estadística ha cubierto este campo, ofreciendo resúmenes de los datos en forma de medias, desviaciones, distribuciones, correlaciones, entre otras muchas medidas. Sin embargo, el simple estudio estadístico de esta cantidad de información resulta insuficiente para la toma de decisiones, pues aporta un conocimiento muy limitado del comportamiento de los datos. Además de las medidas estadísticas, la vasta información oculta patrones y relaciones inherentes de gran utilidad hoy en día y que la minería de datos se encarga de extraer.

La especie humana posee habilidades extremadamente sofisticadas para detectar patrones y descubrir tendencias. Por ejemplo, en un comercio con pocas decenas de clientes, el dueño pude predecir los precios que ha de ofertar para mantener e incrementar sus ventas, ofreciendo un servicio casi personalizado. Sin embargo, en grandes centros comerciales donde se atiende a miles de clientes diariamente, esta tarea es sencillamente imposible de llevar a la práctica. Por tal motivo, si somos capaces automatizar la extracción de la información verdaderamente útil de las ventas y modelar el comportamiento de los clientes de manera adecuada, el director de ventas de este hipotético centro comercial podrá identificar tendencias en las ventas y usar esta información para incrementar los beneficios.

Para obtener conclusiones válidas y útiles al aplicar minería de datos, es necesario complementar este proceso con una adecuada preparación de los datos previa al proceso de minería y un análisis posterior de resultados obtenidos. Así, podemos afirmar que el proceso de minería de datos pertenece a un esquema más amplio denominado Descubrimiento de Conocimiento en Bases de Datos, más conocido como *KDD* (Knowledge Discovery in Databases).

2.1. Descubrimiento de Conocimiento en Bases de Datos

Una de las definiciones más extendidas de *KDD* es [58]:

"*El Descubrimiento de Conocimiento en Bases de Datos es el proceso no trivial de identificación de patrones válidos, novedosos, potencialmente útiles y fundamentalmente comprensibles en los datos*", Fayyad, Piatetsky-Shapiro y Padhraic Smyth (1996).

- **Proceso no trivial**: El término *proceso* denota que el *KDD* es una secuencia de pasos. El que sea *no trivial* se refiere a que el proceso no es un mero recorrido de los datos, sino que implica una inferencia compleja sobre los mismos en busca de ilaciones o conclusiones.

- **Patrones**: La identificación de *patrones* es, en general, la descripción a alto nivel de los datos, encontrando una estructura o un modelo de comportamiento de éstos.

- **Válidos**: Los patrones o modelos descubiertos deben gozar de cierto grado de certeza.
- **Novedosos**: Los patrones deben aportar conocimiento nuevo.
- **Potencialmente útiles**: El modelo debe ser aplicable para la toma de decisiones que impliquen beneficio.
- **Comprensibles**: Se debe generar un modelo fácilmente interpretable por el usuario, si no directamente, sí tras un procesado posterior.

Tras esta definición, podemos intuir que el $\mathcal{KDD}$ no es un campo aislado, sino la convergencia de otros campos. Las principales áreas contribuyentes son el Aprendizaje Automático, las Bases de Datos y la Estadística. Cada una de ellas aporta una serie de técnicas y herramientas que, tras una adecuada aplicación, dan como resultado un modelo de conocimiento. El área de las Bases de Datos se encarga de almacenar, acceder, buscar y actualizar datos. El Aprendizaje Automático aporta algoritmos que mejoran automáticamente a través de la experiencia, centrándose fundamentalmente en la inducción y siendo aplicable a datos tanto numéricos como simbólicos. Por último, la estadística complementa al Aprendizaje Automático aplicando técnicas de deducción e inducción de datos, principalmente numéricos.

El esquema general del proceso de $\mathcal{KDD}$ incluye cinco fases bien diferenciadas. La figura 2.1 ilustra esta secuencia de fases.

1. **Determinación de Objetivos**: Antes de aplicar el proceso de $\mathcal{KDD}$ propiamente dicho, es necesario precisar qué objetivos quieren cumplirse desde el punto de vista del usuario. Esta fase es crucial, ya que dependiendo de los objetivos marcados se elegirán determinadas técnicas de preparación de datos, minería y análisis. Un error en esta fase puede invalidar todo el proceso. Los objetivos marcados determinarán los datos que se han de usar durante el proceso de extracción de conocimiento. Asimismo, los datos disponibles condicionarán qué objetivos son viables y cuales no.
2. **Preparación de los datos**: Los datos originales pueden contener ambigüedades,

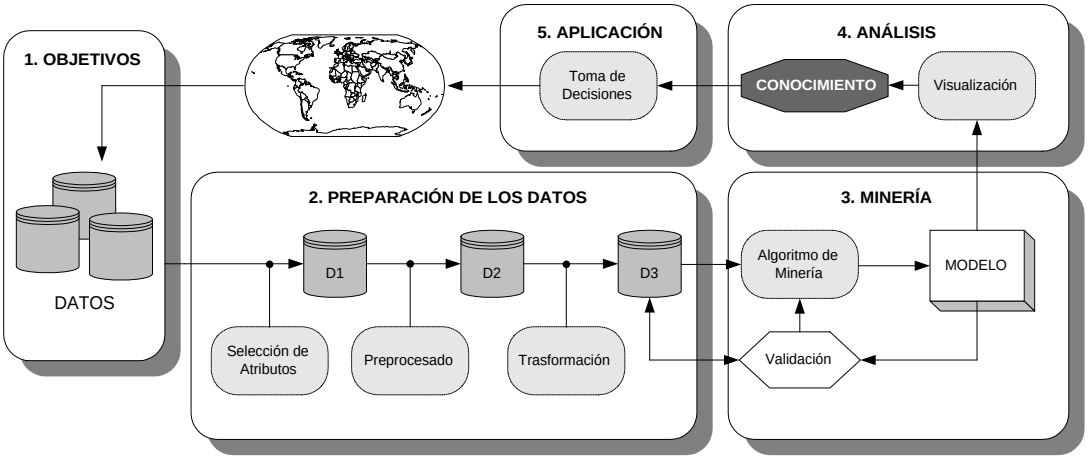


Figura 2.1: Esquema General de $\mathcal{KDD}$ (Knowledge Discovery in Databases).

ruido o, simplemente, no estar en el formato adecuado para su posterior procesamiento. Una adecuada preparación de los datos acelerará el algoritmo de minería y mejorará la calidad del modelo de conocimiento. Normalmente esta fase se divide en otras tres: *selección*, *preprocesado* y *trasformación*. La *selección* es el proceso de distinguir los subconjuntos de datos significativos y descartar aquellos que a priori no aportan información para la generación del modelo teniendo en cuenta, entre otros aspectos, los objetivos fijados. El *preprocesado* engloba a aquellas operaciones destinadas a tratar los valores ausentes, así como eliminar el posible ruido que se haya podido producir durante la recolección de los datos. Por último, la *transformación* de los datos consiste en encontrar una representación de los datos más adecuada dependiendo de los objetivos previamente fijados (*v.g.* normalizar los valores).

3. **Minería de Datos**: La elección del método de minería es fundamental dentro del proceso $\mathcal{KDD}$. La validez y utilidad del modelo obtenido depende en gran parte de esta fase. Además del algoritmo de aprendizaje, esta etapa suele incluir la validación del modelo, la cual, además de evaluar la calidad del mismo, puede ser usada para reorganizar los datos y reajustar el propio algoritmo.

4. **Análisis**: En esta etapa se estudia, interpreta y evalúa el modelo de conocimiento generado por el algoritmo de minería de datos. El uso de técnicas de visualización facilitan al usuario la comprensión e interpretación del modelo obtenido,

permitiendo la aplicación de éste en la toma de decisiones.

5. **Aplicación**: Integración del conocimiento adquirido al campo real de aplicación mediante la toma de decisiones basadas en dicho conocimiento. Esta fase suele incluir la comparación con el conocimiento previo a la aplicación del proceso *KDD*, así como la resolución de potenciales conflictos entre las decisiones tomadas.

Esta investigación se centra principalmente en la tercera etapa, es decir, en la aplicación del algoritmo de minería de datos junto a la validación del modelo de conocimiento extraído. La figura 2.2 muestra un esquema más detallado de la fase de minería, donde podemos distinguir varias partes. Partiendo del conjunto de datos resultante de la fase de preparación, éstos pueden sufrir una última trasformación para adecuarlos al algoritmo de aprendizaje concreto que vaya a ser aplicado, por ejemplo, para aumentar la eficiencia de dicho algoritmo. No incluimos esta transformación en la fase de preparación de datos debido a que únicamente es aplicada durante la etapa de aprendizaje, siendo su utilización fuera de ésta carente de sentido. De igual modo, el algoritmo de aprendizaje puede usar una representación del conocimiento diferente a la usada en el modelo final. La validación del modelo generado, ya sea final o intermedio, es aplicada para evaluar la calidad del mismo y reajustar, si es necesario, tanto el algoritmo de aprendizaje como los datos de entrada. Por último, una vez que el modelo ha sido depurado y validado convenientemente, se pasa a la fase de análisis.

Los datos de entrada, el algoritmo de aprendizaje y la representación del conocimiento que define el modelo están estrechamente relacionados. Dependiendo de los datos disponibles a la entrada se aplicará un algoritmo de aprendizaje adecuado para construir un modelo que satisfaga, en la medida de lo posible, los objetivos marcados al principio. Estos y otros aspectos se tratan con detalle en las secciones siguientes.

2.2. Marco de trabajo y Definiciones

El Aprendizaje Automático (*Machine Learning*) es la rama de la Inteligencia Artificial que estudia el desarrollo de técnicas para extraer de forma automática conocimiento

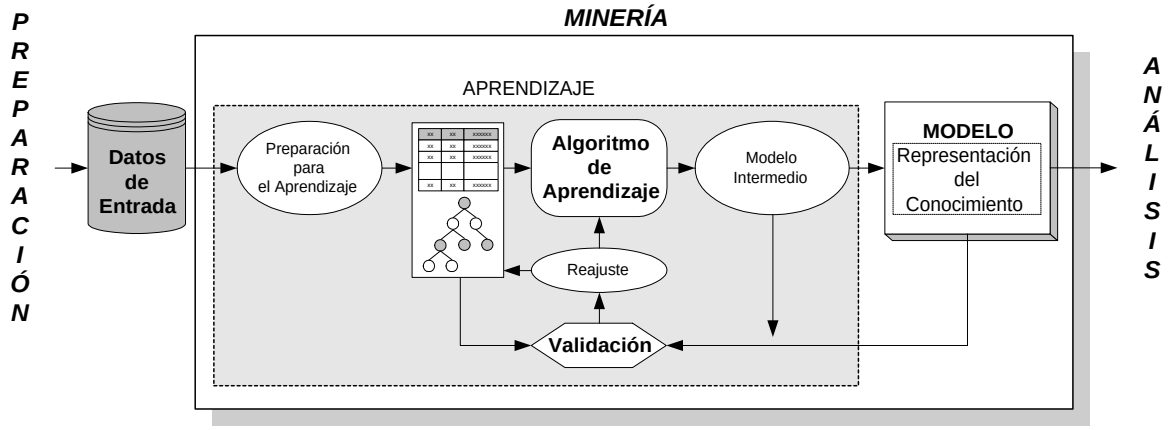


Figura 2.2: Fase de Minería de Datos.

subyacente en la vasta información. Uno de los modelos de aprendizaje más estudiados es el *aprendizaje inductivo*, que engloba todas aquellas técnicas que aplican inferencias inductivas sobre un conjunto de datos para adquirir el conocimiento inherente a ellos. Existen principalmente dos tipos de aprendizaje inductivo: *supervisado* y *no supervisado*. En el aprendizaje supervisado, los casos pertenecientes al conjunto de datos tienen a priori asignada una clase o categoría, siendo el objetivo encontrar patrones o tendencias de los casos pertenecientes a una misma clase. Por contra, el aprendizaje no supervisado no goza de una agrupación de los casos previa al aprendizaje, por los que se limita a buscar la regularidades entre éstos. En esta memoria, el aprendizaje siempre será entendido como supervisado.

A continuación, se presentan una serie de definiciones relacionadas con el marco de trabajo en el que se ha desarrollado esta investigación y que permitirán unificar algunos de los términos y conceptos utilizados a lo largo de esta memoria.

Definición 2.1 (Universo de discurso). *Se denomina universo de discurso al entorno donde se define un determinado problema y viene representado como el producto cartesiano de un conjunto finito de dominios.*

Definición 2.2 (Dominio). *Un dominio es un conjunto de valores del mismo tipo. Desde el punto de vista de esta investigación, existen dos tipos de dominios: continuo (conjunto infinito de valores reales) y discreto (conjunto finito de valores, ya sean numéricos o categóricos).*

Definición 2.3 (Atributo). *Un atributo es una cualidad o característica existente en el universo de discurso que toma valores en un determinado dominio.*

Definición 2.4 (Clase). *Una clase es un atributo especial que categoriza o clasifica un determinado grupo de casos. Se denomina etiquetas de clase al conjunto o dominio de valores que la clase puede tomar, habitualmente discreto. La clase es el atributo sobre el cual se realiza la predicción, por lo que es también denominada atributo de decisión, para diferenciarla del resto de atributos (atributos de condición).*

Definición 2.5 (Ejemplo). *Un ejemplo es un caso o instancia del universo de discurso, el cual es representado por un conjunto de valores de atributos y una etiqueta de clase que lo clasifica.*

Definición 2.6 (Conjunto de datos). *Definimos un conjunto de datos como un subconjunto finito de ejemplos del universo de discurso, el cual se caracteriza por el número de atributos, el dominio de cada uno de ellos y el número de ejemplos que contiene. De manera informal, un conjunto de datos es una base de datos donde la estructura de almacenamiento de la información es irrelevante.*

Definición 2.7 (Conjunto de entrenamiento). *Un conjunto de entrenamiento es un conjunto de datos usado como entrada al algoritmo de aprendizaje.*

Definición 2.8 (Conjunto de test). *Se denomina conjunto de test al conjunto de datos utilizado para medir la fiabilidad del conocimiento adquirido.*

Definición 2.9 (Regla de decisión). *Una regla de decisión es una implicación $\mathcal{P} \Rightarrow \mathcal{C}$. El antecedente $\mathcal{P}$ (o descripción de la regla) es formado por una conjunción de condiciones sobre los atributos de un conjunto de datos, y el consecuente $\mathcal{C}$ indica una etiqueta de clase, de manera que si un ejemplo cumple las condiciones de $\mathcal{P}$, éste será clasificado con la clase establecida por $\mathcal{C}$.*

Definición 2.10 (Cobertura de un ejemplo). *Se dice que un ejemplo es cubierto por una regla si satisface el antecedente de ésta, independientemente de si es o no clasificado correctamente.*

Definición 2.11 (Aciertos y errores de una regla). *Se dice que una regla tiene un acierto cuando cubre a un ejemplo y lo clasifica correctamente, es decir, la clase de dicho ejemplo coincide con la indicada en el consecuente. Por el contrario, una regla comete un error cuando la clase no coincide con la de un ejemplo cubierto.*

Definición 2.12 (Cobertura de una regla). *Se denomina cobertura de una regla a la proporción de ejemplos cubiertos por ésta respecto al total de ejemplos del conjunto de datos.*

Definición 2.13 (Consistencia de una regla). *La consistencia de una regla es la proporción de ejemplos cubiertos por ésta respecto al total de ejemplos del conjunto de datos que comparten la clase de la regla.*

Definición 2.14 (Exactitud de una regla). *La exactitud de una regla es la probabilidad de un ejemplo aleatorio cubierto por la misma sea clasificado correctamente. En otras palabras, es la proporción de aciertos respecto al total de ejemplos.*

Definición 2.15 (Completitud de una regla). *Una regla es completa cuando cubre todos los ejemplos de una clase existentes en el conjunto de entrenamiento.*

2.3. Representación del Conocimiento

La aplicación del proceso $\mathcal{KDD}$ tiene como objetivo extraer conocimiento de un conjunto de datos y modelar dicho conocimiento para su posterior aplicación en la toma de decisiones. La estructura elegida para representar el modelo generado se denomina habitualmente *Representación del Conocimiento*. Esta representación depende del tipo de aprendizaje aplicado, supervisado o no supervisado. Dado que este trabajo se enmarca dentro del aprendizaje supervisado, nos limitaremos a describir las representaciones más comunes en este área: *representación proposicional*, *árboles de decisión* y *reglas*.

Un modelo de conocimiento establece relaciones entre los valores que los atributos pueden tomar y las etiquetas de clase, mientras que la representación del conocimiento implementa el modelo en base a una estructura determinada. La complejidad del modelo es uno de los factores fundamentales para medir el rendimiento del mismo, y está estrechamente relacionada con el tamaño de la estructura de conocimiento.

Cuando los resultados proporcionados por el algoritmo de aprendizaje han de ser interpretados directamente por el usuario o experto, la complejidad y la legibilidad de la estructura adquieren especial importancia. En este sentido, la representación mediante reglas o proposiciones es a menudo más sencilla de comprender que la representación mediante árboles.

Aunque tanto el tamaño de la estructura de conocimiento como su inteligibilidad son características determinantes, la complejidad como medida de rendimiento sólo suele tener en cuenta la primera de ellas, ya que la comprensibilidad es quizás muy subjetiva y también está relacionada con el tamaño.

2.3.1. Representación Proposicional

La representación proposicional representa el modelo de conocimiento mediante expresiones lógicas que establecen las condiciones que los atributos, incluida la clase, deben cumplir. Existen dos alternativas para esta representación: la Forma Normal Conjuntiva (FNC) y la Forma Normal Disyuntiva (FND). Se dice que una expresión está en FNC si está escrita como una conjunción en la cual los términos son disyunciones literales. Análogamente, se dice que una expresión lógica está en FND si está escrita como una disyunción en la cual los términos son conjunciones literales. Si la expresión tiene una subexpresión no atómica o negada no estaría en forma normal. Ambas representaciones son equivalentes respecto a significado, aunque se suele utilizar las FNC.

2.3.2. Árboles de decisión

En un árbol de decisión, cada nodo interno establece una condición o conjunto de condiciones sobre uno o varios atributos, representando en cada rama saliente el cumplimiento de una de esas condiciones. Cada hoja contiene una etiqueta de clase indicando la predicción. La clasificación de un ejemplo se lleva a cabo recorriendo el árbol desde la raíz hasta una de las hojas, siguiendo el camino determinado por el cumplimiento de las condiciones. El ejemplo se clasificará con la clase contenida en el nodo

hoja alcanzado finalmente. Normalmente, los árboles de decisión son binarios, y representan en cada nodo interno una única condición y en cada rama el cumplimiento o no de la misma.

Básicamente existen dos tipos de árboles de decisión [27]: *univariable*, donde cada nodo representa una condición sobre un único atributo, y *multivariable*, donde cada nodo contiene una expresión que involucra a uno o varios atributos a la vez [28]. Los árboles univariables son los más comunes, ya que son mucho más sencillos de generar e interpretar. A este tipo de árbol también se les denomina *paralelos*, puesto que los cortes que determinan sus condiciones son paralelos a los ejes definidos por los atributos. Uno de los sistema más conocidos que genera árboles de decisión paralelos es C4.5 [141], el cual va estableciendo cortes en los atributos para los valores que mayor ganancia de información proporcionan, representando cada uno de estos cortes con un nodo en el árbol.

Los árboles multivariables más conocidos son los llamados *oblicuos*, ya que los cortes que generan son sesgados respecto a los ejes, al contener en los nodos combinaciones lineales de los atributos. Por ejemplo, el sistema OC1 [131] genera árboles cuyos nodos contienen una expresión del tipo

$$\sum_{i=1}^{m} c_i a_i + c_{m+1} > 0 \tag{2.1}$$

donde a_i es el atributo i-ésimo; c_i es el coeficiente real correspondiente a a_i; y m es el número de atributos.

La figura 2.3 muestra los árboles paralelo y oblicuo generados para una base de datos con dos atributos (a_1 y a_2) y dos clases ($\times$ y $\bullet$). La figura también ilustra la distribución de los ejemplos de la base de datos en el plano definido por los dos atributos, representando cada ejemplo por su etiqueta de clase, así como los cortes que cada árbol establece sobre dicho plano. A partir de este ejemplo sencillo, podemos colegir que los árboles paralelos ofrecen menor complejidad desde el punto de vista de la interpretación humana, sobre todo en conjuntos de datos multidimensionales. Por el contrario, los oblicuos gozan de mayor versatilidad, ya que los paralelos pueden considerarse como un subconjunto de éstos. No obstante, los árboles oblicuos generados por OC1 ofrecen peor rendimiento que los paralelos obtenidos por C4.5 en la práctica [112].

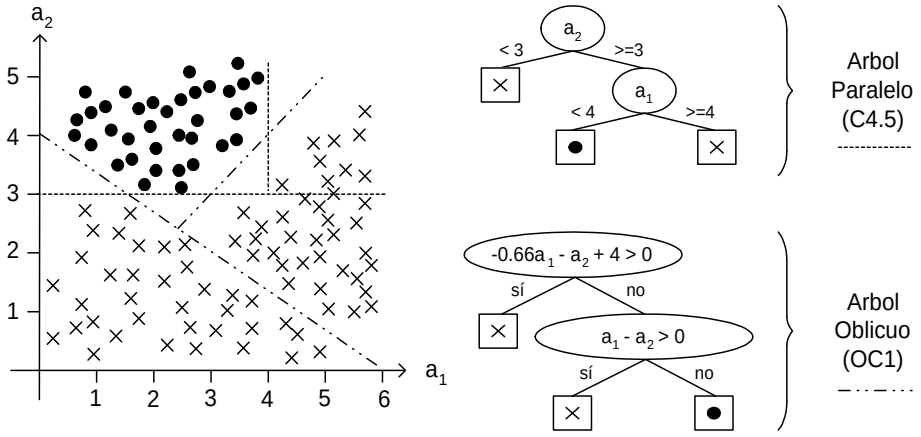


Figura 2.3: Árboles de decisión: Paralelo vs. Oblicuo.

Para medir la complejidad de un árbol de decisión, se cuentan el número de hojas del mismo, es decir, el número de etiquetas de clase que intervienen en el modelo. Dado que los nodos internos de estos árboles tienen siempre dos hijos, el número de hojas es $\frac{n+1}{2}$, siendo n el número total del nodos. Por similitud con las reglas de decisión, a esta medida se le suele denominar número de reglas, ya que determinan el número de posibles decisiones en la clasificación.

2.3.3. Reglas de decisión

En general, una regla de decisión es una regla del tipo "*Si* $\mathcal{P}$ *Entonces* $\mathcal{C}$", donde $\mathcal{P}$ es un predicado lógico sobre los atributos, cuya evaluación cierta implica la clasificación con etiqueta de clase $\mathcal{C}$. Desde el punto de vista de la interpretación humana, esta representación del conocimiento resulta a menudo más clara que los árboles de decisión, sobre todo en aplicaciones reales donde el número de nodos de éstos tienden a aumentar. Esto es debido tanto a la propia estructura como a las técnicas utilizadas para generar éstas. La construcción de los árboles de decisión se basa en una estrategia de *splitting* (división), esto es, dividir el conjunto de datos en dos subconjuntos considerando un único atributo seleccionado por una heurística particular. Por el contrario, el aprendizaje de reglas sigue una estrategia de *covering* (cobertura), esto es, encontrar condiciones de reglas teniendo en cuenta todos los atributos de forma que se cubra la mayor cantidad de ejemplos de una misma clase, y la menor del resto de las clases. En

la figura 2.4 podemos ver el conjunto de reglas de decisión que representa el mismo modelo que el árbol paralelo de la figura 2.3.

R_1 : *Si* $a_2 < 3$ *Entonces* $\times$
R_2 : *Si* $a_1 < 4$ *Y* $a_2 \geq 3$ *Entonces* $\bullet$
R_3 : *Si* $a_1 \geq 4$ *Y* $a_2 \geq 3$ *Entonces* $\times$

Figura 2.4: Reglas de Decisión.

De las diversas propuestas sobre aprendizaje y representación de reglas de decisión existentes en la bibliografía, las siguientes secciones enuncian las principales características de las más relevantes.

Listas de decisión

Las listas de decisión son una representación enmarcada dentro del aprendizaje de conceptos, es decir, sólo son aplicables a atributos discretos con valores Booleanos. Según los valores ($cierto$ o $falso$) de los atributos, un ejemplo puede ser clasificado como una instancia $positiva$ o $negativa$ del concepto a aprender.

Rivest [160] definió una lista de decisión como una lista L de pares (reglas):

$$(f_1, v_1), \ldots, (f_r, v_r)$$

donde f_i es una expresión Booleana sobre los atributos; cada v_i es un valor en $\{0, 1\}$, el cual denota el resultado de la clasificación ($negativa$ o $positiva$); por último, f_r es la función constante de valor $cierto$, siendo v_r el valor por defecto en la clasificación.

La evaluación de una lista de decisión para un ejemplo $x \in X_m$ es: $L(x) = v_j$, donde j es el menor índice tal que $f_j(x) = cierto$[1]. Así, la interpretación de una lista de decisión es similar a la de una regla "*Si* – *Entonces* –; *Sino, Si* – . . . *Sino* –".

Por ejemplo, consideremos la lista de decisión

$$(x_1\bar{x_3}, 0), (\bar{x_1}x_2x_5, 1), (\bar{x_3}\bar{x_4}, 1), (cierto, 0)$$

cuyas funciones Booleanas (f_j) son conjunciones de literales (x_i o $\bar{x_i}$). La evaluación del ejemplo $x = \{0, 0, 0, 0, 1\}$ resulta positiva ($L(0, 0, 0, 0, 1) = 1$) al cumplirse la función de

[1]Este valor siempre existe, ya que la última función (f_r) siempre es cierta.

la tercera regla y no cumplirse ninguna de las anteriores en la lista. Por el contrario, la evaluación del ejemplo $y = \{1, 0, 0, 0, 0\}$ resulta negativa ($L(1, 0, 0, 0, 0) = 0$) al ser cubierta por la primera regla.

Como demostró Rivest, en el marco del aprendizaje de conceptos, las k-DL (conjunto de listas de decisión de cláusulas conjuntivas con un máximo de k literales) son una generalización de los árboles de decisión de profundidad k (k-DT), así como de las variantes proposicionales (k-CNF y k-DNF), para $0 < k < m$, donde m es el número de atributos.

El principal inconveniente de las listas de decisión es que sólo pueden operar sobre atributos con dominios discretos y finitos, lo que restringe en gran medida su área de aplicación.

Reglas con excepciones

Las reglas con excepciones, denominadas en la literatura RDR (*Ripple-Down Rules*) [37], son listas donde cada regla está asociada a otras, sus excepciones. Una RDR es una regla del tipo "*Si* $\mathcal{P}$ *Entonces* $\mathcal{C}_1$, *excepción:* $\mathcal{Q}$ *Entonces* $\mathcal{C}_2$ ...", donde $\mathcal{Q}$ es un predicado que no ha de cumplirse para clasificar ejemplos con clase $\mathcal{C}_1$, de lo contrario, éstos serán etiquetados con clase $\mathcal{C}_2$. La figura 2.5 muestra un ejemplo de RDR, que indica el tipo de tarifa (*Alta*, *Media* o *Baja*) que una aseguradora adjudica a una determinada póliza según diversos parámetros.

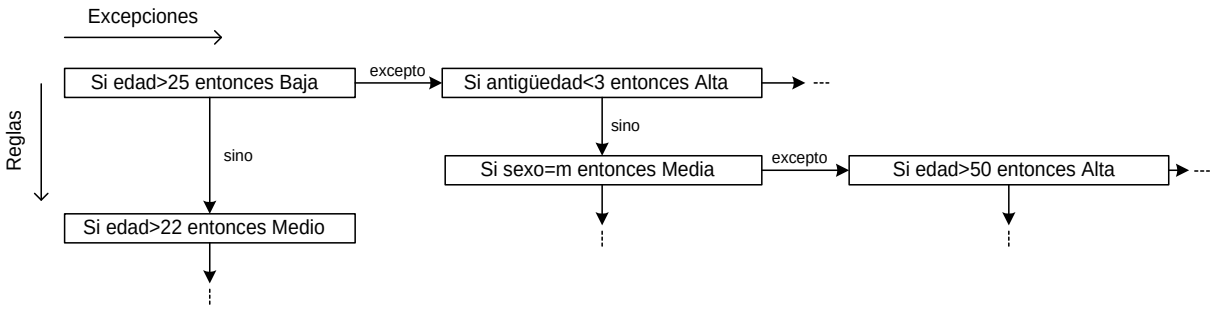


Figura 2.5: Reglas con excepciones (RDR).

El principal problema que presenta este tipo de representación es su difícil interpretación cuando el número de excepciones es elevado. En este sentido son similares a los árboles de decisión, ya el recorrido de las reglas y excepciones se asemeja a la

navegación por un árbol. No obstante, siempre es posible convertir una regla con excepciones en un conjunto de reglas de decisión [164]. Nótese que la regla "*Si* $\mathcal{P}$ *Entonces* $\mathcal{C}_1$, excepción: $\mathcal{Q}$ *Entonces* $\mathcal{C}_2$" es equivalente al conjunto "R_1: *Si* $\mathcal{P} \wedge \neg\mathcal{Q}$ *Entonces* $\mathcal{C}_1$; R_2: *Si* $\mathcal{P} \wedge \mathcal{Q}$ *Entonces* $\mathcal{C}_2$".

Existen en la bibliografía diversas propuestas de algoritmos de aprendizaje de RDR [48, 68, 105, 173], aunque fue Scheffer [164] quien desarrolló un álgebra para este tipo de reglas, así como diversas técnicas para mejorar los métodos de la inducción de RDR.

Reglas jerárquicas de decisión

Las reglas jerárquicas de decisión son conjuntos de reglas $\{R_1, R_2, \ldots, R_k\}$ donde existe una relación de orden preestablecido. Cada R_i es de la forma "*Si* $\mathcal{P}_i$ *Entonces* $\mathcal{C}_i$", de modo que un ejemplo es clasificado por la regla R_i con clase $\mathcal{C}_i$ si cumple el predicado $\mathcal{P}_i$ y no ha cumplido las condiciones establecidas por las $i-1$ reglas anteriores en la jerarquía, como expresa formalmente la ecuación 2.2,

$$e \dashv \mathcal{C}_i \Leftrightarrow \{\forall j : 1 \leq j < i \cdot \neg\mathcal{P}_j(e)\} \wedge \mathcal{P}_i(e) \tag{2.2}$$

donde $e \dashv \mathcal{C}_i$ indica la asignación de la etiqueta de clase $\mathcal{C}_i$ al ejemplo e; y $\mathcal{P}(e)$ expresa la evaluación del predicado lógico $\mathcal{P}$ con los valores de los atributos de e.

Normalmente, los predicados $\mathcal{P}$ se expresan en FNC, es decir, una conjunción de términos, donde cada término es una disyunción de cláusulas atómicas sobre los valores de un único atributo. La figura 2.6 muestra el esquema general de un conjunto de reglas jerárquicas, donde cada $cond_{ij}$ es la condición que la regla R_i establece para el atributo a_j. Si un ejemplo no satisface las condiciones exigidas por ninguna de las $k-1$ primeras reglas, la última regla R_k clasifica dicho ejemplo con una clase por defecto o simplemente indica "*clase desconocida*", contabilizándose éste caso como un error.

R_1 : *Si* $cond_{11}$ *Y* $cond_{12}$ *Y* ... *Y* $cond_{1n}$ *Entonces* $\mathcal{C}_1$
R_2 : *Si no, Si* $cond_{21}$ *Y* $cond_{22}$ *Y* ... *Y* $cond_{2n}$ *Entonces* $\mathcal{C}_2$
... : ... ...
R_k : *Si no,* $\mathcal{C}_k$ o "*clase desconocida*"

Figura 2.6: Conjunto de Reglas Jerárquicas de Decisión.

A diferencia de las listas de decisión [160], las reglas jerárquicas pueden establecer condiciones tanto para atributos discretos como continuos, aunque la sintaxis de estas condiciones es diferente según el caso. Esto aumenta la versatilidad de la representación. Si un atributo es discreto, la condición establecerá qué valores puede tomar el atributo de forma similar a las listas de decisión. En caso de atributos continuos, la condición establece el rango continuo donde el atributo puede tomar valores para satisfacer la condición. La representación del rango puede venir dada en forma intervalar ($a_j \in [li, ls]$) o mediante operadores relacionales ($li \leq a_j \leq ls$). Otro aspecto destacable frente a las listas de decisión es que estas reglas permiten clasificar ejemplos directamente con las etiquetas de clase, y no simplemente como ejemplos positivos o negativos.

Una propiedad de las reglas jerárquicas, que supone una ventaja respecto a otras representaciones, es que la jerarquía permite la existencia de regiones incluidas en otras en el espacio definido por los atributos. En general, el hecho de que las reglas jerárquicas establezcan condiciones sobre varios atributos a la vez, además de la posibilidad de inclusión de regiones, hacen que la complejidad de esta estructura de conocimiento sea habitualmente menor a la de otras representaciones, en términos tanto de legibilidad como de tamaño de la estructura. La figura 2.7 ilustra este aspecto mostrando el conjunto de reglas jerárquicas que representa el mismo modelo que el árbol de la figura 2.3 y las reglas no jerárquicas la figura 2.4. Como se puede observar, la representación resulta más simple y comprensible que en los otros dos casos.

R_1 : *Si* $a_1 \leq 4$ *Y* $a_2 \geq 3$ *Entonces* $\bullet$
R_2 : *Si no,* $\times$

Figura 2.7: Ejemplo de Reglas Jerárquicas de Decisión.

Entre los sistemas que generan reglas jerárquicas, destaca la familia de algoritmos COGITO [1], que implementan diversas variantes de este tipo de reglas [5], las cuales son estudiadas en la sección 3.3.2. Otros autores [104, 106] han propuesto algoritmos de generación de reglas jerárquicas dentro del aprendizaje de conceptos, aunque éstas son más próximas a las listas de decisión que a las reglas jerárquicas aquí expuestas.

2.3.4. Reglas Difusas

La teoría de subconjuntos difusos [183] relaja el concepto de pertenencia de una elemento a un conjunto. En la teoría tradicional, un elemento simplemente pertenece o no a un conjunto. Sin embargo, en la teoría de subconjuntos difusos, un elemento pertenece a un conjunto con un cierto grado de certeza. Aplicando esta idea, el uso de la lógica difusa permite un mejor tratamiento de la información cuando ésta es incompleta, imprecisa o incierta. Por ello, ha sido aplicada por muchos autores en tareas de clasificación, usando a menudo reglas difusas como representación del conocimiento [22, 88]. Estos sistemas son denominados tradicionalmente *Fuzzy Rule–Based Classification Systems*.

Las reglas difusas (*fuzzy rules*) presentan varias diferencias respecto a las reglas de decisión vistas anteriormente (*crisp rules*). Por un lado, las condiciones del antecedente de una regla difusa no son creadas en base a valores concretos ni rangos numéricos determinados, sino a etiquetas lingüísticas. Por ejemplo, los términos *frío*, *templado* y *caliente* son imprecisos, pero asociados a una semántica que les asigne un significado, podrían ser etiquetas lingüísticas para describir la temperatura de un objeto. Por otro lado, en el consecuente de la regla pueden aparecer una o varias etiquetas de clase, así como el grado de certeza o solidez asociado a cada clase en una regla concreta. Así, la estructura de una regla difusa es la siguiente:

$$R_k : \; Si \;\; a_1 \, es \, A_1^k \; Y \;\; \ldots \;\; Y \;\; a_m \, es \, A_m^k \;\; Entonces \;\; < \mathcal{C} >$$

donde cada a_i es un atributo del conjunto de datos; A_i^k son las etiquetas lingüísticas para el atributo a_i en la regla R_k; y $< \mathcal{C} >$ representa el consecuente de la regla, distinguiéndose tres tipos de reglas difusas de clasificación dependiendo de la información incluida en éste [39]:

1. Reglas con una única clase:

   $$R_k : \; Si \;\; a_1 \, es \, A_1^k \; Y \;\; \ldots \;\; Y \;\; a_m \, es \, A_m^k \;\; Entonces \; \mathbf{Clase \, es \, C_j}$$

   donde C_j es una etiqueta de clase.

2. Reglas con una clase y su grado de certeza en la clasificación:

$$R_k : Si \ \ a_1 \, es \, A_1^k \ Y \ \ldots \ Y \ a_m \, es \, A_m^k \ \ Entonces \ \mathbf{Clase \, es \, C_j \ con \ G^k}$$

donde G^k es el grado de certeza para la clase C_j en la clasificación de ejemplos cubiertos por la regla R_k.

3. Reglas con grado de certeza para todas las clases:

$$R_k : Si \ \ a_1 \, es \, A_1^k \ Y \ \ldots \ Y \ a_m \, es \, A_m^k \ \ Entonces \ (\mathbf{G_1^k}, \ldots, \mathbf{G_c^k})$$

donde G_j^k es el grado de certeza de la regla R_k al clasificar ejemplos con clase C_j.

Con respecto a la complejidad de la representación mediante reglas difusas, la forma en la que el ser humano expresa sus ideas es muy similar al modo en que los sistemas difusos representan el conocimiento. Por ello, algunos autores afirman que esta representación aumenta la comprensibilidad del modelo [138].

2.4. Preparación de los Datos

Las bases de datos utilizadas para la extracción de conocimiento son muy susceptibles de presentar ruido, ausencia de valores e inconsistencia, debido principalmente al gran tamaño de éstas. Por ello, es común preprocesar los datos antes de aplicar el algoritmo de minería para eliminar estas deficiencias y garantizar la calidad de los resultados. Existen numerosas técnicas de preprocesado que podemos agrupar es las siguientes categorías [85]:

- Depuración (*data cleaning*): Tratamiento de valores ausentes y eliminación de ruido.
- Trasformación: Conversión de los datos para mejorar el proceso de minería.
- Reducción: Eliminación de atributos y/o ejemplos.
- Discretización: Reducción de la cardinalidad de los atributos continuos.

Aunque las técnicas de discretización pueden incluirse en la categoría de transformación de los datos, éstas tienen especial interés para este trabajo, por lo que serán expuestas en un apartado independiente.

2.4.1. Depuración

Valores ausentes

La ausencia de valores en los atributos de algunos ejemplos de las bases de datos es relativamente frecuente, debido principalmente a fallos cometidos durante el proceso de adquisición de los datos, sea manual o automático. Aunque algunos métodos solventan este problema durante el proceso de aprendizaje, es común aplicar alguna técnica que trate estos ejemplos antes de ofrecerlos al algoritmo de minería de datos.

La técnica de tratamiento de valores ausentes más simple, aunque también la menos recomendable, consiste en eliminar aquellos ejemplos que presenten algún atributo sin valor. El mayor inconveniente de esta técnica es que se podría eliminar información útil para el aprendizaje contenida en los atributos correctos. Para poder mantener los ejemplos en el conjunto de datos, habría que rellenar los valores ausentes con algún valor válido. Una solución sencilla es asignar una constante, por ejemplo "*desconocido*", si el atributo es discreto, o ∞, si es continuo. Aunque esta solución es también muy simple y no elimina información, el algoritmo de aprendizaje podría interpretar erróneamente esas constantes y entender que son valores interesantes. Por esta razón, es recomendable sustituir las ausencias por valores cuya influencia en el aprendizaje sea mínima. En este sentido, la media o la moda, dependiendo si el atributo es continuo o discreto respectivamente, pueden ser valores más apropiados que una constante. Para que el valor de sustitución no sea único para todos los ejemplos con ausencias en un mismo atributo, la media o la moda no se calcula a partir de todos los datos, sino considerando sólo aquellos ejemplos que tienen la misma clase que el que se pretende completar. Aunque este método no es muy exacto, es la uno de los más populares. Finalmente, una técnica más precisa, aunque también más costosa computacionalmente, consiste en sustituir las ausencias por el valor más probable, aplicando algún clasificador (regresión, clasificador Bayesiano o inducción de árboles de decisión) para predecir dicho valor.

Ruido

Ruido es un error aleatorio o variación en el valor de un atributo, debido normalmente a errores en la medida del mismo. A diferencia de la ausencia de valores, el ruido es más difícil de detectar a simple vista, ya que son valores presentes en el conjunto de datos que pueden provocar que el algoritmo de minería de datos obtenga soluciones erróneas. Para mitigar los efectos del ruido en el aprendizaje se aplican las denominadas técnicas de suavizado (*smoothing*).

El método de suavizado más sencillo, conocido como *binning*, consiste en ordenar los valores de un atributo y distribuir tales valores en grupos o recipientes (*bins*) de igual número de valores o de igual rango, independientemente de los valores que contenga. Tras esta partición, se realiza un tratamiento local, sustituyendo los valores de cada grupo por la media, mediana o moda de dicho grupo. Aunque la aplicación de esta técnica suaviza los efectos del ruido, no garantiza la eliminación del mismo, ya que un atributo puede tomar valores que no correspondan a las características del ejemplo al que pertenece. Además, este método no corrige sólo los posibles outliers, sino que realiza cambios en todos los valores, por lo que no es muy recomendable.

Una estrategia más apropiada es aplicar algún método de clustering para detectar los outliers y poder tratarlos posteriormente. Un algoritmo adecuado para este propósito es el denominado SNN (*Similar Nearest Neighbours*) [11], ya que, al contrario que la mayoría de las técnicas de agrupamiento, éste realiza un tratamiento supervisado de los datos, obteniendo grupos de ejemplos de la misma clase. Una vez detectados los outliers, se elimina el ejemplo o bien se aplica algún método de sustitución similar a los descritos para el tratamiento de valores ausentes que introduzca al ejemplo en uno de los clusters de su misma clase.

2.4.2. Transformación

En ocasiones, la forma en que viene dada la información originalmente no es la más adecuada para adquirir conocimiento a parir de ella. En esas situaciones se hace necesario la aplicación algún tipo de trasformación para adecuar los datos al posterior

proceso de aprendizaje, como por ejemplo normalización o cambio de escala, discretización, generalización o extracción de atributos. Esta última trasformación está estrechamente relacionada con la selección de características detallada más adelante y consiste en construir nuevos atributos a parir de combinaciones de los originales. Muchos autores incluyen el tratamiento de los valores ausentes y el ruido dentro de las técnicas de transformación, sin embargo, en este apartado sólo se han considerado aquellas técnicas destinadas a trasformar los datos para mejorar el proceso de aprendizaje, y no a corregir errores en los mismos.

2.4.3. Reducción

En principio, cuanto más información esté disponible, mayor calidad tendrá el modelo de conocimiento generado a partir de ella. Por ello, es común que, en el área de la minería de datos, las bases de datos sean de gran volumen. Sin embargo, en muchos casos, el exceso se datos puede ser contraproducente debido principalmente a la existencia de información redundante o irrelevante para el problema que se desea solucionar. Ello ha motivado el desarrollo de técnicas para reducir el volumen de los datos, las cuales están orientadas fundamentalmente hacia dos objetivos: *selección de atributos* (eliminación de atributos no relevantes) y *editado* (reducción del número de ejemplos).

Editado

Las técnicas de editado tienen como objetivo reducir el número de ejemplos de un conjunto de datos $\mathcal{D}$, obteniendo un subconjunto $\mathcal{S}$ que contenga el mismo conocimiento que $\mathcal{D}$. Para ello se pueden seguir dos estrategias: formar $\mathcal{S}$ a partir de la selección o rechazo de ejemplos contenidos en $\mathcal{D}$, siendo estrictamente $\mathcal{S} \subseteq \mathcal{D}$; o bien construir $\mathcal{S}$ en base a prototipos [33] o reglas [51, 163], que representen grupos de ejemplos de $\mathcal{D}$, aunque dichos prototipos no coincidan con ejemplos de $\mathcal{D}$.

Evidentemente, la búsqueda del subconjunto $\mathcal{S}$ se lleva a cabo aplicando algún tipo de heurística, ya que una búsqueda exhaustiva es impracticable por su elevado coste computacional. Dependiendo del sentido de esta búsqueda, la técnicas de reducción de ejemplos se clasifican en: *incrementales*, donde el conjunto $\mathcal{S}$ es inicialmente vacío y

se le van añadiendo ejemplos de $\mathcal{D}$ seleccionados según un determinado criterio; y *decrementales*, donde inicialmente $\mathcal{S} = \mathcal{D}$ y se van eliminando ejemplos o generalizando éstos en reglas o prototipos. Aunque los métodos decrementales suelen ser más costosos computacionalmente, los incrementales son más sensibles al orden de los ejemplos en el conjunto $\mathcal{D}$.

La bibliografía recoge un amplio catálogo de algoritmos de reducción de ejemplos. Entre los incrementales, podemos destacar CNN [86], que supuso el primer método de editado, o IB3 [12]. Los métodos decrementales son más populares, entre los que podemos resaltar algunos como RNN [69], ENN [178], SNN [159], ENN [178], MULTIEDIT [46], SHRINK [101], VSM [116] y EPO [9].

Selección de atributos

Los algoritmos de selección de características tienen dos objetivos principales: reducir el coste computacional asociado tanto al aprendizaje como al propio modelo de conocimiento generado (eliminando atributos irrelevantes o redundantes) y aumentar la precisión de dicho modelo (eliminando atributos perjudiciales para el aprendizaje).

Para llevar a cabo su objetivo, los métodos de selección realizan una búsqueda sobre el espacio de características, aplicando una función criterio que evalúa la calidad del subconjunto seleccionado. Dicha búsqueda suele ser heurística, ya que una búsqueda exhaustiva supone un problema combinatorio, resultando ésta enormemente costosa. Aunque existen diversos criterios de clasificación de estas técnicas, generalmente son agrupadas en dos categorías según la estrategia de evaluación: *wrappers* [107, 115], donde la función criterio utilizada es el propio conjunto de reglas generadas por el algoritmo de aprendizaje que posteriormente se usará en la clasificación; y *filtros*, cuya función de criterio es independiente del algoritmo de aprendizaje, usando medidas de distancia, información o dependencia. La figura 2.8 ilustra el esquema general del proceso de selección de atributos.

En general, los filtros son más rápidos que los *wrappers*, ya que no necesitan hacer llamadas al algoritmo de aprendizaje para evaluar la calidad de los subconjuntos. Por esta razón son los más utilizados en la práctica, sobre todo cuando la base de datos

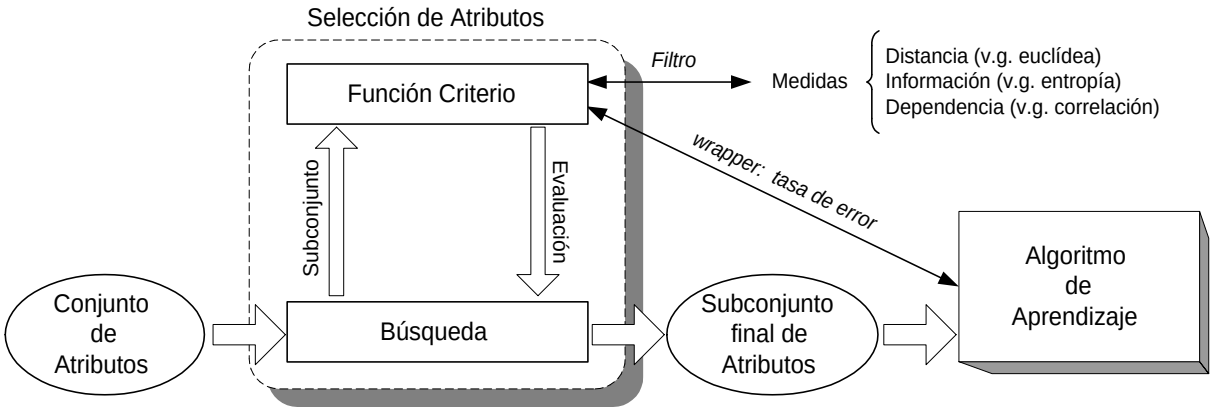


Figura 2.8: Selección de Atributos.

tiene un número elevado de dimensiones. Entre estos algoritmos de selección de atributos y análisis de influencia de éstos destacan FOCUS [13], RELIEF [102, 109], LVF [166] y CHI2 [114].

2.4.4. Discretización

Un gran número de algoritmos de aprendizaje operan exclusivamente con espacios discretos. Sin embargo, muchas bases de datos contienen atributos de dominio continuo, lo que hace imprescindible la aplicación previa de algún método que reduzca la cardinalidad del conjunto de valores que estas características pueden tomar, dividiendo su rango en un conjunto finito de intervalos. Esta trasformación de atributos continuos en discretos se denomina *discretización*.

Al igual que los métodos de aprendizaje, los algoritmos de discretización suelen ser clasificados como *supervisados*, donde la clase de los ejemplos es considerada en el proceso de discretización como atributo de decisión; y *no supervisados*, donde no se tiene en cuenta la clase o bien ésta es tratada como un atributo normal. Evidentemente, el problema de la discretización no supervisada es que suele producir resultados poco apropiados si posteriormente se va a aplicar un algoritmo de aprendizaje supervisado. Por ejemplo, en clasificación, puede provocar pérdida de precisión en las reglas, debido a la posible inclusión de valores con diferente clase en los mismos intervalos.

Además del anterior, es común clasificar los métodos de discretización según otros

dos criterios [52]: *global* vs. *local* y *estático* vs. *dinámico*. Los métodos globales son aplicados una única vez al conjunto de datos antes de ejecutar el algoritmo de aprendizaje, teniendo en cuenta todo el espacio definido por los atributos. Por el contrario, los métodos locales son aplicados a subconjuntos de ejemplos asociados con regiones en el espacio generadas durante el aprendizaje (*v.g.* C4.5 [141]). Por otro lado, un discretizador se considera estático cuando sólo tiene en cuenta un atributo a la vez, estableciendo los intervalos independientemente del resto de atributos del conjunto de datos. En contraste, los métodos dinámicos realizan una búsqueda a través de todos los atributos simultáneamente, considerando los efectos que un determinado corte o intervalo tiene en el resto de características para así poder establecer dependencias entre éstas.

Desde el punto de vista de este trabajo, los métodos de discretización más interesantes son los *supervisados*, en concordancia con el tipo de aprendizaje posterior; *globales*, ya que forma parte del preprocesado y es independiente del algoritmo de minería; y *estáticos*, por ser más sencillos, dejando las detección de posibles dependencias al algoritmo de aprendizaje.

Discretización en intervalos de igual anchura e igual frecuencia

La división en *intervalos de igual anchura* es la más simple de las técnicas de discretización existentes. Se trata de un método no supervisado global que divide el rango de una atributo en k intervalos de igual tamaño. Aunque en para ciertos conjuntos de datos puede resultar efectiva, esta técnica es muy sensible a los *outliers* [30], ya que éstos pueden ampliar el rango del atributo, provocando que los valores válidos se concentren sólo en algunos intervalos, quedando muchos intervalos vacíos. Este problema se solventa utilizando la discretización en *intervalos de igual frecuencia*. Si en el conjunto de datos existen N ejemplos, este método divide el rango del atributo en k intervalos, cada uno de los cuales contiene $\frac{N}{k}$ valores, posiblemente repetidos y todos incluidos en el conjunto de datos.

Ambas técnicas presentan principalmente dos inconvenientes. El primero es el carácter no supervisado de ambas, lo cual puede provocar los problemas anteriormente comentados. El otro inconveniente es la necesidad de determinar a priori el número final de intervalos (k).

ChiMerge

R. Kerber [100] introduce un método de discretización supervisado global que aplica una heurística estadísticamente justificada. El método comienza con un intervalo por valor, y utiliza el test χ^2 para determinar cuándo dos intervalos adyacentes deben ser unidos según las frecuencias relativas de las clases de tales intervalos. El proceso de unión es controlado por un umbral, que es el máximo valor de χ^2 que garantiza la fusión de dos intervalos. Así, si dos intervalos adyacentes presentan un valor de χ^2 superior al umbral preestablecido, éstos se consideran significativamente distintos y, por tanto, no son unidos. El cálculo de χ^2 se realiza según la ecuación 2.3

$$\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{C} \frac{(A_{ij} - E_{ij})^2}{E_{ij}} \tag{2.3}$$

donde C es el número de clases; A_{ij} es el número de ejemplo con clase j en el i-ésimo intervalo; y E_{ij} es la frecuencia esperada de A_{ij}. El mayor inconveniente de este método es determinar el umbral apropiado, ya que un valor excesivamente pequeño provocaría la creación de demasiados intervalos, mientras que un valor demasiado elevado produciría pocos intervalos aunque con un alto grado de impureza[2].

StatDisc

StatDisc es un método heurístico supervisado y global propuesto por Richeldi y Rossotto [150] que, al igual que *ChiMerge*, se basa en un test estadístico para llevar a cabo la discretización. Se trata de un método *bottom-up* que crea una jerarquía de discretizaciones usando el test Φ para unir intervalos. *StatDisc* es más general que *ChiMerge*, ya que considera N intervalos adyacentes en vez de sólo dos. En cada paso, el método va uniendo grupos de intervalos adyacentes y obteniendo diferentes discretizaciones, terminando el proceso al alcanzar un umbral preestablecido para Φ. Una vez obtenida la jerarquía de discretizaciones, el usuario debe seleccionar la más apropiada para el problema que desea resolver.

Al igual que todas las técnicas parametrizadas, el inconveniente de ésta radica en fijar los valores adecuados para el parámetro N y el umbral para Φ.

[2]La impureza se refiere al número de clases distintas dentro de un intervalo. A mayor número de clases distintas, mayor es la impureza del intervalo.

Métodos basados en criterios de entropía mínima

Existen en la literatura numerosos métodos que aplican criterios de minimización de la entropía para discretizar atributos continuos [34, 137, 182]. Entre estos métodos destacan las propuestas de Catlett [31] y Fayyad e Irani [57], que utilizan la entropía de la clase para establecer los límites de los intervalos (cortes) en los que se dividirá el rango de un atributo continuo.

Definición 2.16 (Entropía). *La entropía es la medida del desorden de un sistema mediante la incertidumbre existente ante un conjunto de casos, del cual se espera uno sólo. Sea $\mathcal{D}$ un conjunto de datos etiquetados con clases del conjunto $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ y $frec(\mathcal{C}_i, \mathcal{D})$ el número de ejemplos de $\mathcal{D}$ con clase $\mathcal{C}_i$. Entonces se define la entropía del conjunto $\mathcal{D}$ como*

$$Ent(\mathcal{D}) = -\sum_{i=1}^{k} \frac{frec(\mathcal{C}_i, \mathcal{D})}{|\mathcal{D}|} \times \log_2 \left(\frac{frec(\mathcal{C}_i, \mathcal{D})}{|\mathcal{D}|} \right) \tag{2.4}$$

donde $\frac{frec(\mathcal{C}_i, \mathcal{D})}{|\mathcal{D}|}$ es la probabilidad de que se dé un ejemplo con clase $\mathcal{C}_i$, y $\log_2 \left(\frac{frec(\mathcal{C}_i, \mathcal{D})}{|\mathcal{D}|} \right)$ es la información que transmite un ejemplo de clase $\mathcal{C}_i$. La entropía es máxima cuando todas las cases presentan la misma proporción.

Usando la notación de Fayyad e Irani, dado conjunto de datos $\mathcal{S}$, un atributo $\mathcal{A}$, y un corte $\mathcal{T}$, la entropía de clase de los intervalos $\mathcal{S}_1$ y $\mathcal{S}_2$ inducidos por $\mathcal{T}$ es calculada como

$$E(\mathcal{A}, \mathcal{T}; \mathcal{S}) = \frac{|\mathcal{S}_1|}{|\mathcal{S}|} \times Ent(\mathcal{S}_1) + \frac{|\mathcal{S}_2|}{|\mathcal{S}|} \times Ent(\mathcal{S}_2) \tag{2.5}$$

donde $|\mathcal{S}|$, $|\mathcal{S}_1|$ y $|\mathcal{S}_2|$ indican el número de ejemplos de cada conjunto y $Ent(\cdot)$ es la entropía, la cual es calculada mediante la ecuación 2.4. Así, para cada atributo se selecciona el corte $\mathcal{T}$ entre todas las posibles particiones que minimiza $E(\mathcal{A}, \mathcal{T}; \mathcal{S})$. Una vez establecido el corte, se aplica recursivamente esta heurística a cada una de las dos particiones resultantes ($\mathcal{S}_1$ y $\mathcal{S}_2$) hasta que se satisface un criterio de parada. La diferencia entre el algoritmo de Catlett y la propuesta de Fayyad e Irani radica en ese criterio. Mientras el método de Catlett se detiene cuando el número de ejemplos en un intervalo es suficientemente pequeño o el número de intervalos alcanza un máximo, Fayyad e Irani usan el *principio de longitud de descripción mínima* como condición de parada,

deteniendo el algoritmo si y sólo si

$$Ganancia(\mathcal{A},\mathcal{T};\mathcal{S}) < \frac{\log_2(|\mathcal{S}|-1)}{|\mathcal{S}|} + \frac{\Delta(\mathcal{A},\mathcal{T};\mathcal{S})}{|\mathcal{S}|} \tag{2.6}$$

donde

$$Ganancia(\mathcal{A},\mathcal{T};\mathcal{S}) = Ent(\mathcal{S}) - E(\mathcal{A},\mathcal{T};\mathcal{S})$$
$$\Delta(\mathcal{A},\mathcal{T};\mathcal{S})|\mathcal{S}| = \log_2(3^k-2) - (k\cdot Ent(\mathcal{S}) - k_1\cdot Ent(\mathcal{S}_1) - k_2\cdot Ent(\mathcal{S}_2))$$

y k, k_1 y k_2 son el número de clases distintas de $\mathcal{S}$, $\mathcal{S}_1$ y $\mathcal{S}_2$ respectivamente. Podemos colegir que este criterio puede producir intervalos muy desiguales para un mismo atributo, ya que, una vez establecido un corte, la evaluación de los dos subespacios resultantes es independiente. De este modo, zonas del espacio que presenten una baja entropía serán divididas muy pocas veces, dando intervalos relativamente grandes, mientras que en otras zonas con alta entropía, los cortes serán mucho más próximos.

1-Rules

Robert C. Holte [94] describe un clasificador muy simple denominado 1-$Rules$ (en adelante *1R*), el cual induce reglas sobre un único atributo, de ahí su nombre. Para poder tratar con atributos continuos, *1R* integra un algoritmo de discretización global supervisado que a menudo es denominado como el propio clasificador. Este método disminuye la cardinalidad de un atributo continuo dividiendo el rango de éste en intervalos que contengan una clase ampliamente mayoritaria. Para ello, ordena el conjunto de datos por el atributo a discretizar e intenta dividir el rango en intervalos tan puros como sea posible, según la definición de *clase óptima*.

Definición 2.17 (Clase óptima). *La clase óptima de un intervalo es la clase que más apariciones tiene en dicho intervalo. Análogamente, la clase óptima de un valor es aquella que más veces se repite para tal valor en el conjunto de datos. Al contrario que la de la clase mayoritaria (definición 4.4), un intervalo o valor puede tener más de una clase óptima.*

Para evitar obtener intervalos con un único valor, Holte [95] define el parámetro de usuario *SMALL* como el mínimo número de valores distintos que un intervalo puede contener. Así, el conjunto de valores ordenados es recorrido, estableciendo puntos

de corte que son los límites de intervalos, obligando a que satisfagan las siguientes condiciones:

1. Hay al menos una clase óptima para más de un número mínimo (*SMALL*) de valores en el intervalo. (Esta restricción no se aplica al último intervalo).

2. Un intervalo no puede tener la misma clase óptima que el intervalo o valor inmediatamente anterior o posterior.

El discretizador *1R* maximiza la pureza de los intervalos obtenidos respetando el número mínimo de valores por intervalo impuesto por el parámetro *SMALL*. Tras los estudios experimentales realizados en [94] y [95], Holte sugiere que *SMALL*=6 es un valor apropiado para la mayoría de las bases de datos, aunque también apuntan que si éstas contienen un número reducido de ejemplos, dicho parámetro debe ser establecido en 3. No obstante, estos valores no siempre son los más favorables [72], sino que depende del número de ejemplos y de la distribución de éstos en las clases. Para obtener el valor óptimo de *SMALL* sería necesario la realización de múltiples experimentos.

Otros métodos de discretización

El problema de la discretización de atributos continuos ha sido ampliamente estudiado en la bibliografía, presentándose una gran variedad de métodos que aplican diferentes estrategias de programación como back-tracking [132], programación dinámica [117], divide y vencerás [171] o algoritmos genéticos [16], entre otras.

Por otra parte, muchos algoritmos de aprendizaje han sido aplicados como discretizadores, ya sean supervisados, como por ejemplo C4.5 o *1R*, o no supervisados, como las técnicas de clustering. En general, cualquier clasificador podría ser empleado como método de discretización usando un único atributo durante el aprendizaje. También ciertas técnicas de preprocesado, como el tratamiento del ruido [85] o la selección de atributos [114], están estrechamente relacionadas con la reducción de la cardinalidad, usándose éstas en tareas de discretización aunque originalmente fueran diseñadas para resolver otro problema.

2.5. Métodos de Aprendizaje Supervisado

2.5.1. Técnicas Estadísticas

La Estadística fue inicialmente la rama de la ciencia encargada de extraer información válida de un conjunto de datos, obteniendo inferencias basadas en el cálculo de probabilidades. Los fundamentos teóricos de estas técnicas han sido la base natural de muchos de los métodos de minería de datos empleadas hoy en día. En particular, el problema de la clasificación de nuevos casos a partir de una muestra de datos etiquetados ha sido ampliamente estudiado por esta disciplina, existiendo una extensa literatura al respecto.

Entre las técnicas estadísticas usadas en el campo del aprendizaje, podemos destacar el estudio de las correlaciones entre atributos, el análisis factorial [38], componentes principales [129], discriminante lineal [60] y el método Bayesiano [19]. Esta última técnica merece ser comentada más detenidamente, ya que ha sido empleada como clasificador de referencia por un gran número de autores.

Clasificador Bayesiano

Naïve–Bayes es una técnica de clasificación descriptiva y predictiva basada en la teoría de la probabilidad del análisis de T. Bayes [19], que data de 1763. Esta teoría supone un tamaño de la muestra asintóticamente infinito e independencia estadística entre variables independientes, refiriéndose en nuestro caso a los atributos, no a la clase. Con estas condiciones, se puede calcular las distribuciones de probabilidad de cada clase para establecer la relación entre los atributos (variables independientes) y la clase (variable dependiente). Concretamente, dado el ejemplo $e = (e_1, \ldots, e_m)$, donde e_k es el valor observado para el j-ésimo atributo, la probabilidad a posteriori de que ocurra la clase $\mathcal{C}_i$ viene dada por la regla de Bayes,

$$P(\mathcal{C}_i|e_1, \ldots, e_m) = \frac{P(\mathcal{C}_i) \prod_{k=1}^{m} P(e_k|\mathcal{C}_i)}{P(e_1, \ldots, e_m)} \tag{2.7}$$

donde $P(\mathcal{C}_i)$ es la proporción de la clase $\mathcal{C}_i$ en el conjunto de datos; e igualmente, $P(e_k|\mathcal{C}_i)$ se estima a partir de la proporción de ejemplos con valor e_k cuya clase es $\mathcal{C}_i$.

Como podemos deducir, el cálculo de $P(e_k|\mathcal{C}_i)$ obliga a que los valores e_k sean discretos, por lo que si existe algún atributo continuo, éste debe ser discretizado previamente.

Aplicando la ecuación 2.7, la clasificación de un nuevo ejemplo e se lleva a cabo calculando las probabilidades condicionadas de cada clase y escogiendo aquella con mayor probabilidad. Formalmente, si $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_d\}$ es el conjunto de clases existentes, el ejemplo e será clasificado con aquella clase $\mathcal{C}_i$ que satisface la expresión 2.8.

$$\forall j \neq i \cdot P(\mathcal{C}_i|e_1, \ldots, e_m) > P(\mathcal{C}_j|e_1, \ldots, e_m) \tag{2.8}$$

Como se puede observar, el clasificador bayesiano es un método sencillo y rápido. Además, puede demostrarse teóricamente que maximiza la exactitud de la predicción de manera óptima. Sin embargo, la suposición de independencia estadística de las variables es una limitación importante, ya que este hecho es relativamente infrecuente.

2.5.2. Vecino Más Cercano

Las técnicas de vecinos más cercanos (*NN, Nearest Neighbours*) basan su criterio de aprendizaje en la hipótesis de que los miembros de una población suelen compartir propiedades y características con los individuos que los rodean, de modo que es posible obtener información descriptiva de un individuo mediante la observación de sus vecinos más cercanos.

Los fundamentos de la clasificación por vecindad fueron establecidos por E. Fix y J. L. Hodges [61, 62] a principio de los años 50. Sin embargo, no fue hasta 1967 cuando T. M. Cover y P. E. Hart [42] enuncian formalmente la regla del vecino más cercano y la desarrollan como herramienta de clasificación de patrones. Desde entonces, este algoritmo se ha convertido en uno de los métodos de clasificación más usados [40, 41, 43, 53].

La regla de clasificación NN se resume básicamente en el siguiente enunciado: *Sea* $\mathcal{D} = \{e_1, \ldots, e_N\}$ *un conjunto de datos con* N *ejemplos etiquetados, donde cada ejemplo* e_i *contiene* m *atributos* $(e_{i1}, \ldots, e_{im})$*, pertenecientes al espacio métrico* $\mathcal{E}^m$*, y una clase* $\mathcal{C}_i \in \{\mathcal{C}_1, \ldots, \mathcal{C}_d\}$*. La clasificación de un nuevo ejemplo* e' *cumple que*

$$e' \dashv \mathcal{C}_i \Leftrightarrow \forall j \neq i \cdot d(e', e_i) < d(e', e_j) \tag{2.9}$$

donde $e' \dashv \mathcal{C}_i$ *indica la asignación de la etiqueta de clase* $\mathcal{C}_i$ *al ejemplo* e'*; y* d *expresa una distancia definida en el espacio* m*-dimensional* $\mathcal{E}^m$*.*

Así, un ejemplo es etiquetado con la clase de su vecino más cercano según la métrica definida por la distancia d. La elección de esta métrica es primordial, ya que determina qué significa más cercano. La aplicación de métricas distintas sobre un mismo conjunto de entrenamiento puede producir resultados diferentes. Sin embargo, no existe una definición previa que indique si una métrica es buena o no. Esto implica que es el experto quien debe seleccionar la medida de distancia más adecuada.

La regla NN puede generalizarse calculando los k vecinos más cercanos y asignando la clase mayoritaria entre esos vecinos. Tal generalización se denomina k–NN. Este algoritmo necesita la especificación a priori de k, que determina el número de vecinos que se tendrán en cuenta para la predicción. Al igual que la métrica, la selección de un k adecuado es un aspecto determinante. El problema de la elección del k ha sido ampliamente estudiado en la bibliografía. Existen diversos métodos para la estimación de k [177]. Otros autores [54] han abordado el problema incorporando pesos a los distintos vecinos para mitigar los efectos de la elección de un k inadecuado. Otras alternativas [157] intentan determinar el comportamiento de k en el espacio de características para obtener un patrón que determine a priori cuál es el número de vecinos más adecuado para clasificar un ejemplo concreto dependiendo de los valores de sus atributos. En un estudio más reciente, F. J. Ferrer et al. [59] desarrollan un algoritmo de clasificación NN no parametrizado que adapta localmente el valor k.

El algoritmo k–NN se engloba dentro de las denominadas técnicas de aprendizaje perezoso (*lazy learning*), ya que no genera una estructura de conocimiento que modele la información inherente del conjunto de entrenamiento, sino que el propio conjunto de datos representa el modelo. Cada vez que se necesita clasificar un nuevo ejemplo, el algoritmo recorre el conjunto de entrenamiento para obtener los k vecinos y predecir su clase. Esto hace que el algoritmo sea computacionalmente costoso tanto en tiempo, ya que necesita recorrer la totalidad de los ejemplos en cada predicción, como en espacio, por la necesidad de mantener almacenado todo el conjunto de entrenamiento.

Pese a los numerosos inconvenientes respecto a la eficiencia (coste computacional) y la eficacia (elección de la métrica y el k adecuados), k–NN tiene en general un buen

comportamiento. Cover y Hart [42] demostraron que, cuando el número de ejemplos tiende a infinito, el error asintótico de NN está acotado superiormente por el doble del error de Bayes (óptimo).

2.5.3. Inducción de Árboles de Decisión

Los árboles de decisión, descritos en la sección 2.3.2, es una de las formas más sencillas de representación del conocimiento adquirido. Dentro de los sistemas basados en árboles de decisión, habitualmente denominados TDIDT (*Top Down Induction of Decision Trees*), se pueden destacar dos familias o grupos: la familia ID3, cuyos máximos representantes son el propio algoritmo ID3 propuesto por Quinlan [141] y el sistema CLS de Hunt et al. [96]; y la familia de árboles de regresión, cuyo exponente más significativo es CART, desarrollado por Breiman et al. [26].

Los TDIDT se caracterizan por utilizar una estrategia de divide y vencerás descendente, es decir, partiendo de los descriptores hacia los ejemplos, dividen el conjunto de datos en subconjuntos siguiendo un determinado criterio de división. A medida que el algoritmo avanza, el árbol crece y los subconjuntos de ejemplos son menos numerosos.

ID3 puede considerarse como una versión preliminar de C4.5, el cual resuelve algunos inconvenientes de su antecesor sobre el uso de atributos continuos, el tratamiento de valores ausentes y el proceso de poda. De los sistemas TDIDT, los pertenecientes a la familia ID3 son los más referenciados en el campo del aprendizaje, por lo que serán expuestos con más detalle a continuación.

ID3

El método de clasificación experimental ID3 (*Induction Decision Trees*), desarrollado por J. R. Quinlan [139, 140, 141], genera un árbol de decisión paralelo de forma recursiva, aplicando un criterio de división basado en el concepto de medida de la información de Shannon. Cada nodo interno de dicho árbol contiene un test sobre uno de los atributos, de cuyo valor dependerá el camino a seguir para clasificar un ejemplo, y cada hoja contiene una etiqueta de clase. Así, la clasificación de un ejemplo se lleva a cabo recorriendo el árbol desde la raíz hasta una de las hojas que determinará la clase del mismo.

Inicialmente, el algoritmo toma todo el conjunto de datos $\mathcal{D}$. Si todos los ejemplos pertenecen a una misma clase $\mathcal{C}$, el proceso finaliza, insertando un nodo hoja con dicha clase. En caso contrario, se selecciona aquel atributo $\mathcal{A}$ que mejor divide el conjunto de datos y se inserta un nodo con dicho atributo para establecer un test. Una vez creado el nodo, para cada valor distinto $\mathcal{A}_i$ del atributo $\mathcal{A}$, se traza un arco y se invoca recursivamente al algoritmo para generar el subárbol que clasifica los ejemplos de $\mathcal{D}$ que cumplen que $\mathcal{A} = \mathcal{A}_i$. Dicha invocación es realizada sin tener en cuenta el atributo $\mathcal{A}$ y substrayendo del conjunto de datos $\mathcal{D}$ todos aquellos ejemplos donde $\mathcal{A} \neq \mathcal{A}_i$. El proceso se detiene cuando todas las instancias de un conjunto pertenecen a la misma clase.

ID3 utiliza una propiedad estadística denominada *ganancia de información* como heurística de selección de atributos para fijar un test. Esta propiedad no es más que la reducción esperada de la entropía (desorden) de los datos al conocer el valor de un atributo. Así, el atributo $\mathcal{A}$ seleccionado para determinar la división será aquel que mayor ganancia obtenga respecto al conjunto $\mathcal{D}$, según la ecuación 2.10,

$$Ganancia(\mathcal{D}, \mathcal{A}) = Ent(\mathcal{D}) - \sum_{i=1}^{|\mathcal{A}|} \frac{|\mathcal{D}(\mathcal{A}_i)|}{|\mathcal{D}|} \times Ent(\mathcal{D}(\mathcal{A}_i)) \tag{2.10}$$

donde $|\mathcal{A}|$ es el número de valores distintos de del atributo $\mathcal{A}$; $Ent(\cdot)$ es la entropía, definida por la ecuación 2.4; $\mathcal{D}(\mathcal{A}_i)$ es el subconjunto de $\mathcal{D}$ para el cual $\mathcal{A} = \mathcal{A}_i$, siendo $|\mathcal{D}(\mathcal{A}_i)|$ su cardinal; y $|\mathcal{D}|$ es el número total de ejemplos.

Pese a su simplicidad y bajo coste computacional, ID3 presenta inconvenientes importantes, algunos de los cuales son corregidos por su sucesor C4.5. Los más evidentes son la incapacidad para trabajar con atributos continuos y tratar valores ausentes. Sin embargo, presenta una serie de problemas que afectan directamente a la precisión del árbol generado. En primer lugar, la heurística usada para establecer los test es propensa a seleccionar aquellos atributos con mayor número de valores distintos, ya que a mayor número de particiones, la entropía de cada subconjunto tiende a ser menor. En segundo lugar, ID3 resulta muy vulnerable a la presencia de ruido e inconsistencia en los datos, lo cual ocasiona la generación de *hojas muertas* que clasifican ejemplos de más de una clase. Por último, la limitada capacidad de generalización del algoritmo provoca la aparición de *hojas vacías*, que no clasifican ningún ejemplo del conjunto de

entrenamiento y, por lo tanto, no se les asigna etiqueta de clase. Esto implica que no se podrán realizar predicciones sobre aquellos ejemplos incluidos en las zonas del espacio cubiertas por hojas vacías por no aparecer en el conjunto de entrenamiento.

Por otra parte, el algoritmo obliga a que todos los ejemplos sean clasificados correctamente. Esto, unido a los problemas de generalización y ruido, hace que ID3 produzca árboles de mucha profundidad sin que esto beneficie a la precisión de los mismos. Quinlan [143] propuso como solución un método de poda para reducir el error y el tamaño de los árboles. Dicho método sustituía un subárbol completo por una hoja etiquetada con la case mayoritaria del subárbol si ésta sustitución mejoraba o al menos iguala la clasificación original.

C4.5

El algoritmo C4.5 fue propuesto por Quinlan [145] a finales de los años 80 para mejorar las carencias de su predecesor ID3. Desde entonces, ha sido uno de los sistemas clasificadores más referenciados en la bibliografía, principalmente debido a su extremada robustez en un gran número de dominios y su bajo coste computacional.

C4.5 introduce principalmente las siguientes mejoras:

1. Trata eficazmente los valores desconocidos calculando la ganancia de información para los valores presentes.

2. Maneja los atributos continuos, aplicando una discretización previa.

3. Corrige la tendencia de ID3 a seleccionar los atributos con muchos valores distintos para establecer los test cambiando el criterio de división.

C4.5 produce un árbol de decisión similar al de ID3, con la salvedad de que puede incluir condiciones sobre atributos continuos. Así, los nodos internos pueden contener dos tipos de test según el dominio del atributo seleccionado para la partición. Si el atributo $\mathcal{A}$ es discreto, la representación es similar a la de ID3, presentando un test con una condición de salida (rama $\mathcal{A} = v_i$) por cada valor v_i diferente del atributo. Por contra, si el atributo $\mathcal{A}$ es continuo, el test presenta dos únicas salidas, $\mathcal{A} \leq Z$ y $\mathcal{A} > Z$, que comparan el valor de $\mathcal{A}$ con el umbral Z. Para calcular Z, se aplica

un método similar al usado en [26], el cual ordena el conjunto de k valores distintos del atributo $\mathcal{A}$ presentes en el conjunto de entrenamiento, obteniendo el conjunto de valores $\{v_1, v_2, \ldots, v_k\}$. Cada par de valores consecutivos aporta un posible umbral $Z = \frac{v_i+v_{i+1}}{2}$, teniendo en total $k-1$ umbrales, donde k es como mucho igual al número de ejemplos. Una vez calculados los umbrales, C4.5 selecciona aquel que maximiza el criterio de separación.

Como se mencionó anteriormente, el criterio de maximización de la ganancia de información usado en ID3 produce un sesgo hacia los atributos que presentan muchos valores distintos. C4.5 resuelve este problema usando la razón de ganancia (*gain ratio*) como criterio de separación a la hora de establecer un test. Esta medida tiene en cuenta tanto la ganancia de información como las probabilidades de los distintos valores del atributo. Dichas probabilidades son recogidas mediante la denominada información de separación (*split information*), que no es más que la entropía del conjunto de datos $\mathcal{D}$ respecto a los valores del atributo $\mathcal{A}$ en consideración, siendo calculada como

$$InformacionDeSeparacion(\mathcal{D}, \mathcal{A}) = -\sum_{i=1}^{|\mathcal{A}|} \frac{|\mathcal{D}(\mathcal{A}_i)|}{|\mathcal{D}|} \times \log_2 \left(\frac{|\mathcal{D}(\mathcal{A}_i)|}{|\mathcal{D}|} \right) \tag{2.11}$$

donde $|\mathcal{A}|$ es el número de valores distintos del atributo $\mathcal{A}$; $\mathcal{D}(\mathcal{A}_i)$ es el subconjunto de $\mathcal{D}$ para el cual $\mathcal{A} = \mathcal{A}_i$, siendo $|\mathcal{D}(\mathcal{A}_i)|$ su cardinal; y $|\mathcal{D}|$ es el número total de ejemplos.

La información de separación simboliza la información potencial que representa dividir el conjunto de datos, y es usada para compensar la menor ganancia de aquellos test con pocas salidas. Con ello, tal y como muestra la ecuación 2.12, la razón de ganancia es calculada como el cociente entre la ganancia de información (ecuación 2.10) y la información de separación (ecuación 2.11). Tal cociente expresa la proporción de información útil generada por la división.

$$RazonDeGanancia(\mathcal{D}, \mathcal{A}) = \frac{Ganancia(\mathcal{D}, \mathcal{A})}{InformacionDeSeparacion(\mathcal{D}, \mathcal{A})} \tag{2.12}$$

C4.5 maximiza este criterio de separación, premiando así a aquellos atributos que, aun teniendo una ganancia de información menor, disponen también de menor número de valores para llevar a cabo la clasificación. Sin embargo, si el test incluye pocos valores, la información de separación puede ser cercana a cero, y por tanto el cociente sería inestable. Para evitar tal situación, el criterio selecciona un test que maximice la

razón de ganancia pero obligando a que la ganancia del mismo sea al menos igual a la ganancia media de todos los test examinados [127].

C4.5 ha resultado ser un sistema muy efectivo en la práctica, capaz de ofrecer una representación relativamente simple de los resultados con un bajo coste computacional. En concreto, para un conjunto de datos con N ejemplos y m atributos, el coste medio de construcción del árbol es de $\Theta(mNlog_2N)$, mientras que la complejidad del proceso de poda es de $\Theta(N(log_2N)^2)$. Por contra, el algoritmo presenta también dos inconvenientes importantes derivados de la representación del conocimiento que obtiene y la metodología seguida para ello:

1. La representación mediante árboles de decisión paralelos[3] puede provocar que zonas contiguas en el espacio no puedan ser unidas para simplificar la regla. Esto hace que el árbol tienda a crecer sustancialmente en aplicaciones reales, complicando la compresión del mismo.

2. La estrategia seguida establece en cada paso una única frontera de decisión para un solo atributo, sin posibilidad de reajustar el modelo en pasos posteriores. Es decir, C4.5 establece en un momento dado una condición sobre un atributo porque en ese instante entiende que es la mejor, sin tener en consideración que en el proceso posterior de establecer condiciones sobre los demás atributos, esa primera opción pudiera no ser la mejor.

C4.5Rules

Motivado principalmente por el primero de los inconvenientes citados para C4.5, Quinlan [145] propuso un método para transformar un árbol de decisión en un conjunto ordenado de reglas de decisión. La aplicación de C4.5 junto a éste método de traducción es conocido como C4.5Rules.

Los pasos para la generación de las reglas son:

1. *Creación del árbol*: induce un árbol de decisión con C4.5.

2. *Reglas Iniciales*: construye un conjunto de reglas de decisión convirtiendo cada

[3] Árboles univariable (véase la sección 2.3.2, página 19)

camino distinto de la raíz a una hoja en una regla. Así, son creadas tantas reglas como hojas tiene el árbol de decisión inicial.

3. *Generalización*: examina las reglas iniciales y elimina cualquier condición que no contribuya a la mejora de la precisión de las mismas.

4. *Agrupamiento y selección*: las reglas generalizadas son agrupadas en conjuntos de reglas por clase, cada uno de los cuales cubren a una clase particular. Posteriormente, extrae de cada conjunto de reglas un subconjunto que maximiza la precisión en la predicción para la clase asociada aplicando el principio de Longitud Descripción Mínima (*MDL, Minimum Description Length*) [146, 158].

5. *Ordenación*: las reglas son ordenadas de forma decreciente por el error cometido y se establece una *clase por defecto* para clasificar aquellos ejemplos de entrenamiento que no son cubiertos por ninguna regla actual. La clase más frecuente de esos casos es designada por defecto. Así, la clasificación de nuevos ejemplos se lleva a cabo siguiendo la secuencia de reglas.

6. *Evaluación y poda*: el conjunto de reglas es evaluado sobre el conjunto de entrenamiento para determinar si alguna regla afecta negativamente a la precisión, en cuyo caso es eliminada. Este proceso se repite hasta que el modelo no admita alguna mejora adicional.

Tras este proceso se obtiene un conjunto de reglas de decisión con una precisión aproximadamente igual a la de un árbol de decisión podado, aunque mucho más sencillo desde el punto de vista de la interpretación humana.

OC1

El sistema OC1 (*Oblique Classifier 1*), desarrollado por S. K. Murthy et al. [131], es una herramienta de inducción de árboles de decisión oblicuos basada en una propuesta anterior de Breiman et al. denominada CART (*Classification And Regression Trees*) [26]. Un árbol de decisión oblicuo contiene en cada nodo interno una combinación lineal de los atributos, en lugar de la condición sobre un único atributo que contenían los árboles paralelos a los ejes (véase la sección 2.3.2, página 19).

OC1 combina una estrategia de escalada (*hill-climbing*) con un proceso aleatorio para generar las divisiones oblicuas en forma de hiperplanos. Para cada nodo, usa una técnica similar a la de C4.5 para obtener el mejor hiperplano paralelo según un determinado criterio que maximiza la bondad de la división. Posteriormente explora diferentes rotaciones de dicho hiperplano realizando una búsqueda local probabilística mediante una técnica basada en *simulated annealing* [103, 120]. Cuando se cumple el criterio de parada, toma el hiperplano con máxima bondad encontrado y pasa a generar el siguiente nodo.

Entre las diferentes medidas de bondad que se pueden aplicar en OC1, las que mejores resultados en promedio ofrecen son la ganancia de información [141], el criterio de Gini y la regla de Twoing [26].

Aunque en ciertos dominios, los árboles de decisión oblicuos son más precisos que los paralelos, su inducción también es más costosa computacionalmente, debido principalmente a la búsqueda estocástica de las rotaciones. Otro inconveniente notable es su difícil interpretación.

2.5.4. Inducción de Reglas de Decisión

A diferencia de los sistemas de inducción árboles de decisión, que utilizan una estrategia de divide y vencerás, las técnicas de inducción de reglas siguen una estrategia generalización y especialización ascendente. Al igual que ID3 se considera el punto de partida para el desarrollo de sistemas de inducción de árboles de decisión, la familia de algoritmos AQ[123] es considerado en el mismo sentido para estrategias de aprendizaje mediante inducción de reglas.

AQ

Desarrollado originalmente por R. S. Michalski en 1973 [123], el algoritmo AQ (*Algorithm for Quasi-optimal solutions*), ha sido objeto de numerosas reimplementaciones y mejoras en los últimos 30 años [32, 99, 124, 125]. Se trata un algoritmo de cubrimiento progresivo, el cual induce conjuntos de reglas del tipo "*Si* $\mathcal{P}$ *Entonces* $clase = \mathcal{C}$", donde $\mathcal{P}$ es un predicado booleano sobre los valores de los atributos en FND.

AQ obtiene n conjuntos de reglas, uno por cada clase $\mathcal{C}_i \in \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$, de forma que

cada conjunto $\mathcal{S}_i$ cubre todos ejemplos de una única clase $\mathcal{C}_i$ (ejemplos positivos) sin cubrir ninguno de clase distinta (ejemplos negativos). La generación de cada conjunto $\mathcal{S}_i$ es independiente. Así, el algoritmo toma una clase $\mathcal{C}_i$ y divide el conjunto de datos en dos subconjuntos $\mathcal{D}^+$ y $\mathcal{D}^-$ de ejemplos positivos ($clase = \mathcal{C}_i$) y negativos ($clase \neq \mathcal{C}_i$), respectivamente. Posteriormente, va calculando las reglas mediante la generalización gradual de las descripciones de ejemplos positivos seleccionados (*semillas*). Cuando el conjunto de reglas $\mathcal{S}_i$ cubre los ejemplos de $\mathcal{D}^+$, se pasa a generar el siguiente conjunto $\mathcal{S}_{i+1}$ para la clase $\mathcal{C}_{i+1}$.

Para calcular las reglas de un determinado conjunto $\mathcal{S}_i$, AQ selecciona al azar una semilla $e \in \mathcal{D}^+$ y genera un conjunto de reglas maximales que caracterizan dicha semilla sin contener ejemplos negativos. Posteriormente, selecciona la mejor regla de este conjunto de acuerdo con un criterio de preferencia y la añade al conjunto $\mathcal{S}_i$, pasando a seleccionar otra semilla para continuar la generación de reglas.

CN2

El algoritmo CN2 fue desarrollado por P. Clark y T. Niblett [36] para atacar los problemas de ruido y sobreajuste que se encuentran en sistemas como AQ [123] o ID3 [140], combinando eficiencia para de tratar con ruido de ID3 con la forma de las reglas y flexibilidad de búsqueda de AQ.

A diferencia de algunas implementaciones como AQ11 [124] o AQ15 [125], CN2 no trata el ruido imponiendo pre y post-condiciones sobre el algoritmo básico de AQ, sino que es en sí una generalización de éste. Por otro lado, respecto al problema del sobreajuste, CN2 elimina la dependencia de un ejemplo específico durante su búsqueda para ampliar así el espacio de reglas, siendo ésta una de sus principales contribuciones.

Otro aspecto interesantes introducido por este método es la incorporación de un test de significatividad que asegura que la distribución de los ejemplos en las clases es significativamente distinta de la que podría ocurrir por casualidad. El usuario proporciona un umbral de significatividad por debajo del cual las reglas son rechazadas. La medida significatividad se basa en la razón de verosimilitud, que mide la distancia entre dos distribuciones.

La versión original de CN2 genera una lista ordena de reglas aplicando un criterio

de entropía mínima parecido al usado en ID3 como medida heurística. Este criterio fue sustituido por la estimación del error Laplaciano en una versión mejorada del algoritmo [35]. Dicha versión también contemplaba la posibilidad de generar reglas no ordenadas.

2.5.5. Aprendizaje de Reglas Mediante Algoritmos Genéticos

Aunque el aprendizaje de reglas mediante Algoritmos Genéticos es descrito detalladamente en el Capítulo 3, en esta sección haremos un bosquejo de las técnicas más importantes.

Dentro del campo del aprendizaje supervisado, los Algoritmos Genéticos y Evolutivos han sido ampliamente utilizados para la inducción de reglas, obteniendo excelentes resultados. Existen dos puntos de vista fundamentales en la aplicación de Algoritmos Genéticos a problemas de clasificación: Michigan y Pittsburgh.

En el enfoque de Michigan, cada individuo, de longitud fija, codifica a una única regla, siendo toda la población la solución al problema. Los algoritmos que siguen la esta filosofía suelen denominarse Sistemas Clasificadores (*Classifier Systems*), los cuales permiten la competitividad entre soluciones apoyándose en técnicas de premio y castigo o asignación de créditos. Entre este tipo de clasificadores destacan los algoritmos CS-1 de Holland [93], XCS de Wilson [180] y la propuesta de Riolo [151].

Los algoritmos que aplican el enfoque de Pittsburgh siguen los criterios establecidos por S. F. Smith [168] y son denominados comúnmente Sistemas de Aprendizaje (*Learning Systems*). Al contrario de la estrategia de Michigan, los sistemas de aprendizaje definen individuos de longitud variable que representan conjuntos reglas, pudiendo ser cada uno de los individuos de la población una solución completa al problema. Así, cada individuo compite con el resto para cubrir todos los casos. Destacan en este apartado las herramientas SAMUEL [83], GABIL [45], GIL [98] y GASSIST [17].

Existen en la literatura otras propuestas que no se ajustan totalmente a ninguna de las dos filosofías anteriores. Éste es el caso de la herramienta COGITO, cuya metodología es más cercana a la escuela de Pittsburgh, pero usa una población de los individuos de longitud fija.

2.6. Medidas de Rendimiento

En general, el rendimiento de un sistema puede ser medido según dos criterios [94]: su precisión en la clasificación y su complejidad. Estos dos aspectos caracterizan la calidad de los modelos de conocimiento generados por un sistema de aprendizaje, siendo ambas de igual importancia.

2.6.1. Precisión

La medida más efectiva de precisión de un sistema clasificador es tasa de error[4] (ER, *Error Rate*), expresada en términos de probabilidad o porcentaje. Normalmente no se dispone de una expresión analítica de la tasa de error real de un problema particular, por lo que ésta debe ser estimada a partir de los propios datos mediante la ecuación 2.13.

$$ER = \frac{num.\ errores}{num.\ de\ ejemplos} \tag{2.13}$$

Para calcular la tasa de error de un clasificador, se divide el conjunto de datos en dos subconjuntos: entrenamiento, usado para el aprendizaje, y test, sobre el cual se mide la tasa de error empírica del modelo. Existen diversos métodos para realizar esta división, los cuales son descritos en la sección 2.6.3.

2.6.2. Complejidad

La complejidad de un modelo de conocimiento comprende diversas características como el tamaño de la estructura y su legibilidad, así como el esfuerzo computacional necesario tanto para generarlo como para aplicarlo. Una estimación de la complejidad aceptada en la comunidad científica y extensamente usada en la bibliografía es el número de reglas (NR).

El cálculo del número de reglas depende de la representación del conocimiento utilizada por el sistema de aprendizaje. En la representación mediante reglas de decisión, el cálculo es trivial, resumiéndose en un simple conteo del las reglas. Sin embargo, el

[4] Algunos autores prefieren usar la medida complementaria, la tasa de aciertos.

numero de reglas incluidas en un árbol de decisión es el número de caminos diferentes desde la raíz hasta las hojas, es decir, el número de hojas diferentes. En el caso de árboles binarios completos[5], se puede aplicar la expresión $\frac{n+1}{2}$, donde n es el número de nodos.

Aunque el número de reglas resulte una medida efectiva del tamaño del modelo y, en cierto modo, de su legibilidad, no proporciona información sobre la comprensibilidad del mismo. Sin embargo, sería interesante poder medir cómo de interpretable es un modelo desde el punto de vista humano, ya que, al fin y al cabo, será un experto quien aplique el modelo. Quizá por ser ésta una característica más subjetiva, no ha sido tratada formalmente.

2.6.3. Métodos de Validación

Como se ha mencionado anteriormente, es necesario dividir la muestra de aprendizaje en los conjuntos de entrenamiento y test para aproximar el error estimado a la tasa real. La estrategia para validar un sistema de aprendizaje depende esencialmente de la manera en que dicha división es realizada. Algunos autores utilizan el mismo conjunto para el entrenamiento y el test, lo que produce una tasa de error casi siempre menor a la real y a menudo una estimación demasiado optimista[64]. Para que la estimación sea válida, los conjuntos de entrenamiento y test deben ser independientes, o al menos diferentes [47]. Los métodos de validación más destacados son:

- **Validación cruzada**. La validación cruzada con k conjuntos (*k-fold cross validation*) es atribuida a M. Stone y aparece descrita en [169]. Consiste en dividir los datos en k subconjuntos de ejemplos con aproximadamente igual tamaño y evaluar el sistema k veces. En cada evaluación, se deja uno de los subconjuntos para el test y se entrena el sistema con los $k-1$ restantes. Así, el error estimado es la media de las k tasas obtenidas. A partir de este método de validación cruzada básico, se han desarrollado variantes para aproximar mejor la tasa de error.

- **Validación cruzada completa**. El método de validación cruzada completa (*complete k-fold cross validation*) realiza una exploración completa de todas las posibles

[5] Árboles en los que todos los nodos internos tienen dos hijos.

combinaciones de $\frac{N}{k}$ ejemplos en el conjunto de test, donde N es el número total de ejemplos, dejando el resto para el entrenamiento. En concreto se llevan a cabo $\binom{N}{N/k}$ evaluaciones, lo cual resulta extremadamente costoso [71].

- **Validación cruzada estratificada**. Una variante del método básico, denominada validación cruzada estratificada (*stratified k-fold cross validation*), distribuye los ejemplos intentando mantener la misma proporción de instancias de cada clase en el conjunto de entrenamiento y en el de test. Éste es quizá el método de validación a más recomendable [26], ya que mantiene para las evaluaciones la misma proporción de clases del conjunto total de datos, además de no aumentar significativamente el coste computacional respecto al método original.

- **Validación dejando uno fuera**. Este método es la validación cruzada llevada al extremo, es decir, tomando k igual al número de ejemplos (N) del conjunto de datos. Así, el clasificador entrena con $N - 1$ ejemplos, dejando uno fuera para realizar el test, de ahí su nombre (*leave–one–out*) [110]. Además de la elevada varianza de la tasa de error obtenida, el mayor inconveniente de este método es su alto coste computacional, por lo que no es recomendable su uso con más de 100 ejemplos [176].

- **Split Sample**. *Split sample* es un tipo de validación no cruzada donde sólo existe un conjunto de test sobre el cual se estima el error. El tamaño dicho conjunto suele rondar el 20–30 % del conjunto original, empleándose el resto para entrenamiento. La selección de los ejemplos para cada conjunto es aleatoria, aunque es aconsejable asegurar la misma proporción de ejemplos de distintas clases en ambos.

- **Bootstrapping**. Existen varias formas de aplicar la validación *bootstrapping* o validación por secuencia [55]. La más simple consiste en realizar un muestreo aleatorio con reemplazo en el conjunto de datos, copiando los ejemplos seleccionados en el conjunto de entrenamiento hasta que éste alcance el tamaño del original. El conjunto de test lo conforman aquellos ejemplos no incluidos en el entrenamiento. Aunque la tasa de error de este conjunto es un estimador del error real, lo habitual es repetir el proceso varias veces y calcular la media.

De los diversos estudios presentes en la literatura [55, 82, 111, 169], podemos colegir que no existe un método mejor que el resto para todas las situaciones. Weiss y Kulikowski realizaron un interesante análisis comparativo [176], tras el cual concluyeron que, para muestras mayores a 50 ejemplos la validación cruzada era el método más adecuado, mientras que para menor número de casos se recomendaba *bootstrapping*.

Capítulo 3

Aprendizaje Evolutivo

Una diferencia inferior a un grano en una balanza
puede determinar qué individuos han de vivir y cuáles perecerán.
CHARLES DARWIN.

En el área del Aprendizaje Automático, cuando el usuario conoce la clase de los ejemplos del conjunto de datos y desea obtener un modelo de conocimiento capaz de predecir la clase de nuevos ejemplos a partir de los valores de los atributos de éstos, nos encontramos en el campo del Aprendizaje Supervisado.

Las técnicas de aprendizaje supervisado pueden ser clasificadas siguiendo diferentes criterios. Desde la perspectiva de este trabajo, los dos criterios más destacados para discernir los distintos tipos de técnicas son la representación del conocimiento que obtienen y la metodología seguida para ello. Según este último, las técnicas que pueden aplicarse como método de búsqueda de patrones son muy variadas: vecinos más cercanos, redes neuronales, inducción, etc. Cuando el método de aprendizaje aplica Técnicas de Computación Evolutiva para la obtención de patrones que modelen conocimiento inherente a un conjunto de datos, nos encontramos en el campo del Aprendizaje Evolutivo. De igual modo y atendiendo al otro criterio de clasificación de las técnicas de aprendizaje, el modelo de conocimiento puede ser representado por diversas estructuras como la representación proposicional, árboles de decisión, listas de decisión o reglas de decisión. Son estas últimas, las técnicas que generan reglas de decisión, las que mayor interés adquieren en esta investigación.

Este capítulo resume las características y factores influyentes en el aprendizaje evolutivo de reglas de decisión, así como las propuestas más destacadas en este campo.

3.1. Conceptos de Computación Evolutiva

La Computación Evolutiva engloba un conjunto de técnicas que aplican el concepto Darwiniano de Evolución Natural a la búsqueda de soluciones a un problema. En general, estas técnicas parten de una serie de soluciones iniciales y simulan el proceso evolutivo de manera que tales soluciones se van trasformando y adaptando cada vez mejor al entorno. Para poder simular el proceso evolutivo en un ordenador es necesario tener en cuenta cuatro factores fundamentales:

- **Codificación**: Representación interna de las soluciones, trasformando éstas en "individuos" que puedan evolucionar.
- **Operadores genéticos**: Funciones de reproducción de los individuos (cruce y mutación).
- **Evaluación**: Definir una función de aptitud que cuantifique la adaptación de un individuo al entorno, es decir, la bondad de una solución frente al problema.
- **Selección**: Mecanismo que decide qué individuos deben reproducirse dependiendo de su aptitud.

Desde el punto de vista de la evolución, un ser vivo puede identificarse con un par genotipo-fenotipo. El fenotipo es el conjunto de rasgos físicos y psíquicos de un individuo, mientras que el genotipo es el código genético que contiene la información para que se desarrolle el fenotipo. Así, en la evolución natural, las características físicas y psíquicas de cualquier ser vivo vienen determinadas por su código genético. Es este código genético el que posibilita la herencia de tales características entre padres e hijos. Desde este punto de vista, la codificación es el código genético que representa las características de las soluciones al problema, mientras que los operadores genéticos son las funciones que establecen cómo se transmiten las características de los padres a los hijos. Se distinguen dos tipos de operadores, *cruce* y *mutación*, los cuales pueden actuar a nivel de genotipo y/o fenotipo. El cruce es la obtención de nuevos individuos a partir de dos o más individuos denominados padres. En cambio, la mutación es una alteración en el genotipo o fenotipo que hace que el individuo cambie su comportamiento.

Podemos establecer un paralelismo entre el problema que se pretende solucionar y el entorno donde los individuos (soluciones) han de evolucionar. Por ello, de los cuatro aspectos mencionados anteriormente, la evaluación de los individuos es el más dependiente del problema, ya que modela las condiciones que deciden qué individuos son los más fuertes y por tanto más probabilidades tienen de sobrevivir. Por último, el mecanismo de selección simula los aspectos aleatorios de la evolución natural que hacen que no siempre sobrevivan los más fuertes y mueran los más débiles, sino que la aptitud de un individuo establezca su probabilidad de supervivencia y reproducción.

Históricamente, podemos hablar de tres paradigmas de la computación evolutiva:

- Programación Evolutiva
- Estrategias Evolutivas
- Algoritmos Genéticos y Evolutivos

Aunque el origen y el desarrollo de estas técnicas fueron independientes, la evidente relación entre ellas ha hecho que desarrollen aspectos comunes. Esto hace que cada vez sea más difícil establecer las fronteras que separan los distintos paradigmas de la computación evolutiva. No obstante, aunque nuestro interés se centra en los algoritmos genéticos y evolutivos, haremos una breve reseña de las características de cada uno.

Programación Evolutiva

Se centra en el comportamiento de los individuos (fenotipo) sin tener en cuenta los aspectos genéticos específicos (genotipo) y, por lo tanto, sin necesidad de codificación. Hace una abstracción a nivel de especie, considerando a cada individuo como una especie distinta e independiente del resto. Por ello, sólo aplica mutación, ya que el cruce entre especies carece de sentido. Asimismo, aplica un mecanismo de selección estocástico. Las principales áreas de aplicación de la programación evolutiva son [63]: predicción, generalización, control automático, planificación de rutas, diseño de redes neuronales y reconocimiento de patrones.

Estrategias Evolutivas

Al igual que la programación evolutiva, las estrategias evolutivas operan con el fenotipo. Sin embargo, en este caso la abstracción se hace a nivel de individuo en vez de especie. Esto posibilita que se produzca recombinación, es decir, que varios individuos interactúen entre sí para producir nuevos individuos. Este cruce se produce a nivel de fenotipo y es una operación secundaria frente a la mutación. Otro aspecto destacable de las estrategias evolutivas es que aplican una selección determinista, frente a la probabilística usada por la programación evolutiva y los algoritmos genéticos. Las estrategias evolutivas han sido aplicadas tradicionalmente a resolver problemas de bioquímica, óptica, magnetismo y diseño en ingeniería.

Algoritmos Genéticos y Evolutivos

El primer aspecto a destacar de los algoritmos genéticos es que su desarrollo fue motivado por la resolución de problemas de aprendizaje automático [90, 91], aunque en realidad son técnicas de optimización y búsqueda con un amplio espectro de aplicaciones.

Los individuos evolucionan aplicando cruces y mutaciones a nivel genotípico, lo cual implica la necesidad de una función de codificación que no existía en los otros dos paradigmas de la computación evolutiva. La codificación es la representación interna que el algoritmo utiliza como genotipo de los individuos. Cada característica de un individuo es codificada por un elemento denominado *gen*. En la representación tradicional, llamada *codificación binaria*, cada gen es uno o varios bits. La figura 3.1 ilustra la relación entre un genotipo binario y su fenotipo correspondiente.

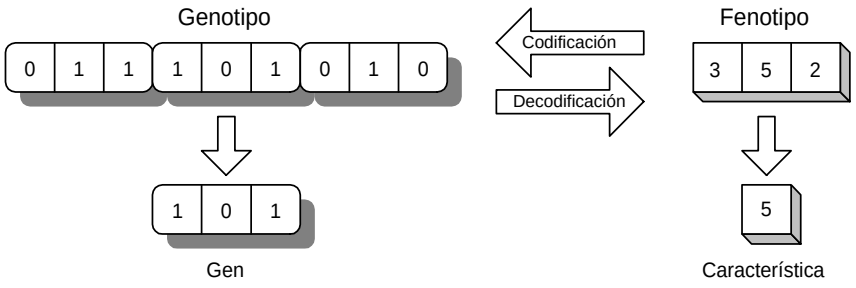


Figura 3.1: Genotipo vs. Fenotipo.

Los algoritmos genéticos dan mayor importancia al operador de cruce frente al de mutación. Mientras que los operadores genéticos son aplicados a nivel genotípico, la evaluación de los individuos se suele realizar a nivel fenotípico, decodificando el genotipo y cuantificando su aptitud. Asimismo, al igual que la programación evolutiva, aplican selección probabilística.

Aunque inicialmente los algoritmos genéticos sólo usaban la codificación binaria para representar a los individuos, la aplicación de este paradigma a nuevos problemas ha fomentado el desarrollo de nuevos tipos de codificación más cercanas al problema a resolver. Así, representaciones como la *codificación real* o la *codificación híbrida* pueden simplificar notablemente el proceso de búsqueda de soluciones. Los algoritmos genéticos que usan estas codificaciones diferentes a la binaria se denominan *Algoritmos Evolutivos*. Estos algoritmos utilizan una representación más cercana al fenotipo de los individuos, pero siguen el mismo proceso de evolución que los algoritmos genéticos, es decir, los cruces y mutaciones se realizan a nivel genotípico. Por ello, ambos tipos de algoritmos se suelen incluir dentro del mismo paradigma de la computación evolutiva.

3.2. Reglas mediante Algoritmos Genéticos

Básicamente, los algoritmos genéticos usados en la obtención de reglas son clasificados según dos grandes vertientes: Michigan y Pittsburgh. La principal diferencia entre estas dos estrategias radica en la representación de los individuos de la población, es decir, la codificación de las reglas. Esta diferencia de codificación e interpretación de las reglas afecta directamente al método de generación de las mismas. Así, la elección de una u otra estrategia depende fundamentalmente del tipo de reglas que se quieran generar, lo cual está claramente relacionado con el tipo de tarea de minería de datos que se lleve a cabo.

Michigan

Cada individuo de la población codifica una única regla y mantiene una longitud constante. Cada regla es evaluada sin tener en cuenta al resto de la población, lo cual hace que no exista un nexo de unión entre las reglas y que éstas evolucionen con independencia. La corta longitud de los individuos tiende a reducir el tiempo empleado para

calcular la función de evaluación y a simplificar el diseño de los operadores genéticos. Sin embargo, la evaluación separada de reglas individuales dificulta el análisis y evaluación de las interacciones entre las reglas. Por ello, cuando se pretende obtener un conjunto de reglas relacionadas y no reglas simples e independientes es necesario la aplicación de los denominados nichos [118], lo cual tiende a incrementar el coste computacional. Este tipo de algoritmos genéticos es aplicado cuando se busca un conjunto reducido de reglas independientes con una alta precisión. En general, a los sistemas que permiten la competitividad entre reglas ayudada por técnicas de premio y castigo se les denomina "sistemas clasificadores", los cuales siguen los criterios establecidos por Holland (Universidad de Michigan)[92]. Dado que este tipo de sistemas no se ajustan al propósito de este estudio, su descripción pormenorizada no es incluida en este documento.

Pittsburgh

Cada individuo de la población codifica un conjunto de reglas. Aunque la longitud de las reglas es constante, la longitud de dichos individuos puede ser variable, ya que cada individuo puede codificar un número distinto de reglas. Las reglas no son evaluadas por separado sino en conjunto, siendo la calidad de una regla dependiente del resto. En otras palabras, la interacción entre las reglas es importante. La mayor longitud de los individuos respecto a los métodos que siguen la filosofía de Michigan hace que el cálculo de la función de evaluación sea más costoso computacionalmente. Por otro lado, la variabilidad de la longitud origina modificaciones en los operadores genéticos para poder ser aplicados a individuos relativamente complejos. Es común en la literatura denominar "sistemas de aprendizaje" a los sistemas regidos por las propuestas de Smith (Universidad de Pittsburgh)[168]. La filosofía de Pittsburgh ha sido habitualmente aplicada para resolver, entre otras tareas de minería de datos, problemas de clasificación. Entre los sistemas de aprendizaje diseñados para tareas de clasificación destacan GABIL [45] y GIL [98], los cuales son ampliamente referenciados en la bibliografía.

3.2.1. GABIL

Siguiendo de una forma minimalista la filosofía de la escuela de Pittsburgh, K. A. De Jong et al. [45] desarrollaron un sistema de aprendizaje basado en un algoritmo genético, el cual denominaron GABIL (*GA-based batch-incremental concept learner*). Este sistema aprende y refina reglas de clasificación de conceptos, y por lo tanto opera exclusivamente con atributos discretos. Tales reglas presentan en su antecedente una conjunción de condiciones, cada una de las cuales representa el conjunto de valores que un determinado atributo puede tomar en FND modificada[1] [122]. El consecuente de la regla indica la clase (concepto) con que ha de clasificarse un ejemplo que satisfaga las condiciones impuestas en el antecedente.

GABIL utiliza codificación binaria como representación interna de las reglas. Esta representación tiene un tamaño fijo para todas las reglas, donde cada atributo viene representado por una cadena de bits (*test*) de longitud igual al número de valores diferentes que el atributo puede tomar. Cada uno de estos bits simboliza la presencia o ausencia de uno de los valores en la condición correspondiente, de forma que el i-ésimo bit se pondrá a 1 si el valor i-ésimo del dominio del atributo aparece en la condición y a 0 en caso contrario. Con esta semántica, un test donde todos los bits tomen valor 1 significa que el atributo puede tomar cualquier valor de su dominio y, por tanto, la condición podría omitirse en la regla. Nótese que ésta codificación no produce pérdida de generalidad al exigir la existencia de un test para cada uno de los atributos. Por último, la clase es representada mediante un bit que indica si la hipótesis sobre el concepto a aprender es positiva o negativa.

Cada individuo de la población genética está formado por la concatenación de diferentes reglas, presentando por tanto una longitud variable, aunque acotada por un máximo preestablecido por el usuario.

Ejemplo 3.1. *Si el conjunto de valores permitidos para el atributo a_1 es {pequeño, mediano, grande}, y el de a_2 es {blanco, rojo, verde, azul, negro}, entonces las reglas*

R_1: Si (a_1 = pequeño) Entonces es C
R_2: Si (a_1 = mediano o grande) y (a_2 = blanco) Entonces es C

[1] GABIL utiliza una modificación de la Forma Normal Disyuntiva que permite disyunciones internas.

son representadas por el individuo mostrado en la figura 3.2.

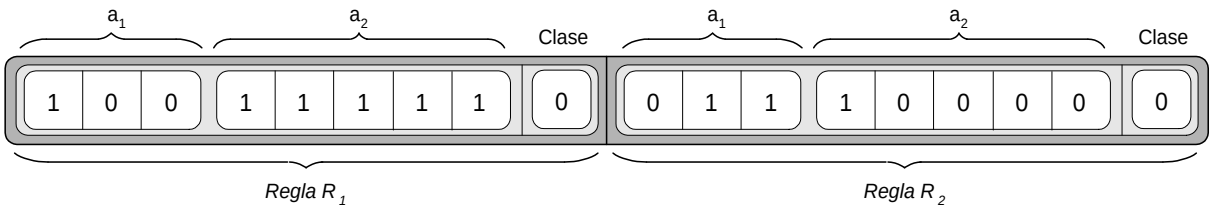


Figura 3.2: Codificación en GABIL.

Siguiendo el enfoque minimalista, se aplica una función de evaluación muy simple, la cual sólo tiene en cuenta los resultados en la clasificación, ignorando aspectos como la complejidad de la reglas. Así, la bondad de un individuo es medida en función del porcentaje de aciertos de dicho individuo sobre el conjunto de entrenamiento, en concreto

$$\varphi(i) = (porcentaje\ de\ aciertos)^2 \tag{3.1}$$

Esta función de evaluación dirige la búsqueda hacia soluciones que contemplen sólo aciertos, es decir, impulsa la evaluación hacia conjuntos de reglas *completas* y *consistentes* (véase página 18, definiciones 2.13 y 2.15).

Respecto a los operadores genéticos, GABIL utiliza un cruce bipuntual con probabilidad 0.6 adaptado a la longitud variable de los individuos, ya que exige que los dos cortes se encuentren en posiciones semánticamente idénticas. La mutación es la usual de las cadenas binarias, aplicándose con una probabilidad muy baja (0.001). Además de los operadores genéticos usuales, los autores introdujeron dos operadores de sesgo para dirigir la búsqueda hacia la generalización y especificación de las soluciones: operador de inserción alternativa (AA: *Adding Alternative*) y operador de eliminación de condición (DC: *Dropping Condition*). El operador AA es un operador de mutación que aumenta la generalidad de una regla añadiendo una nueva alternativa a ésta mediante la incorporación de una disyunción interna a una de las condiciones [122]. Por otra parte, el operador DC elimina condiciones de las reglas poniendo a 1 todos los bits de un test si el número inicial de bits a 1 supera la mitad [81].

El modo de funcionamiento del algoritmo es definido por los autores como *batch-incremental*. En cada paso toma un ejemplo del conjunto de entrenamiento y comprueba

si existe un conjunto de reglas en la población que lo clasifican correctamente, en cuyo caso pasa al siguiente ejemplo. Si por el contrario, la clasificación resulta incorrecta, el algoritmo genético es invocado para generar un nuevo conjunto de reglas que clasifiquen correctamente a todos los ejemplos tratados hasta el momento, incluido el actual.

Aunque el nombre de GABIL es utilizado normalmente para referirse a la propuesta general, los autores utilizan diferente nomenclatura para denominar a las distintas alternativas de ejecución del algoritmo. Así, GABIL es la propuesta original sin la utilización de los operadores de sesgo. GABIL+A incorpora el operador AA con probabilidades de 0.25 y 0.75 de mutar a 0 y 1, respectivamente. Cuando el algoritmo integra el operador DC es denominado GABIL+D, aplicando éste con probabilidad 0.6. Por último, GABIL+AD es la combinación de los dos anteriores, incorporando ambos operadores específicos.

Pese a que los resultados obtenidos por los tres sistemas anteriores mejoraron a la propuesta original, también mostraron que no existe un criterio global para la aplicación de los operadores de sesgo adecuado para todos los problemas. Por ello, los autores dotaron de adaptabilidad a la aplicación de los operadores AA y DC, incorporando a los individuos dos bits de control que indicaban cuándo debían ser aplicados ambos operadores. Esta versión del algoritmo, denominado GABIL-$adaptativo$, produjo resultados sensiblemente mejores a los de las propuestas anteriores.

3.2.2. GIL

GIL (*Genetic-based Inductive Learning*) es un sistema de aprendizaje desarrollado por C. Z. Janikow [98], el cual tiene la capacidad de aprender múltiples conceptos, permitiendo más de dos etiquetas para la clase. Aunque comparte muchos aspectos con GABIL, no sigue la filosofía minimalista de éste, siendo un sistema mucho más complejo en todos los aspectos.

Respecto a la representación de las reglas, el autor opta por una versión simplificada del lenguaje VL_1 [126] que restringe el uso de los operadores relacionales, permitiendo únicamente el "$=$" para facilitar el diseño de los operadores genéticos. La representación interna de reglas es una codificación binaria similar a la descrita anteriormente para el sistema GABIL.

La bondad de los individuos se determina en función de su complejidad, completitud y consistencia. GIL extiende estas dos últimas propiedades, ya que, no sólo pueden ser medidas para una regla independiente, sino también para conjuntos de reglas. Así, se calcula la completitud y consistencia para una regla respecto al conjunto al que pertenece e igualmente para un conjunto de reglas respecto al total de ejemplos, como muestra la tabla 3.1,

Tipo	Completitud	Consistencia
Conjunto de reglas	$\frac{\epsilon^+}{E^+}$	$1 - \frac{\epsilon^-}{E^-}$
Regla	$\frac{e^+}{\epsilon^+}$	$1 - \frac{e^-}{\epsilon^-}$

Tabla 3.1: Medidas de completitud y consistencia [98].

donde e^+/e^- es el número de ejemplos positivos/negativos cubiertos por la regla, ϵ^+/ϵ^- es el número de ejemplos cubiertos por el conjunto de reglas, y E^+/E^- es total de ejemplos. La completitud y la consistencia son combinadas mediante la ecuación 3.2, asignándole a cada una de ellas un grado de influencia o peso (w_1 y w_2 respectivamente), dando como resultado lo que los autores denominan *corrección*. Los pesos w_1 y w_2 pueden ser configurados para dirigir la búsqueda hacia soluciones más completas o consistentes.

$$correccion = \frac{w_1 \times completitud + w_2 \times consistencia}{w_1 + w_2} \tag{3.2}$$

La tercera propiedad que influye en la evaluación es la *complejidad* de las reglas, la cual es calculada a partir del número de reglas y condiciones de éstas, como muestra la ecuación 3.3. Esta medida de la complejidad se emplea para calcular posteriormente el coste de descripción [181].

$$complejidad = 2 \times num\_reglas + num\_condiciones \tag{3.3}$$

Finalmente, la función de evaluación combina la *corrección* de una regla con su *coste* de descripción para medir tanto el rendimiento como la complejidad de los individuos. Concretamente, GIL aplica la función de evaluación

$$\varphi(i) = correccion \times (1 + w_3 \times (1 - coste))^f \tag{3.4}$$

donde el w_3 es un valor normalizado en $[0, 1]$ que establece la influencia del coste; y f es una función que modela la edad de la población, creciendo muy lentamente en el intervalo $[0, 1]$ a medida que el proceso evolutivo avanza.

El aspecto más complejo del sistema son los operadores genéticos, tanto por su definición como por el modo de aplicación de los mismos. GIL define operadores de tres tipos: generalización, especialización e independencia. Los dos primeros tipos tienen el mismo propósito que los definidos por GABIL, mientras que los operadores de independencia persiguen el aumento de la diversidad de la población. Independientemente del tipo de los operadores, éstos tienen tres niveles distintos de actuación:

1. *Nivel de conjunto de reglas*, donde los operadores actúan sobre varios individuos a la vez. Tales operadores son: intercambio, copia, adición, generalización, eliminación y especialización.

2. *Nivel de regla*, donde se actúa sobre segmentos de individuos que correspondan a una única regla, definiéndose los operadores: división, eliminación de condición, cambio de conjunción, inserción de condición y división completa.

3. *Nivel de condición*, las operaciones se realizan sobre un segmento de los individuos correspondiente al antecedente de una regla. Los operadores de condición son: cambio, extensión y restricción de referencia.

La probabilidad de aplicación de cada tipo de operador es ajustada durante el proceso evolutivo, siendo calculada a partir de la completitud y la consistencia.

Usando los aspectos mencionados anteriormente, GIL aplica un algoritmo genético típico, aunque la evolución se produce en los tres niveles descritos para los operadores. Así, en cada iteración, los individuos son evaluados y seleccionados según su bondad para formar una nueva población. Posteriormente, se aplican los operadores genéticos para hacer que el sistema evolucione, de modo que cada estructura (conjuntos, reglas y condiciones) invoca probabilísticamente a los operadores definidos para su nivel. Este ciclo es repetido hasta alcanzar las soluciones deseadas o agotar los recursos computacionales.

En general, el análisis de los resultados de GIL muestra el buen comportamiento del sistema, obteniendo mejoras respecto a sistemas como AQ15 [125], C4.5 [145] y GABIL.

Según el propio Janikow, el mayor inconveniente del sistema GIL es la cantidad de parámetros de entrada (aproximadamente 40) que necesitan ser especificados.

3.2.3. SIA

La propuesta de G. Venturini [172], denominada SIA (*Supervised Inductive Algorithm*), es un sistema capaz de aprender reglas a partir de un conjunto de datos multietiquetados[2] y que, a diferencia de los sistemas mostrados anteriormente, operara tanto con dominios discretos como continuos. SIA genera un conjunto independiente de reglas por cada clase distinta, siendo las reglas de la forma:

$$R: \ \textit{Si}\ cond_1 \ \wedge \ cond_2 \ \wedge \ \ldots \ \wedge \ cond_m\ \textit{Entonces Clase=}\mathcal{C}_i\textit{, Fortaleza}$$

donde cada $cond_i$ es la condición sobre el i-ésimo atributo. Si a_i es discreto, la condición es de la forma $a_i = valor$. Si por el contrario es continuo, la condición es $B \leq a_i \leq B'$, con B y B' como límites del intervalo definido para a_i en la regla. También ofrece la posibilidad de no tener en cuenta un atributo, en cuyo caso la condición correspondiente se sustituye por "$\star$". Cada regla tiene un coeficiente de calidad asignado denominado *fortaleza*, la cual es calculada según el criterio $C_q(R)$

$$\begin{cases} C_q(R) = \frac{c - \alpha nc + \beta g}{csize} \\ C_q(R) \geq 0 \end{cases} \tag{3.5}$$

donde c es el número total de ejemplos que R clasifica correctamente; α es un coeficiente mayor o igual a 0; nc es el número de ejemplos clasificados incorrectamente por R; β es un coeficiente que vale 0, -0.001 ó $+0.001$; g es una medida de la generalidad de R normalizada en $[0, 1]$, donde 0 es muy específica y 1 muy general; por último, $csize$ es el número total de ejemplos con clase $\mathcal{C}_i$.

Para poder aplicar el algoritmo, el conjunto de datos es preprocesado para transformar cada ejemplo en una terna: $(e, \mathcal{C}_L, w)$; donde $e = \{e_1, \ldots, e_m\}$ es el vector de atributos; $\mathcal{C}_L$ es la lista de clases a las que pertenece el ejemplo; y w es el peso del ejemplo en el conjunto de datos. Este peso es añadido para guardar la distribución original de los datos.

[2]Conjunto de datos donde los ejemplos pueden pertenecer a más de una clase.

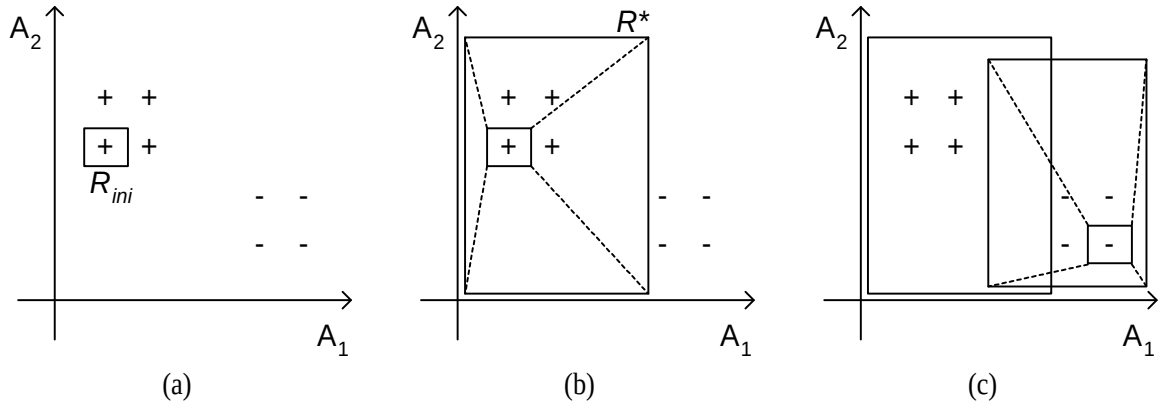


Figura 3.3: Aprendizaje de reglas por SIA [172].

SIA aplica un método de cubrimiento que toma un ejemplo $(e, \mathcal{C}_L, w)$ del conjunto de datos, donde exista una clase $\mathcal{C}_i \in \mathcal{C}_L$ que no haya sido cubierta por una regla anterior y genera la regla R_{ini} más específica que clasifica correctamente dicho ejemplo con clase $\mathcal{C}_i$ (figura 3.3a). Posteriormente invoca a un algoritmo genético que generalice R_{ini} para obtener la regla óptima R^* que maximiza el criterio de evaluación $C_q(R)$ (figura 3.3b). Luego, añade R^* al conjunto de reglas y marca todas las clases $\mathcal{C}_i$ en los ejemplos cubiertos por R^*. Para continuar el proceso, toma un nuevo ejemplo con alguna clase no cubierta para producir una nueva regla (figura 3.3c). Este proceso se repite hasta que todas las clases de todos los ejemplos han sido cubiertas. Una vez generadas todas las reglas, se aplica un algoritmo de filtrado que elimina reglas redundantes.

El algoritmo genético encargado de generar la regla óptima R^* a partir de R_{ini} sigue los principios establecidos en [92], realizando una búsqueda probabilística paralela en el espacio de reglas. Comenzando con una población P inicialmente vacía, va generando reglas aplicando los siguientes operadores[3]:

1. Creación(0.1): introduce nuevos individuos en P generalizando R_{ini}.

2. Generalización(0.8): genera una regla R' a partir de otra R seleccionada de P y generaliza su antecedente introduciendo "$\star$" o ampliando los límites B y B'. Si $C_q(R') > C_q(R)$, entonces R es reemplazada por R' en P; en caso contrario, R' es insertada en P.

[3]Entre paréntesis se indica la probabilidad de aplicación del operador

3. Cruce(0.1): aplica el cruce uniforme a dos reglas seleccionadas de P, obteniendo dos hijos que son insertados en la población.

La inserción de una nueva regla R' en la población se lleva a cabo siempre que P no supere un tamaño máximo preestablecido, en concreto 50 individuos. Si dicho umbral es superado, la nueva regla no se inserta, sino que sustituye a la regla R_{low} con menor $C_q(R_{low})$, siempre que $C_q(R_{low}) < C_q(R')$.

Los operadores anteriores se aplican hasta que, tras un número predefinido de iteraciones, no se haya generado ninguna regla con mayor fortaleza que el mejor de los individuos de $P \cup R_{ini}$. Cuando la búsqueda finaliza, el algoritmo genético devuelve el mejor individuo R^* de P, es decir, aquel que maximiza $C_q(R^*)$.

Las pruebas empíricas llevadas a cabo por Venturini [172] comparan el rendimiento de su herramienta frente a C4.5 y C4.5Rules [144]. Respecto a la precisión de las reglas, SIA obtuvo resultados comparables a los producidos por C4.5 en sus dos versiones, mostrando una mejora de tal precisión conforme aumentaba el número de ejemplos del fichero de entrenamiento. Sin embargo, el número de reglas aprendidas por SIA fue significativamente mayor al de los otros dos sistemas, sobre todo respecto a C4.5Rules. No obstante, la principal desventaja de este algoritmo es el elevado tiempo de ejecución que precisa para la extracción de las reglas.

3.2.4. GASSIST

Uno de los inconvenientes más importantes de las propuestas GABIL y GIL, y en general de los sistemas basados en la escuela de Pittsburgh, es el alto coste computacional que presentan. Ello es debido principalmente a que cada individuo de longitud variable contiene una solución completa al problema, lo cual obliga a evaluar éstos usando el conjunto de datos completo.

J. Bacardit y J. M. Garrell [17, 18] proponen un clasificador denominado GASSIST (*GA-based Classifier System*) basado en GABIL, que aborda eficazmente el problema del coste desde dos perspectivas: reduciendo el tamaño de los individuos mediante la generalización de las soluciones y realizando un aprendizaje incremental que evite la necesidad de usar todos los ejemplos para llevar a cabo la evaluación. Además, este

sistema amplia el dominio de aplicación introduciendo en la codificación tanto atributos discretos como continuos.

GASSIST hereda de GABIL la codificación, los operadores genéticos y la función de evaluación, aunque introduciendo algunos aspectos para mejorar su eficiencia y eficacia [17]. Así, para la representación de las reglas, GASSIST usa la codificación binaria para dominios simbólicos, resultando poblaciones de individuos de longitud variable, cada uno de los cuales mapea un conjunto de reglas. Esta codificación es adoptada para dominios continuos aplicando un algoritmo de discretización previo [16] que obtiene un conjunto de *micro-intervalos*, los cuales son tratados de forma similar a los valores de un atributo discreto. Respecto a los operadores genéticos, se aplican la mutación clásica y el cruce monopuntual o bipuntual adaptados a individuos de longitud variable. GASSIST también toma la función de evaluación $\varphi(i) = (porcentaje\ de\ aciertos)^2$ de GABIL, aunque añade un factor de penalización ($factorPen$) que sanciona a aquellos individuos cuyo número de reglas (r) supere un máximo (r_{max}), de forma que

$$\varphi'(i) = \begin{cases} \varphi(i) & Si\ r \leq r_{max} \\ \varphi(i) \times factorPen & Si\ r > r_{max} \end{cases} \tag{3.6}$$

donde $factorPen = max\{1 - 0.0005(r - r_{max}^2), 0.01\}$. Esta evaluación, junto a un método de comparación de individuos aplicado durante la fase de selección, logra un grado de generalización correcto.

Para evitar el crecimiento desmesurado de los individuos, los autores introducen un *operador de purga*, que se aplica tras la evaluación y cuyo propósito es eliminar reglas inútiles de los mismos. Este operador mejora la generalización de las reglas, pero puede provocar de manera indirecta el sobreajuste de los individuos al contener éstos soluciones muy específicas adaptadas a clasificar el conjunto de entrenamiento, lo que puede suponer una pérdida de rendimiento en la fase de test.

Como se ha mencionado anteriormente, el modo de aprendizaje es incremental, y consiste en dividir el conjunto de datos de entrenamiento en subconjuntos o segmentos y aplicar el algoritmo genético usando tales segmentos en lugar de todos los ejemplos. En [18] se proponen tres esquemas diferentes de aprendizaje incremental:

1. Aprendizaje incremental básico: divide el conjunto de datos en N segmentos uniformes y aplica $\frac{num\_iter}{N}$ iteraciones seguidas del algoritmo genético a cada segmento, siendo num_iter el número total de iteraciones.

2. Aprendizaje incremental con un segmento total: Añade una etapa al final del proceso que usa todos los ejemplos para evitar la pérdida de conocimiento global.

3. Aprendizaje incremental con segmentos alternantes: cambia de segmento en cada iteración para evitar la pérdida de conocimiento global sin añadir coste computacional.

Tras las pruebas empíricas realizadas, los autores concluyen que el aprendizaje con segmentos alternantes usando dos segmentos es una buena elección como configuración por defecto.

3.3. COGITO

COGITO [1] es una familia de algoritmos evolutivos cuyo propósito es obtener un conjunto de reglas de decisión jerárquicas capaz de clasificar un conjunto de datos con la mayor precisión posible en el contexto del aprendizaje supervisado. La mejora en términos de eficiencia y eficacia de COGITO ha supuesto la mayor parte de la motivación de esta investigación. Por ello, esta sección describe esta familia de algoritmos con mayor nivel de detalle que las anteriores propuestas. Aunque se trate de una serie de algoritmos, en adelante trataremos COGITO como una única herramienta y comentaremos sus diversas versiones en cuanto a codificación de los individuos y el tipo de reglas que genera.

Los dos aspectos determinantes en el buen funcionamiento de esta herramienta son la representación del conocimiento adquirido como reglas jerárquicas y la metodología seguida para generar esas reglas. COGITO, a diferencia de otras herramientas como C4.5, no divide el espacio por un sólo atributo, sino que extrae secuencialmente una región del espacio. Esta forma de obtener las reglas disminuye la probabilidad de que existan regiones contiguas con la misma clase. Por otro lado, la incorporación de la jerarquía al conjunto de reglas permite obtener regiones incluidas en otras, lo cual puede

reducir significativamente la complejidad de la representación al reducir el número de reglas debido al solapamiento entre ellas.

A diferencia de otras propuestas como GABIL o GIL, COGITO no restringe los valores de los atributos del conjunto de datos a un dominio simbólico, sino que trata a la vez atributos discretos y continuos. La búsqueda de una codificación apropiada para cada dominio, así como las variaciones en la representación de las reglas, ha generado diversas versiones de la herramienta, aplicando diferentes funciones codificadoras y operadores genéticos, aunque manteniendo la misma metodología. COGITO aplica un algoritmo evolutivo de cubrimiento secuencial típico [127], donde los individuos de la población son reglas de decisión codificadas. Antes de describir el algoritmo (sección 3.3.3), veremos las codificaciones y representaciones usadas en las distintas versiones de la herramienta.

3.3.1. Codificaciones

En una regla, el antecedente es una conjunción de condiciones, donde cada condición establece restricciones sobre los valores que un atributo puede tomar para que se cumpla el consecuente. Todas las codificaciones usadas en las distintas versiones de COGITO coinciden en la estructura. Cada individuo codifica a una única regla, asignando un gen o grupo de genes a cada condición.

Codificación Binaria

La primera versión de COGITO [8] utiliza la codificación binaria para representar tanto los atributos discretos como los continuos. Esta opción fue suscitada principalmente por su fácil implementación y por su justificación teórica mediante el Teorema de los Esquemas [92].

Para representar los atributos discretos, la codificación binaria es similar a la codificación usada por los algoritmos de aprendizaje de conceptos. Así, se asigna un bit a cada posible valor simbólico del atributo, indicando con un 1 o un 0 la presencia o ausencia de dicho valor en la condición respectivamente.

La codificación binaria de atributos continuos sin aplicar algún método de discretización previo, plantea un problema de pérdida de precisión debido al carácter infinito

del dominio de valores. ¿Cómo representar un dominio infinito de valores con un número finito de bits?. COGITO soluciona este problema apoyándose en que no todos los valores del dominio de un atributo continuo aparecen en el conjunto de datos. Así, el número de bits que codifica los posibles valores de un atributo viene determinado por la *Longitud de Codificación Mínima (L)* [154], cuya expresión viene dada por la ecuación 3.7.

$$L = \left\lceil lg_2 \left(1 + \frac{u - l}{\delta} \right) \right\rceil \tag{3.7}$$

$$\begin{aligned} u &= max\{x \mid x \in Dom\} \\ l &= min\{x \mid x \in Dom\} \\ \delta &= min\{|x_i - x_j| \ \mid \forall\, x_i, x_j \in Dom, x_i \neq y_j\} \end{aligned} \tag{3.8}$$

donde l y u son los valores mínimo y máximo del dominio de valores del atributo (Dom) respectivamente; y δ es la menor diferencia entre dos pares distintos de valores (x_i y x_j) presentes en el conjunto de datos para dicho atributo. En resumen, esta codificación produce valores para los individuos de la población que se encuentran entre los dos valores más cercanos de un atributo.

Codificación Real

El uso de la codificación binaria para representar atributos continuos plantea diversos problemas, lo que ha motivado que muchos autores hayan planteado utilizar algún tipo de codificación distinta para el manejo de tales atributos [14, 23, 56, 174, 175]. En este sentido, la codificación real parece más apropiada para codificar este tipo de dominio, simplemente por ser más natural y cercana al problema.

La codificación real pura es aquella que permite que un gen pueda tomar valores en todo el dominio real. Esto plantea dos problemas desde el punto de vista teórico: por un lado, al ser infinito el número de símbolos del alfabeto, el tamaño del espacio de búsqueda también lo es; y por otro lado, aunque estrechamente relacionado con el problema anterior, al ser el número de esquemas infinito, coexistirán mucho esquemas sintácticamente distintos con el mismo significado. Ambos problemas se solventan permitiendo únicamente la utilización de algunos valores reales, transformando el espacio

real en un espacio real discreto y por tanto un alfabeto finito [1]. Como muestra la ecuación 3.9, para calcular cuales son los valores del alfabeto (Ω_i) para un atributo (a_i), se toma el límite inferior (l_i) del rango de valores del atributo en el conjunto de datos y se le suma una cantidad $K\delta_i$, donde K es una constante entera y δ_i es el incremento mínimo entre dos valores consecutivos en el alfabeto. El valor mayor valor de este alfabeto es el límite superior (u_i) del rango de valores del atributo, cumpliéndose siempre que $l_i + K\delta_i \leq u_i$.

$$\Omega_i = \{l_i, l_i + \delta_i, l_i + 2\delta_i, \ldots, l_i + K\delta_i, \ldots, u_i\} \tag{3.9}$$

J. C. Riquelme et al. [153] desarrollaron una versión de COGITO con codificación real, que sólo operaba con dominios continuos, aplicando la idea de espacio real discreto anteriormente descrita, donde el valor de δ_i era un 1 % del rango del atributo a_i, es decir $\delta_i = 0.1(u_i - l_i)$. Cada regla es codificada por un único individuo, siendo cada condición del antecedente representada por tres genes y la clase por un único gen. La figura 3.4 ilustra un ejemplo de individuo con codificación real y la regla a la cual codifica.

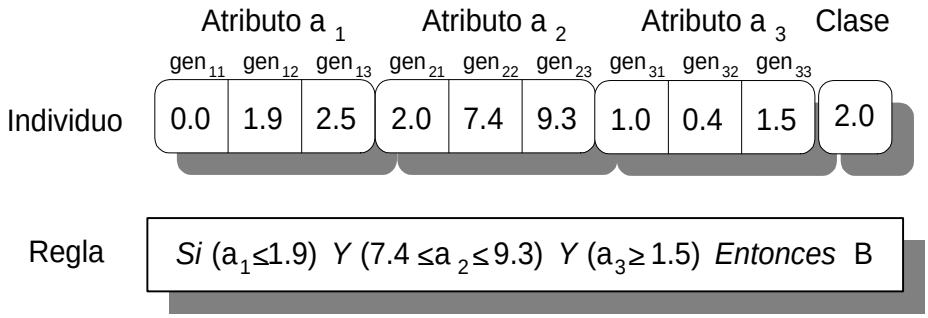


Figura 3.4: Codificación Real.

De cada terna de genes que representa una condición sobre el atributo a_i, el primero determina el operador que se aplica y los otros dos los valores de la condición. La interpretación de estos genes es la siguiente:

- $gen_{i1} = 0.0 \Rightarrow a_i \leq gen_{i2}$ (el gen_{i3} no se tiene en cuenta)
- $gen_{i1} = 1.0 \Rightarrow a_i \geq gen_{i3}$ (el gen_{i2} no se tiene en cuenta)
- $gen_{i1} = 2.0 \Rightarrow gen_{i2} \leq a_i \leq gen_{i3}$

Aunque esta versión de COGITO sólo admite atributos continuos, supone la clase discreta, siendo su codificación una simple enumeración de los distintos valores nominales de ésta.

Los operadores genéticos aplicados son el cruce aleatorio multipunto [2], tres cruces reales segmentados diseñados ex profeso (intermedio, forzado al mínimo y forzado al máximo) y dos tipos de mutación (incremental y forzada al límite)[153]. Además, el módulo evolutivo incorpora elitismo, es decir, el mejor individuo de cada generación es replicado directamente a la siguiente.

Codificación Híbrida

Una de las principales razones que justifican el uso de algoritmos evolutivos con codificación real es la precisión para representar valores de los atributos. Otra razón es la facilidad para explorar graduación de funciones con atributos continuos [56]. Como se describe en la sección anterior, el uso de la codificación real para representar atributos continuos es una buena elección en la práctica. Sin embargo, el hecho de no tratar los atributos discretos es un inconveniente evidente de la versión de COGITO con codificación real. Este problema hubiera sido relativamente fácil de solventar con una simple enumeración de las todas combinaciones de valores de cada atributo discreto, aplicando una codificación similar a la empleada para representar la clase. Sin embargo, parece que la codificación binaria es más adecuada para el tratamiento de este tipo de dominios. Siguiendo este razonamiento, J. C. Riquelme et al. [156] optan por utilizar una representación que denominan *codificación híbrida*, la cual aplica codificación binaria para los atributos discretos y real para los continuos.

La codificación binaria es idéntica a la usada en la primera versión de COGITO para atributos discretos, donde cada posible valor es codificado por un bit que indica la presencia o ausencia de dicho valor. Cuando todos los genes toman valor 1 a la vez en un individuo, ese atributo no aparece en la regla, ya que su valor no influirá en los ejemplos clasificados por la misma. Por otra parte, la clase no es tratada como un atributo discreto más, sino que se adopta la codificación de anteriores versiones, donde se enumeran los valores que dicha clase puede tomar y se representa con un único gen.

Respecto a la codificación real aplicada, ésta difiere de la descrita anteriormente. En

este caso, cada atributo continuo no es codificado por tres genes, sino que se opta por una representación mas sencilla y natural basada en intervalos. Así, la condición sobre un atributo continuo (a_i) es de la forma $a_i \in [li_i, ls_i]$, donde se define un gen real para el límite inferior del intervalo (li_i) y otro gen real para el límite superior (ls_i). Si $li_i = min(a_i)$ o $ls_i = max(a_i)$, entonces ese límite no aparecerá en la regla, interpretándose el intervalo como $(-\infty, ls_i]$ o $[li_i, +\infty)$ respectivamente. Esta representación permite incluso ambos límites ($a_i \in (-\infty, +\infty)$), en cuyo caso la condición no aparecerá en la regla puesto que el atributo podrá tomar cualquier valor del rango.

La figura 3.5 muestra un ejemplo sencillo de individuo híbrido con un atributo de cada tipo y una clase, donde el atributo a_1 toma valores en el intervalo $[1.0, 6.2]$ y el atributo a_2 puede tomar cinco valores discretos diferentes.

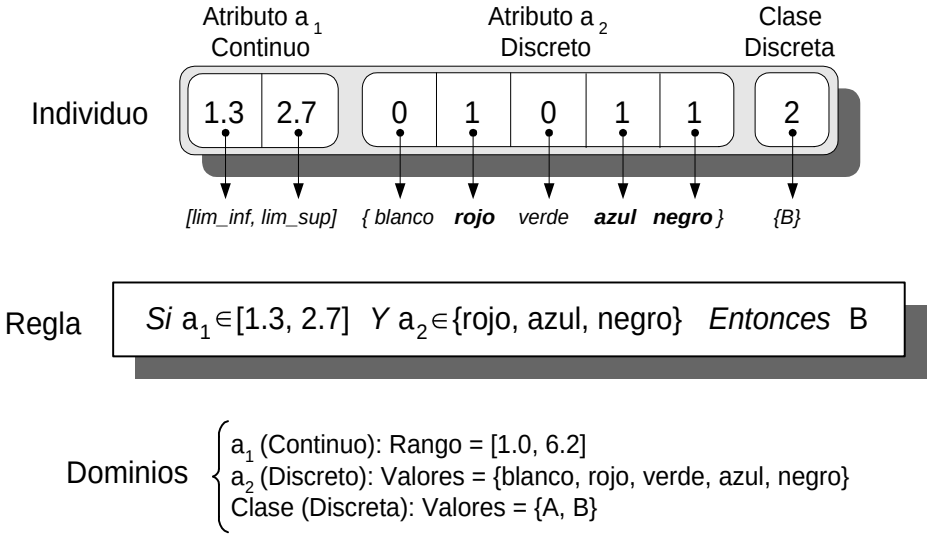


Figura 3.5: Ejemplo de codificación híbrida.

El individuo híbrido ilustrado por la figura 3.5 codifica un tipo de regla que define un hiperrectángulo paralelo a los ejes. Sin embargo, esta codificación permite otras representaciones de las reglas incluyendo pequeñas modificaciones en los individuos. Estas representaciones alternativas serán descritas en la sección 3.3.2.

Codificación Indexada

El principal problema de usar valores reales para representar los límites de un intervalo es que permite que los genes adopten cualquier valor del dominio real, cuando

en realidad, no todos los valores de ese dominio son buenos candidatos para tal propósito. En otras palabras, el hecho de no restringir el conjunto de valores reales que los límites de los intervalos pueden tomar hace que existan un gran número de esquemas (teóricamente infinitos) que comparten exactamente el mismo significado. En este sentido, la elección de un único representante por cada grupo de esquemas semánticamente idénticos reduciría significativamente la cardinalidad del conjunto de posibles esquemas, con la consiguiente reducción del espacio de búsqueda.

Por este motivo, para obtener el conjunto de posibles límites de intervalo para cada atributo continuo, en la codificación indexada [152] se aplica un sencillo algoritmo de discretización diseñado *ad hoc* [1]. A los valores obtenidos por este algoritmo se les denomina cortes, los cuales son almacenados en un vector ordenado. Así, se tendrá un vector de cortes por cada atributo continuo del conjunto de datos, de modo que, los límites de cada intervalo sólo pueden tomar valores de ese vector. La codificación indexada representa cada atributo continuo con dos genes, uno por cada límite del intervalo que codifica. Cada gen contiene la posición o índice del corte correspondiente en el vector. El uso de esos índices como genes da nombre a esta codificación.

Respecto a los atributos discretos, la representación de éstos es idéntica a la codificación híbrida. Así pues, un individuo codificado usando la representación indexada es similar a un individuo híbrido donde se han reemplazado los genes reales por números enteros. La figura 3.6 muestra un ejemplo de codificación indexada, donde el *Individuo 1* codifica la misma regla que el ejemplo de codificación híbrida de la figura 3.5.

Una versión mejorada de COGITO con codificación indexada, denominada COGITO+, fue obtenida aplicando el algoritmo de discretización USD [72], obtenido como resultado de esta investigación.

3.3.2. Representaciones de las reglas

Todas las versiones de COGITO generan reglas de decisión jerárquicas, las cuales deben de ser evaluadas en el mismo orden en que fueron obtenidas. Esta jerarquía permite que puedan existir reglas incluidas en otras, lo cual reduce considerablemente el numero final de reglas frente a otras herramientas. Sin embargo, el hecho de que

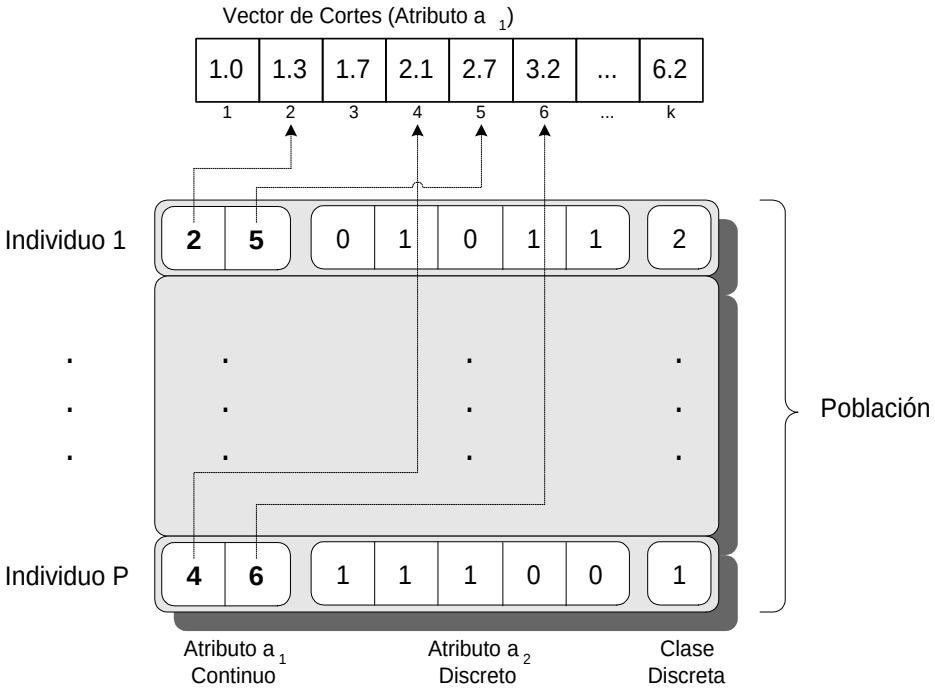


Figura 3.6: Ejemplo de codificación indexada.

sean reglas de decisión jerárquicas no condiciona la forma que dichas reglas tengan en el hiperespacio definido por los atributos de la bases de datos. Así, diferentes interpretaciones de la codificación dan lugar a diversas representaciones de las reglas. En concreto COGITO puede adoptar tres representaciones distintas: hiperrectángulos paralelos a los ejes, hiperrectángulos oblicuos e hiperelipses [5]. Estas tres representaciones fueron implementadas mediante codificación híbrida, aunque las dos últimas fueron usadas fundamentalmente para atributos continuos.

Hiperrectángulos paralelos a los ejes

Todas las versiones de la herramienta descritas anteriormente obtienen reglas en forma de hiperrectángulos paralelos a los ejes, la cual es la representación más inteligible de las tres.

La interpretación de la codificación híbrida fue descrita en la sección anterior, donde la condición establecida por la regla para cada atributo continuo es codificada mediante dos genes. Tales genes son números reales que representan los límites inferior

y superior del intervalo. Esta codificación puede observarse gráficamente en la figura 3.5, donde los dos primeros genes codifican al intervalo exigido al atributo a_1. La inclusión de dos límites en las condiciones, junto a la posibilidad de descartar uno de ellos, da mayor versatilidad a la representación mediante hiperrectángulos paralelos a los ejes frente a la representación mediante árboles de decisión.

Hiperrectángulos oblicuos

Esta representación se planteó con el objetivo de generar reglas más versátiles que los rectángulos paralelos a los ejes, de manera que las condiciones que se establecieran para los atributos continuos no fueran independientes, sino combinaciones lineales de dichos atributos [7, 155]. Los hiperrectángulos oblicuos se obtienen a partir de los paralelos a los ejes, aplicando rotaciones a éstos respecto a su centro de gravedad. En general, para rotar un hiperrectángulo es necesario aplicar un ángulo de rotación por cada par de ejes. Por tanto, para codificar un individuo con m atributos son necesarios $2m$ genes para los intervalos (límites inferior y superior), $m-1$ genes para los ángulos y un gen para la clase, en total $3m$ genes. La figura 3.7 muestra un ejemplo de codificación de una regla oblicua (R′) con dos atributos continuos y una rotación de ángulo α_1, generada a partir de la una regla paralela a los ejes (R).

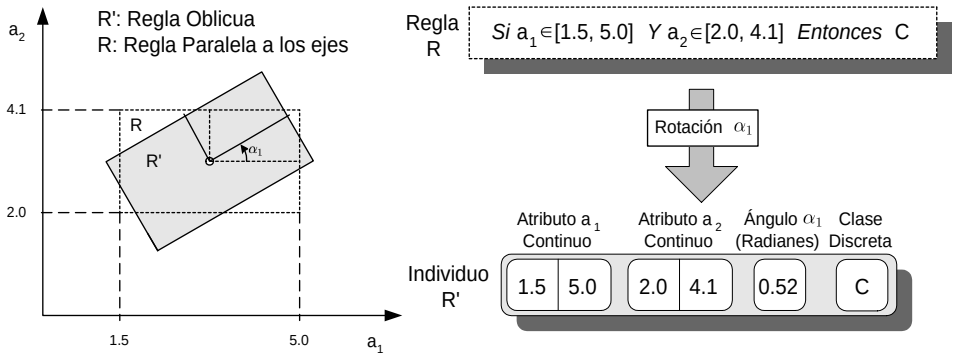


Figura 3.7: Regla Oblicua.

Una cuestión importante es cómo se evalúan los ejemplos respecto a las reglas oblicuas. Sea $r' = (li_1, ls_1, li_2, ls_2, \ldots, li_m, ls_m, \alpha_1, \ldots, \alpha_{m-1}, c_r)$ la regla oblicua obtenida a partir de la paralela $r = (li_1, ls_1, li_2, ls_2, \ldots, c_r)$. Entonces, se considera que un ejemplo $e = (a_1, a_2, \ldots, a_m, c_e)$ es cubierto por una regla r' si rotando el ejemplo e con ángulos

$(-\alpha_1, \ldots, -\alpha_{m-1})$ respecto al centro de gravedad de la regla, este nuevo ejemplo e' es cubierto por la regla paralela r.

Hiperelipses

De forma intuitiva, se puede suponer que, en general, los ejemplos de una misma clase tienden a agruparse alrededor de uno o más centros de gravedad. Esta idea plantea que si se representan las reglas con forma de hiperelipses multidimensionales, podemos obtener mayor precisión en la clasificación respecto a las reglas con forma rectangular [1].

La codificación de este tipo de individuos se basa en la ecuación 3.10, de modo que para m atributos (a_i) se necesitan m valores para el centro (c_i) y m valores para los semiejes (s_i). Por tanto, estos individuos tienen dos genes reales por atributo y uno más para la clase.

$$\frac{(a_1 - c_1)^2}{s_1} + \frac{(a_2 - c_2)^2}{s_2} + \cdots + \frac{(a_m - c_m)^2}{s_m} \leq 1 \tag{3.10}$$

Por otra parte, un ejemplo es cubierto por una regla hiperelíptica cuando satisface la ecuación 3.10.

Crítica a las representaciones de las reglas

Como se muestra en [5], junto a otros trabajos mencionados anteriormente, ninguna de estas representaciones puede considerarse la más adecuada para todas las bases de datos. La mejora de una representación frente a las otras dos depende de la distribución de los datos, pero dado que dicha distribución no es conocida generalmente, no es posible optar a priori por la mejor de éstas. Una posible solución a este problema es ejecutar COGITO con cada representación y elegir posteriormente la más adecuada. Sin embargo, dado el coste computacional que esto supondría, J.S. Aguilar [1] opta por la representación más sencilla e inteligible, es decir, hiperrectángulos paralelos a los ejes. Siguiendo el mismo razonamiento, las siguiente secciones se desarrollarán adoptando esta representación de la reglas.

```
COGITO
   Entrada:   D: Conjunto de ejemplos
   Salida:    R: Conjunto de Reglas
Comienzo
1.   Mientras existan ejemplos por cubrir en 𝒟
2.        Inicialización de la población con P individuos
3.        Repetir G veces // G = número de generaciones
4.             Evaluación de todos los individuos de la población
5.             Selección del mejor individuo
6.             Réplicas
7.             Cruce y Mutación específicos
8.        Fin Repetir
9.        Añadir el mejor al conjunto de reglas R
10.       Eliminar los ejemplos cubiertos
11.  Fin Mientras
Fin
```

Figura 3.8: Pseudocódigo de COGITO.

3.3.3. Algoritmo

El algoritmo evolutivo que aplica COGITO es un método de cubrimiento secuencial, el cual es mostrado por la figura 3.8. En cada iteración se inicializa la población de individuos (línea 2) y se ejecuta el proceso evolutivo (líneas 3–8). Tras dicho proceso, se selecciona el mejor individuo (línea 9), generando así una regla por cada iteración, la cual es usada para eliminar los ejemplos cubiertos del conjunto de datos de entrenamiento (línea 10) [172]. El algoritmo termina cuando todos los ejemplos han sido cubiertos, es decir, cuando el conjunto de datos es vacío. Así, el método seguido obtiene un conjunto de reglas de decisión jerárquicas que han de ser evaluadas durante la clasificación en el mismo orden que fueron generadas.

Inicialización

La inicialización de la población se lleva a cabo mediante la selección aleatoria de P ejemplos del conjunto de datos, generando una regla por cada uno de estos ejemplos y forzando que ésta lo cubra. De este modo se obtiene una población inicial de P individuos que cubren al menos a un ejemplo del conjunto de datos.

Evaluación

La evaluación de los individuos de la población es un factor crítico en todo algoritmo evolutivo. COGITO evalúa la población de individuos realizando un recorrido secuencial de la misma y aplicando la función de evaluación a cada individuo. Dicha función cuantifica la calidad de un individuo respecto al resto de la población, asignándole a éste un valor numérico, denominado *bondad*, según el número de ejemplos que clasifique correcta e incorrectamente. En concreto, la función de evaluación aplicada por COGITO es mostrada por la ecuación 3.11.

$$f(r) = 2(N - E(r)) + A(r) + cobertura(r) \tag{3.11}$$

donde N es el número de ejemplos del conjunto de datos; $E(r)$ es el número de errores cometidos por r (*i.e.* el número de ejemplos cubiertos por la regla pero erróneamente clasificados); $A(r)$ es el número de aciertos de r (*i.e.* el número de ejemplos clasificados correctamente por la regla); y $cobertura(r)$ es el volumen normalizado de la región que cubre la regla r.

La evaluación de la población requiere aplicar la función de evaluación a todos los individuos de la misma para asignarle a cada uno su valor de bondad. Para ello, es necesario realizar un conteo de los aciertos y errores de cada individuo. Un ejemplo es cubierto por un individuo cuando sus valores satisfacen todas las condiciones del antecedente de la regla que el individuo representa. Asimismo, un ejemplo es clasificado correctamente por un individuo cuando, además de ser cubierto, ambos tienen la misma clase, contabilizando un acierto por parte de la regla. En caso de que un ejemplo sea cubierto pero las clases no coincidan, entonces la regla comete un error. El recorrido lineal para evaluar un individuo es ilustrado por el ejemplo de la figura 3.9.

En general, para un conjunto de datos de N ejemplos y m atributos, el coste computacional de evaluar una población de P individuos es de $\Theta(PNm)$. Teniendo en cuenta que la evaluación de la población es realizada una vez por cada generación, el coste total de hacer todas las evaluaciones durante el proceso evolutivo es de $\Theta(GPNm)$, siendo G el número de generaciones. J.S. Aguilar [1] afirmó que más del 85 % del tiempo de cálculo se dedica a la evaluación de los individuos de la población genética. Esto nos da una idea de la importancia de la evaluación dentro del proceso evolutivo desde

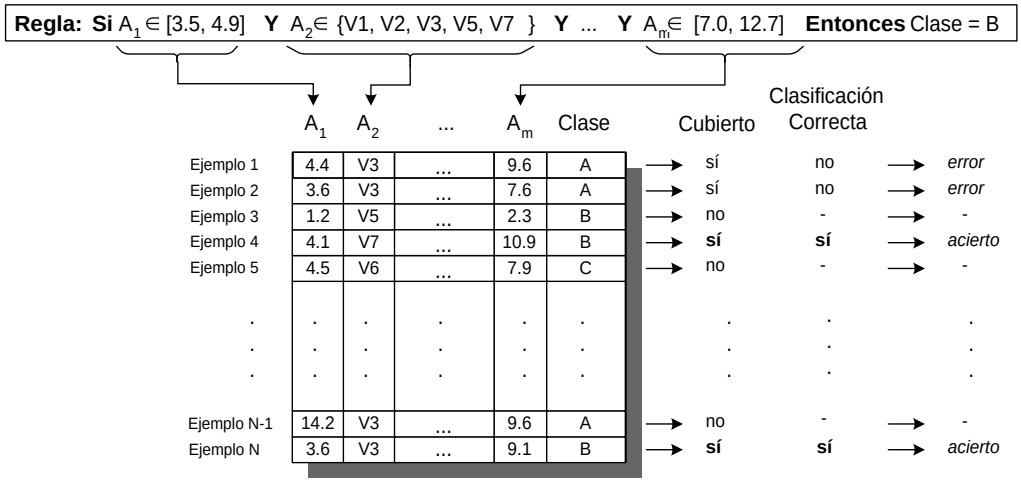


Figura 3.9: Ejemplo de evaluación lineal.

el punto de vista de la eficiencia del algoritmo, siendo uno de los aspectos abordados en esta investigación.

Reemplazo

Denominamos reemplazo a la selección, réplica y reproducción (cruce) de los individuos de una población para formar la siguiente (líneas 5–7 de la figura 3.8). COGITO incluye elitismo, replicando el mejor individuo de la población a la siguiente sin ser mutado. Un porcentaje de los individuos de la nueva población es obtenido mediante réplicas, es decir, copias de individuos seleccionados usando el método de la ruleta de la fortuna [44]. El resto de la población es formada mediante cruces, seleccionando los padres utilizando de nuevo el método de la ruleta. Tanto las réplicas como la descendencia obtenida mediante los cruces son mutadas dependiendo de la probabilidad de mutación por individuo que se aplique.

Capítulo 4

Discretización Supervisada No Paramétrica

No rompas el silencio si no es para mejorarlo.
ANÓNIMO.

4.1. Introducción

Uno de los aspectos fundamentales en la extracción eficiente de conocimiento útil es la calidad de las fuentes de información. La limpieza, validez y representatividad de los datos, así como la cardinalidad y dimensionalidad de los mismos son determinantes a la hora de aplicar con garantías un algoritmo de minería de datos. Las técnicas que se encargan de garantizar, en la medida de lo posible, estas características se enmarcan dentro de las fases de *selección* y *preprocesado* del proceso *KDD* descrito en el Capítulo 2.

En ocasiones, las técnicas de selección y preprocesado no proporcionan un modelo adecuado para su tratamiento directo por parte del algoritmo de minería. Ello obliga a la aplicación de alguna trasformación adicional que convierta los datos en un modelo analítico adaptado al algoritmo de aprendizaje concreto. En este sentido, muchos de estos algoritmos necesitan operar con un espacio de atributos discreto, lo que hace imprescindible la aplicación de algún método de discretización que disminuya la cardinalidad de los valores que los atributos continuos pueden tomar. Este punto es especialmente delicado, ya que la discretización puede ocasionar pérdida de información

relevante para el aprendizaje, deteriorando la fiabilidad del modelo posteriormente generado.

Aunque existen métodos de discretización generales, dada la diversidad de problemas que se plantean en el área del aprendizaje automático, es prácticamente imposible encontrar un único método que dé solución a todos esos problemas y a la vez sea eficaz y eficiente. Por ello, se han diseñado diversos algoritmos de discretización para solucionar problemas específicos en el campo del aprendizaje automático [15, 21, 52, 57, 108], algunos de los cuales son aplicados exclusivamente en el marco del aprendizaje supervisado.

Este planteamiento, trasladado al desarrollo de la propuesta central de esta tesis, la herramienta HIDER, nos llevó a plantearnos el diseño del algoritmo presentado en este capítulo: Discretización Supervisada No Paramétrica (*Unparametrized Supervised Discretization*), en adelante USD [72, 73, 74].

4.2. Motivación y objetivos

Como se detallará en el siguiente capítulo, la herramienta HIDER no opera directamente con dominios de tipo continuo, principalmente debido a la codificación de los individuos en el algoritmo evolutivo. En concreto, dicha codificación maneja los valores de los atributos continuos como conjuntos finitos de intervalos disjuntos y, por tanto, se requiere una discretización previa que genere tales intervalos. Tras estudiar las diferentes propuestas recogidas en la bibliografía, el método que, en principio, mejor se adecuaba a los requerimientos de HIDER fue el denominado 1R[1] [94]. Sin embargo, las pruebas realizadas con este método no fueron suficientemente satisfactorias, principalmente por dos razones: por un lado, los modelos generados a posteriori por HIDER obtenían una tasa de error y complejidad superiores a los rangos que estimábamos adecuados, debido a que 1R no tiene en cuenta explícitamente el error que comete al incluir en un mismo intervalo ejemplos con diferentes clases; y por otro lado, la precisión y número de intervalos obtenidos dependían de un parámetro de usuario, denominado *SMALL*. Este parámetro era distinto para cada base de datos y, por tanto,

[1] *1–Rule* (véase la sección 2.4.4, página 36)

condicionaba indirectamente el error cometido por el modelo final.

Así pues, nos planteamos el diseño de un algoritmo de discretización orientado a la generación de reglas de decisión, el cual debe cumplir algunas premisas:

1. **Supervisado**. El método debe tener en cuenta la clase de los ejemplos.

2. **Intervalos generados**. Debe obtener un conjunto finito de intervalos disjuntos para cada atributo continuo que cubra todo el rango de valores.

3. **Precisión**. El error introducido por la discretización debe ser mínimo, ya que tal error será arrastrado por el algoritmo de aprendizaje y por tanto podría producir modelos de menor precisión que aquellos generados por herramientas sin discretización previa.

4. **Complejidad**. A menor número de intervalos, menor será el espacio de búsqueda para el algoritmo evolutivo. Por tanto, el cardinal del conjunto de intervalos ha de ser lo más reducido posible sin penalizar la precisión. Este punto no es esencial, ya que el algoritmo de aprendizaje siempre tendrá la opción de unir intervalos adyacentes al generar una regla para establecer las fronteras de decisión.

5. **Eficiencia**. El método debe ser de un coste computacional reducido para no incrementar aún más el coste ya elevado del algoritmo evolutivo.

6. **No parametrizado**. El algoritmo no debe depender de ningún parámetro de usuario que pueda influir en la calidad de los resultados finales.

7. **Determinista**. Es recomendable que el método no esté sujeto a condiciones azarosas.

USD es el resultado del estudio de diversas propuestas teniendo en cuenta los objetivos anteriormente descritos. Antes de describir el algoritmo, definiremos ciertos conceptos necesarios para la mejor comprensión del mismo.

4.3. Definiciones

Definición 4.1 (Valor puro e impuro). *Decimos que un valor es puro, para un atributo cualquiera, cuando tiene la misma clase para todas las apariciones en los distintos ejemplos del conjunto de datos. En caso contrario, es decir, cuando dos ejemplos tienen el mismo valor para un mismo atributo pero distinta clase, se denominará valor impuro.*

Definición 4.2 (Corte). *Definimos los cortes como los valores que delimitan los intervalos calculados a lo largo del proceso de discretización. Es decir, para un determinado atributo a_j, el i-ésimo corte (c_i) será el límite superior abierto del intervalo I_i y el límite inferior cerrado del intervalo I_{i+1}. Cada corte c_i es calculado como la semisuma del mayor valor del atributo a_j para los ejemplos contenidos en el intervalo I_i y el menor valor del atributo a_j para los ejemplos contenidos en el intervalo I_{i+1}. La figura 4.1 muestra gráficamente esta idea.*

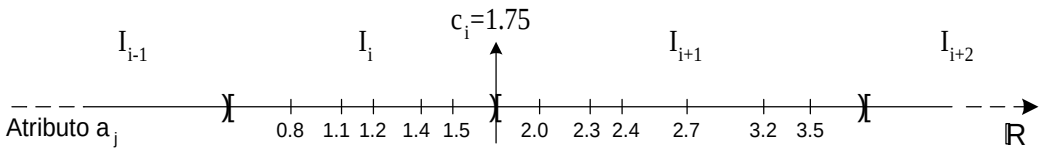


Figura 4.1: Ejemplo de cálculo de un corte simple.

Definición 4.3 (Intervalo puro e impuro). *Decimos que un intervalo es puro, para un atributo cualquiera, cuando todos los valores que contiene pertenecen a la misma clase. En caso contrario decimos que el intervalo es impuro.*

Definición 4.4 (Clase mayoritaria). *La clase mayoritaria de un intervalo es la clase con más apariciones dentro del intervalo. Así, el número de apariciones de la clase mayoritaria en un intervalo puro será igual al número de ejemplos que contenga el intervalo.*

Definición 4.5 (Aciertos y errores de un intervalo). *Se denomina aciertos de un intervalo al número de ejemplos contenidos en él cuya clase es igual a la clase mayoritaria de tal intervalo. Análogamente, se denomina errores de un intervalo al número de ejemplos contenidos en él con clase distinta a la clase mayoritaria de dicho intervalo.*

Definición 4.6 (Bondad de un intervalo). *Se define la bondad de un intervalo como la relación entre los aciertos y los errores de dicho intervalo. Por tanto, la bondad de un intervalo es la medida de la pureza del mismo. La ecuación que define la bondad puede variar dependiendo de la penalización por error que queramos considerar.*

Ejemplo 4.1. *Una posible expresión de la bondad es mostrada en la ecuación 4.1, en la que la penalización por error es alta, al encontrarse el número de errores en el denominador.*

$$Bondad = \frac{aciertos}{1 + errores} \tag{4.1}$$

La ecuación 4.1 acota el valor de la bondad en el intervalo $(0, 1]$*. Así, el valor 1 indica que el intervalo es puro. Nótese que no es posible que un intervalo sea totalmente impuro, i.e. bondad igual a 0, ya que esto indicaría que sólo contiene errores, lo cual no es posible dado que siempre existe algún valor*[2] *que determina la clase mayoritaria y éste se contabiliza como un acierto. Por tanto, en el peor de los casos tendremos un intervalo con un único acierto y* $|\mathcal{C}| - 1$ *errores, donde* $|\mathcal{C}|$ *es el número de clases diferentes, quedando la bondad acotada inferiormente por* $\frac{1}{|\mathcal{C}|}$*.*

4.4. Algoritmo

El objetivo que el algoritmo USD persigue es dividir los atributos continuos en intervalos de máxima bondad, de forma que la bondad media de todos los intervalos finales para un determinado atributo sea lo más alta posible. Todo el proceso se realiza sin la necesidad de que el usuario introduzca ningún parámetro ni información adicional. La forma en que se calculan los intervalos hacen que el algoritmo sea determinista. La descripción del algoritmo USD aparece en la figura 4.2.

El algoritmo toma como entrada el conjunto de datos y devuelve un conjuntos de cortes que son los límites de los intervalos resultantes de la discretización. El procedimiento principal se divide en dos partes bien diferenciadas. La primera parte calcula los intervalos iniciales (línea 2) que posteriormente serán refinados en la segunda parte (líneas 3–19) dependiendo de las bondades que se obtengan tras realizar las dos

[2] *Los intervalos generados por* USD *siempre contienen al menos un valor del conjunto de datos, no permitiendo la existencia de intervalos vacíos.*

USD ($\mathcal{D}$, $\mathcal{C}$)
Entrada: $\mathcal{D}$: Conjunto de ejemplos
Salida: $\mathcal{C}$: Conjunto de Cortes
1. **Comienzo**
2. InitializaCortes($\mathcal{D}$, $\mathcal{C}$)
3. **Para** cada atributo continuo de $\mathcal{D}$
4. **Para** cada intervalo I_i excepto el último
5. **Si** CondiciónDeUnión(I_i, I_{i+1})=CIERTO
6. MarcarPosibleUnión(I_i, I_{i+1}) y su bondad
7. **Fin si**
8. **Fin para**
9. **Mientras** haya posibles uniones
10. i := Unir intervalos con posible-unión marcada y máxima bondad
11. **Si** CondiciónDeUnión(I_i, I_{i+1})=CIERTO
12. MarcarPosibleUnión(I_i, I_{i+1}) y su bondad
13. **Fin si**
14. **Si** CondiciónDeUnión(I_{i-1}, I_i)=CIERTO
15. MarcarPosibleUnión(I_{i-1}, I_i) y su bondad
16. **Fin si**
17. **Fin mientras**
18. **Fin Para**
19. **Fin**

20. **CondiciónDeUnión (**I_i**,** I_{i+1}**)** =
21. {(I_i tiene la misma clase mayoritaria que I_{i+1}) **O** (hay empate en I_i o I_{i+1})}
22. **Y** {la bondad de la unión de I_i y I_{i+1} es mayor o igual a la media de la bondad de I_i y la bondad de I_{i+1}}

Figura 4.2: Algoritmo USD.

acciones posibles: unir dos intervalos consecutivos eliminando el corte que los separa o dejarlos independientes.

4.4.1. Cálculo de los intervalos iniciales

El cálculo de los intervalos iniciales lo lleva a cabo el procedimiento $InicializaCortes$ (figura 4.2, línea 2), el cual podría constituir un método de discretización simple por sí solo. Este proceso maximiza la pureza de los intervalos con el propósito de obtener las mejores bondades posibles, es decir, obtiene intervalos tan puros como sea posible

Procedimiento InicializaCortes($\mathcal{D}$, $\mathcal{C}$)
Entrada: $\mathcal{D}$: Conjunto de ejemplos
Salida: $\mathcal{C}$: Conjunto de Cortes
1. **Comienzo**
2. **Para** cada atributo continuo a de $\mathcal{D}$
3. Ordena(a) // Ordena por valor y clase
4. i := 1
5. **Mientras** i<|$\mathcal{D}$|
6. **Si** $v_i \neq v_{i+1}$ Y {$Clase(v_i) \neq Clase(v_{i+1})$ O $Puro(v_i) \neq Puro(v_{i+1})$}
7. $\mathcal{C}$= $\mathcal{C} \oplus \{\frac{v_i+v_{i+1}}{2}\}$
8. **Fin si**
9. i:=i+1
10. **Fin Mientras**
11. **Fin para**
12. **Fin**

Figura 4.3: Cálculo de intervalos iniciales en USD.

independientemente de los ejemplos que contenga. Esto hace que el número de intervalos sea relativamente elevado, aunque este aspecto no supone una desventaja puesto que, posteriormente, el proceso de refinamiento reducirá considerablemente dicho número. Al finalizar el procedimiento $InicializaCortes(\mathcal{D}, \mathcal{C})$, el parámetro de salida $\mathcal{C}$ contendrá un conjunto de cortes iniciales por cada atributo continuo del conjunto de datos de entrada $\mathcal{D}$.

La figura 4.3 muestra el pseudocódigo del algoritmo de cálculo de intervalos iniciales. Como se puede observar, el método trata cada atributo por separado. Antes de establecer los cortes, es necesario ordenar[3]el atributo por valor y clase (línea 3), de modo que en caso de que un valor sea impuro, las apariciones de éste queden ordenadas por la clase. Posteriormente se recorre el conjunto ordenado de valores, estableciendo un corte en la semisuma de dos valores consecutivos (línea 7) si se cumple la condición de corte (línea 6), donde $Clase(v_i)$ es la clase del i-ésimo ejemplo para ese valor del atributo y $Puro(v_i)$ determina si el valor es puro o impuro. Con esta condición se maximiza la pureza de los intervalos, ya que un corte es fijado cuando dos valores consecutivos y distintos tienen distinta clase o distinta condición de pureza, es decir, uno es puro y el otro impuro. El hecho de establecer los cortes en el punto medio entre dos

[3]Esta ordenación se lleva a cabo aplicando el método *QuickSort* [89].

N	a_k	$frec(C_A)$	$frec(C_B)$	Clase Mayoritaria
1	1.0	5	0	A
2	1.2	4	0	A
3	1.4	0	3	B
4	1.6	0	4	B
5	1.8	6	0	A
6	2.0	5	1	A
7	2.2	5	2	A
8	2.4	0	4	B
9	2.6	1	6	B
10	3.0	1	5	B
11	3.2	0	4	B
12	3.4	6	2	A
13	3.6	1	3	B
14	3.8	8	1	A
15	4.0	6	0	A
16	4.2	2	7	B
17	4.4	8	0	A

Tabla 4.1: Ejemplo de conjunto de datos.

valores consecutivos asegura que ningún corte coincidirá con valores presentes en el conjunto de datos. Como se mencionó con anterioridad, el conjunto de cortes fijados al final del procedimiento maximiza la pureza de los intervalos a los cuales delimitan con independencia del número de ejemplos que contengan.

Ejemplo 4.2. *Para facilitar la comprensión del proceso de cálculo de los intervalos iniciales, planteamos un ejemplo sencillo. Supongamos que tenemos los datos de la tabla 4.1, que representa un atributo particular (a_k) de un conjunto de datos con 100 ejemplos y dos clases. Nótese que el conjunto de datos ha sido ordenado previamente por el atributo en cuestión. El significado de cada columna es el siguiente: la primera enumera los valores distintos del conjunto de datos; la segunda columna contiene el valor propiamente dicho del atributo a_k; las dos siguientes dan la frecuencia de aparición según la clase; y por último se destaca la clase mayoritaria.*

La tabla 4.1 recoge todas las posibles situaciones que se pueden dar en un conjunto de datos. Las situaciones corresponden a todas las combinaciones entre valores consecutivos que tienen clase igual y distinta y, a su vez, el valor es puro e impuro. En general, el número de posibilidades viene dado por la expresión $4|C|^2$, donde $|C|$ es el

v_i	v_{i+1}	$Clase(v_i)$	$Clase(v_{i+1})$	$Puro(v_i)$	$Puro(v_{i+1})$	¿Fijar Corte?	Cortes
1.0	1.2	A	A	Puro	Puro	No	–
1.2	1.4	A	B	Puro	Puro	Sí	1.3
1.4	1.6	B	B	Puro	Puro	No	–
1.6	1.8	B	A	Puro	Puro	Sí	1.7
1.8	2.0	A	A	Puro	Impuro	Sí	1.9
2.0	2.2	A	A	Impuro	Impuro	No	–
2.2	2.4	A	B	Impuro	Puro	Sí	2.3
2.4	2.6	B	B	Puro	Impuro	Sí	2.5
2.6	3.0	B	B	Impuro	Impuro	No	–
3.0	3.2	B	B	Impuro	Puro	Sí	3.1
3.2	3.4	B	A	Puro	Impuro	Sí	3.3
3.4	3.6	A	B	Impuro	Impuro	Sí	3.5
3.6	3.8	B	A	Impuro	Impuro	Sí	3.7
3.8	4.0	A	A	Impuro	Puro	Sí	3.9
4.0	4.2	A	B	Puro	Impuro	Sí	4.1
4.2	4.4	B	A	Impuro	Puro	Sí	4.3

Tabla 4.2: Ejemplo de fijación de cortes en USD.

número de clases diferentes de la base de datos. A partir de la tabla 4.1 calcularemos los cortes. Los valores por parejas son tratados de forma consecutiva, es decir, se analizan el primero y el segundo; a continuación el segundo y el tercero; y así sucesivamente.

La tabla 4.2 muestra cuándo se ha de fijar un corte o no en cada una de las situaciones. Las dos primeras columnas indican qué valores (v_i y v_{i+1}) se están considerando; las dos siguientes ($Clase$) muestran sus clases mayoritarias; la quinta y sexta columna ($Puro$) indican si el valor es puro o impuro; finalmente, las dos últimas precisan si se fija o no un corte entre los valores que se están considerando así como el valor que toma el corte si se establece. En concreto, el cálculo de intervalos iniciales para este ejemplo fija 14 cortes (los 12 cortes intermedios que muestra la tabla junto a los dos extremos 1.0 y 4.4), formando el conjunto de intervalos: {[1.0, 1.3), [1.3, 1.7), [1.7, 1.9), [1.9, 2.3), [2.3, 2.5), [2.5, 3.1), [3.1, 3.3), [3.3, 3.5), [3.5, 3.7), [3.7, 3.9), [3.9, 4.1), [4.1, 4.3), [4.3, 4.4]}.

Del estudio de este ejemplo con todas las posibles situaciones, podemos colegir mediante un mapa de Karnaugh [119] las condiciones para fijar un corte (figura 4.3, línea 6) en el proceso de cálculo de los intervalos iniciales.

4.4.2. Refinamiento de los intervalos

Partiendo de los cortes obtenidos tras el cálculo de los intervalos iniciales, se pretende reducir el número de cortes uniendo intervalos consecutivos sin que se produzca pérdida de la bondad global. Este proceso se lleva a cabo de forma independiente para cada atributo.

La figura 4.2 muestra el algoritmo USD, donde le proceso de refinamiento es realizado entre las líneas 3 y 18. Básicamente, el refinamiento consiste en recorrer los intervalos para cada atributo y evaluar, para cada par de intervalos consecutivos, si es o no posible unirlos dependiendo de la condición de unión expresada entre las líneas 20 y 22. En esta condición, el primer término de la conjunción evita que dos intervalos con diferente clase mayoritaria sean unidos, pues dicha unión no podría ser ventajosa en términos de bondad. El segundo término de la conjunción compara la bondad de la unión con la semisuma de las bondades de los intervalos que intervienen en el par. Esta semisuma representa la bondad media que obtendríamos si no se produce la unión de los dos intervalos. Si un par de intervalos satisface la condición de unión, se marca una posible unión, almacenando también la bondad de dicha unión (línea 6).

Una vez calculadas todas las posibles uniones para el atributo en estudio, se pasa a ejecutar el bucle que desencadena el refinamiento propiamente dicho (línea 9 a 17), uniéndose en cada iteración sólo aquellos intervalos cuya posible unión sea máxima (línea 10). Esta unión produce un nuevo conjunto de intervalos donde se calculan las nuevas posibles uniones y bondades de los dos pares de intervalos en los que interviene el nuevo intervalo resultado de la unión anterior (líneas 11 a 16).

El proceso se ejecuta mientras que el conjunto de posibles uniones no sea vacío. Cuando en una iteración no se ha producido unión, se pasa a tratar el siguiente atributo continuo.

Ejemplo 4.3. *La figura 4.4 representa el proceso completo de refinamiento para el ejemplo 4.2 en forma tabular. Las dos primeras columnas (N, a_k) son similares a las de la tabla 4.1. Las tres columnas siguientes corresponden a los intervalos iniciales ($I.I.$) obtenidos en la tabla 4.2, la clase mayoritaria ($C.M.$) de dichos intervalos y la bondad de éstos ($Bnd.$) aplicando la expresión de la bondad de la ecuación 4.1. Los tres siguientes grupos de columnas*

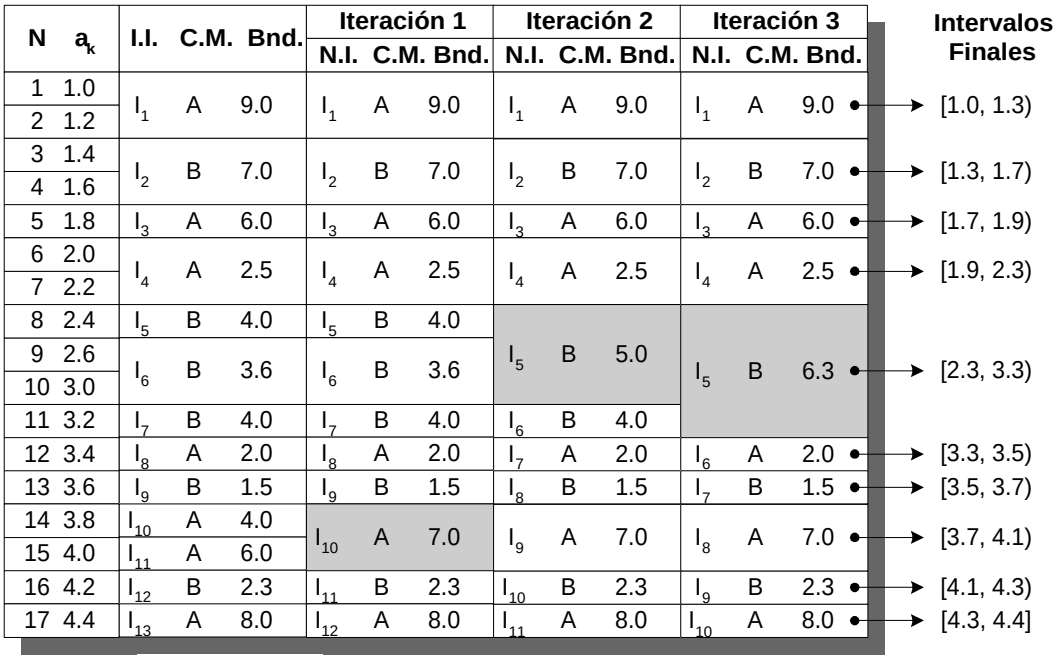


Figura 4.4: Ejemplo de refinamiento de intervalos en USD.

corresponden a las iteraciones que el proceso de refinamiento de intervalos lleva a cabo para este ejemplo, indicando en cada caso los nuevos intervalos ($N.I.$), su clase mayoritaria y su bondad, respectivamente.

Partiendo de los intervalos iniciales se va aplicando el proceso de refinamiento a los intervalos resultantes de cada iteración. La figura 4.4 muestra sombreados los intervalos generados a partir de las uniones. En la figura 4.5 se pueden observar los cálculos realizados en la primera iteración, para los cuales es necesario conocer, además de la clase mayoritaria, las ocurrencias dicha clase (M) y las ocurrencias del resto de clases (m) de cada intervalo, mostradas por la columna $C.M.(M|m)$. En esta primera iteración, de todos los pares de intervalos consecutivos, en cuatro casos coinciden las clases mayoritarias, de los cuales sólo en tres de ellos la bondad de la unión es mayor que la bondad de la media y por tanto son marcados como posibles uniones (✓). De las tres posibles uniones se toma la de máxima bondad, uniéndose los intervalos I_{10} e I_{11} formando el nuevo intervalo I_{10} de bondad igual a 7.0. Sobre el nuevo conjunto de intervalos se aplica nuevamente el proceso de refinamiento, dando como resultado la unión de los intervalos I_5 e I_6 en la iteración 2 (ver figura 4.4). Por último, en la tercera

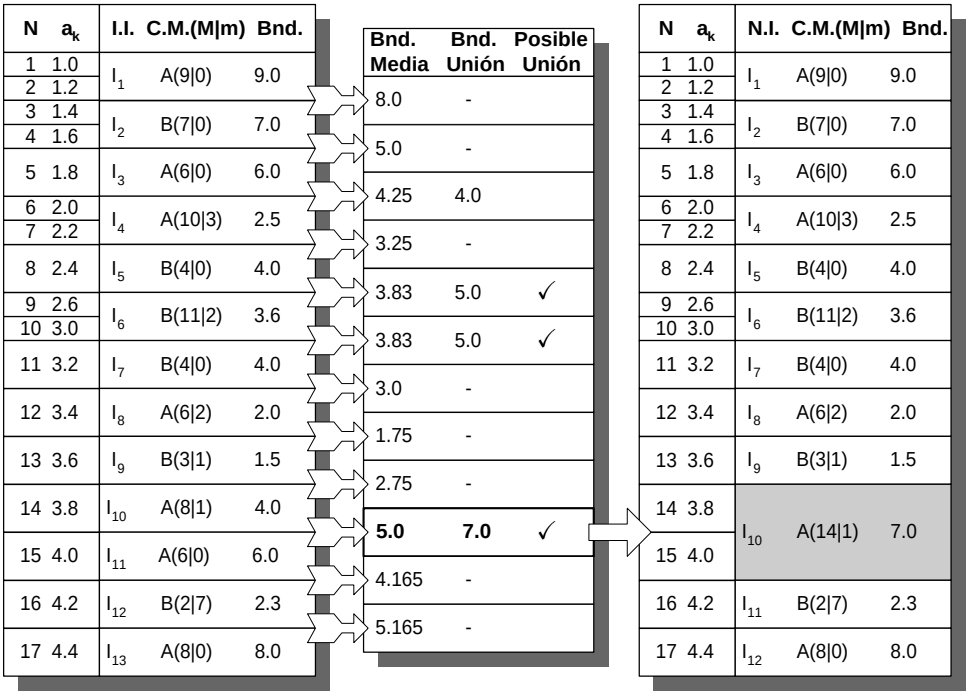


Figura 4.5: Ejemplo de refinamiento de intervalos en USD: 1ª Iteración.

iteración son unidos los intervalos I_5 e I_6 obtenidos tras la iteración 2, dando como conjunto final de intervalos: {[1, 1.3), [1.3, 1.7), [1.7, 1.9), [1.9, 2.3), [2.3, 3.3), [3.3, 3.5), [3.5, 3.7),[3.7, 4.1), [4.1, 4.3), [4.3, 4.4]}.

Justificación del refinamiento

En general, el objetivo de aplicar un algoritmo de discretización a un conjunto de valores continuos es disminuir la cardinalidad dicho conjunto, transformando el dominio inicial teóricamente infinito en un dominio discreto y finito. En el área del aprendizaje supervisado, y más concretamente en el marco de este trabajo, a dicho objetivo se añaden dos más: primero, maximizar la bondad de los intervalos obtenidos, cuya solución trivial es generar un intervalo por cada valor; y segundo, minimizar el número de intervalos sea el menor posible, lo cual es igualmente trivial asignando un único intervalo a todo el rango. Conseguir el equilibrio entre ambos aspectos es una difícil tarea como veremos a continuación.

En principio, si el proceso de inicialización ha generado k cortes, incluyendo los

extremos del rango de valores del atributo, el número de posibles intervalos es la combinación de k elementos tomados de dos en dos, es decir:

$$\binom{k}{2} = \frac{k(k-1)}{2} \tag{4.2}$$

Quizá el número de intervalos posibles pueda no parecer excesivo si el número de cortes es moderado. Sin embargo, el número de posibles combinaciones de intervalos que cubran todo el rango sí lo es. En concreto, el cardinal del conjunto de combinaciones de intervalos generados a partir de k cortes es 2^{k-2}, como se muestra en la demostración 4.1. Este orden exponencial hace que no sea factible realizar una exploración exhaustiva de todas las posibilidades para obtener una solución óptima. Dicha exploración consistiría en analizar, no sólo pares de intervalos, sino las posibles combinaciones de intervalos consecutivos. Sin embargo, aplicando el proceso de refinamiento tratando sólo los pares se logran resultados apropiados sin que suponga un consumo elevado de recursos, ya que el coste computacional es lineal respecto al número de cortes iniciales.

Demostración 4.1 (por inducción). *Debemos demostrar que, dado un conjunto de k cortes, el número de combinaciones $\mathcal{P}(k)$ de los posibles intervalos generados a partir de estos k cortes es*

$$\mathcal{P}(k) = 2^{k-2} \quad (\forall k \geq 2) \tag{4.3}$$

- *Hipótesis inductiva: (véase la figura 4.6)*
  - *Para $k = 1$: $\mathcal{P}(2) = 2^0 = 1$, se cumple.*
  - *Para $k = 2$: $\mathcal{P}(3) = 2^1 = 2$, se cumple.*
  - *Supongamos cierto para $k = n$: $\mathcal{P}(n) = 2^{n-2}$.*
- *Tesis inductiva:*
  - *$k = n + 1$: $\mathcal{P}(n+1) = 2^{n-1}$.*
- *Demostración:*
  1. *Partimos de $k = n$ cortes, los cuales pueden producir conjuntos de un número máximo de $n - 1$ intervalos (primera de las combinaciones de cualquiera de los esquemas de la figura 4.6).*

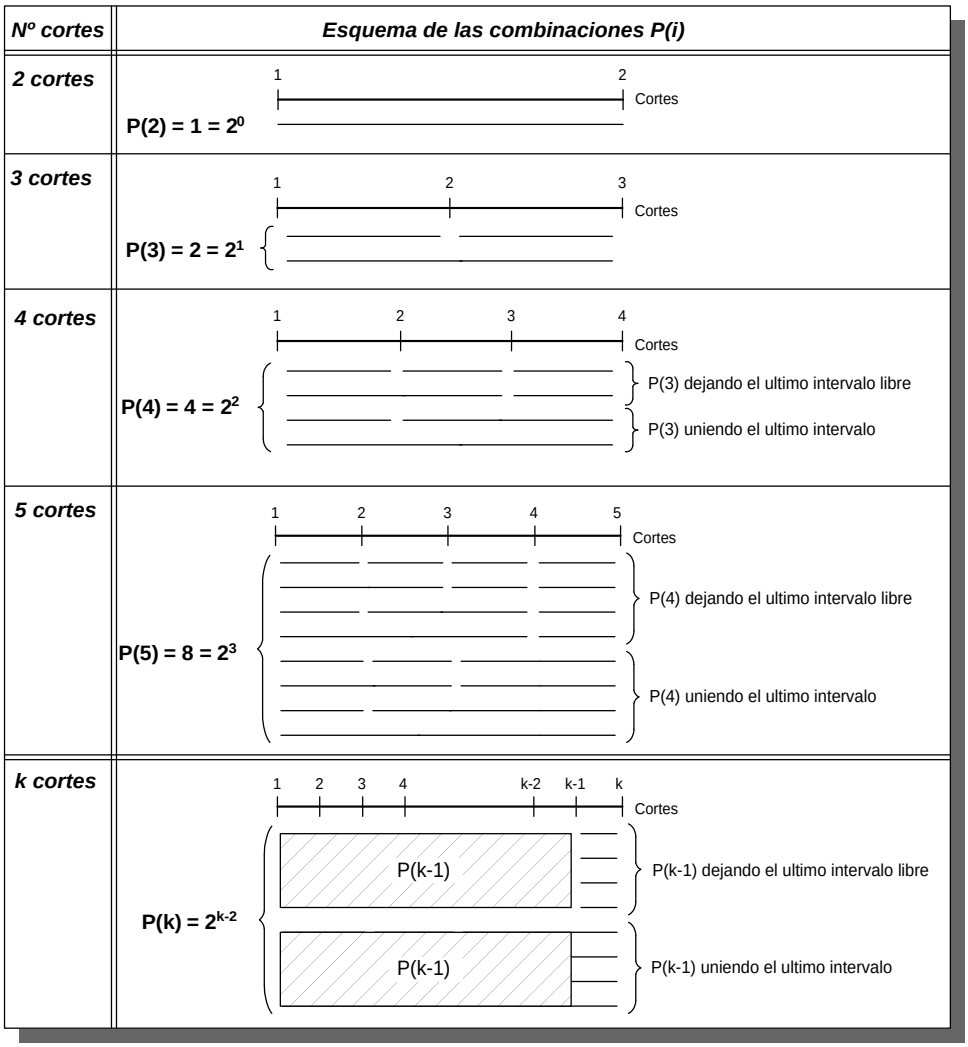


Figura 4.6: Posibles combinaciones de intervalos.

2. *Añadimos un corte más ($k = n+1$) que divide en dos cualquiera de los anteriores $n-1$ intervalos, resultando ahora un máximo de n intervalos. Con independencia de la posición de este nuevo corte, la hipótesis de inducción nos dice que el número de combinaciones debidas a los n primeros cortes es $\mathcal{P}(n) = 2^{n-2}$.*

3. *El último intervalo (entre el corte n y $n+1$) puede o no unirse con el inmediatamente anterior, luego por cada una de las 2^{n-2} combinaciones anteriores tenemos dos posibilidades.*

4. *Luego $\mathcal{P}(n+1) = 2 \times 2^{n-2} = 2^{n-1}$, es cierto $\forall k \geq 2$.* $\square$

4.5. Estudios Experimentales

El problema de la discretización ha sido ampliamente tratado en las últimas décadas. Algunos autores [52, 113] han realizado diversos estudios de clasificación y comparación de los métodos más referenciados en el área de la minería de datos. En este sentido, recientemente, Aguilar et al. [3] han llevado a cabo un estudio experimental entre diversos métodos de discretización, desde el punto de vista de la generación de reglas de decisión mediante algoritmos evolutivos. Los métodos de discretización contemplados son: ID3[4][141], Fayyad & Irani [57], discretización aleatoria, discretización en intervalos de igual anchura e igual frecuencia y USD. Concretamente, el objetivo de este trabajo es determinar qué método de discretización permite a los sistemas basados en algoritmos evolutivos obtener mejores resultados. Para ello, aplican cada uno de los métodos anteriores a un conjunto de bases de datos del almacén de la Universidad de California Irvine (*UCI Repository of Machine Learning Databases*) [24]. Posteriormente, se generan modelos de conocimiento para las bases de datos discretizadas usando tres algoritmos evolutivos de aprendizaje de reglas: ECL [49], GASSIST [18] y nuestra propuesta, HIDER. Aplicando validación cruzada de 10 conjuntos, se obtienen la tasa media de error y el número medio de reglas para cada base de datos, cada discretizador y cada algoritmo de aprendizaje. Con estos resultados, los autores establecen un *ranking* de discretizadores. Observando los resultados, los autores concluyen que USD es el mejor discretizador de los estudiados. La principal razón radica en que USD presenta un comportamiento más estable que los otros métodos. En muchos de los casos fue el mejor, manteniéndose en la media para el resto y no presentando el peor resultado en ninguna prueba.

El estudio comentado anteriormente da una idea de la robustez de USD ante problemas de diferente naturaleza. Sin embargo, no contempla uno de nuestros sistemas de referencia, el 1R.

[4]Se refiere al método de discretización usado por clasificador ID3.

4.5.1. Clasificador USD-C

El objetivo del algoritmo USD no es la simple discretización de los atributos continuos, sino que está orientado a la posterior generación de reglas de decisión por parte de un algoritmo de aprendizaje, en concreto la herramienta HIDER. Este objetivo se traduce en intentar minimizar el error que la discretización introduce más que en reducir al máximo el número de intervalos finales. Por ello, nos planteamos medir experimentalmente como USD reduce el error introducido por la discretización en relación a otros métodos.

Como ya se ha mencionado, entre los algoritmos estudiados, 1R resultaba el método más adecuado para el propósito de HIDER, puesto que, de alguna manera, es capaz de controlar la pureza de los intervalos mediante el parámetro *SMALL*. Este método es, además de un discretizador, un clasificador simple, ya que los intervalos que genera son considerados reglas simples de una única condición en el antecedente (de ahí su nombre, *One–Rule*). Por ello, a pesar de no estar diseñado para este tipo de tareas, nos propusimos construir una herramienta de clasificación simple *ad hoc*, que denominamos USD-*Classifier* (en adelante USD-C), basada en nuestra propuesta, para su posterior comparación con 1R.

Construcción del Clasificador Simple

Dada una base de datos con atributos continuos, se aplica un método de discretización, en nuestro caso USD. Una vez generado el conjunto de cortes $\{c_{i1}, c_{i2}, \ldots, c_{ik_i}\}$ para cada atributo a_i, cada par consecutivo de cortes definirá un intervalo, que en este caso será el antecedente de una regla ($a_i \in [c_{ij}, c_{i(j+1)})$). El consecuente, es decir, la etiqueta de clase ($\mathcal{C}_p$) que la regla asignará a los ejemplos que cubra, será la clase mayoritaria del intervalo. Así, obtenemos un grupo de reglas de clasificación G_i para cada atributo a_i, como muestra la figura 4.7.

Cada grupo G_i es independiente y clasifica todo el conjunto de datos. Nótese que las reglas pertenecientes a un mismo grupo son disjuntas. Esto, unido a que los puntos de corte obtenidos por la discretización no pueden coincidir con ningún valor presente en la base de datos, imposibilita que un mismo ejemplo pueda pertenecer a dos reglas de un mismo grupo, a la vez que asegura que todos los ejemplos son cubiertos.

$$
\begin{array}{rl}
\text{Grupo } G_1: & \\
R_1: & \textit{Si } a_1 \in [c_{i1}, c_{12}) \textit{ Entonces } \mathcal{C}_1 \\
R_2: & \textit{Si } a_1 \in [c_{i2}, c_{13}) \textit{ Entonces } \mathcal{C}_2 \\
\ldots: & \quad \ldots \quad \ldots \quad \ldots \\
R_{k_1-1}: & \textit{Si } a_1 \in [c_{1k_1-1}, c_{1k_1}] \textit{ Entonces } \mathcal{C}_p \\
 & \\
\text{Grupo } G_2: & \\
R_1: & \textit{Si } a_2 \in [c_{21}, c_{22}) \textit{ Entonces } \mathcal{C}_1 \\
R_2: & \textit{Si } a_2 \in [c_{22}, c_{23}) \textit{ Entonces } \mathcal{C}_2 \\
\ldots: & \quad \ldots \quad \ldots \quad \ldots \\
R_{k_2-1}: & \textit{Si } a_2 \in [c_{2k_2-1}, c_{2k_2}] \textit{ Entonces } \mathcal{C}_p \\
 & \\
\vdots & \quad \vdots \quad \vdots \\
 & \\
\text{Grupo } G_m: & \\
R_1: & \textit{Si } a_m \in [c_{m1}, c_{m2}) \textit{ Entonces } \mathcal{C}_1 \\
R_2: & \textit{Si } a_m \in [c_{m2}, c_{m3}) \textit{ Entonces } \mathcal{C}_2 \\
\ldots: & \quad \ldots \quad \ldots \quad \ldots \\
R_{k_m-1}: & \textit{Si } a_m \in [c_{mk_m-1}, c_{mk_m}] \textit{ Entonces } \mathcal{C}_p
\end{array}
$$

Figura 4.7: Sistema de Clasificación Simple.

Clasificación de Ejemplos

Una vez obtenido el clasificador simple nos planteamos cómo clasificar un nuevo ejemplo. Cada grupo G_i de reglas cubre todo el conjunto de datos, por lo que un ejemplo puede ser cubierto hasta por m reglas, donde m es el número de atributos. Por tanto, es necesario definir un criterio de selección que determine qué regla ha de ser empleada para clasificar el ejemplo, pues tales reglas no tienen por qué tener la misma etiqueta de clase.

Se estudiaron diversos criterios de selección, de modo que, dadas las reglas que clasificaban a un ejemplo, podíamos aplicar: *selección aleatoria*, donde simplemente se elegía una de las reglas al azar; *selección por votación*, que consistía en asignar al ejemplo la clase más frecuente en las reglas que clasificaban al mismo; *selección por aciertos*, que optaba por aquella regla que más aciertos presentaba; y *selección por bondad*, que calculaba la bondad de cada regla en función del número de aciertos y errores cometidos, escogiendo la de máxima bondad. Fue este último el criterio, aplicando la misma

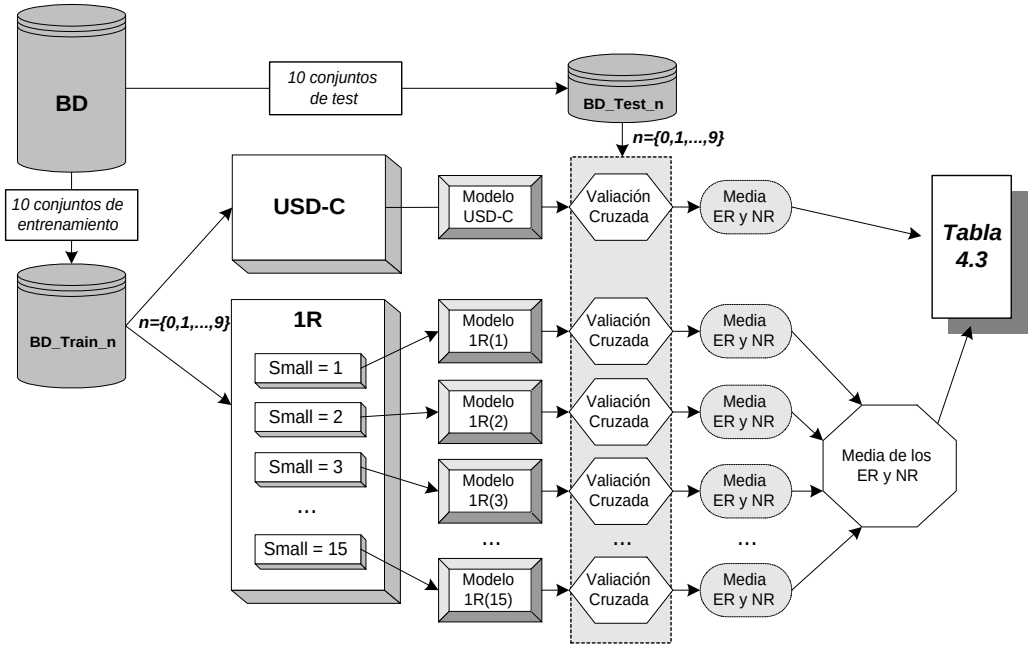


Figura 4.8: Diseño de las pruebas (USD-C vs. 1R).

función de bondad que el propio USD (ecuación 4.1, pág. 87), el que mejores resultados arrojó en las pruebas empíricas y, por tanto, es el empleado en el desarrollo de los experimentos descritos a continuación.

Resultados Empíricos

El propósito de estos experimentos es comprobar la reducción del error que USD-C logra frente a 1R. Es importante señalar que 1R fue ejecutado para 15 valores diferentes del parámetro *SMALL* entre 1 y 15. Así, denotaremos como 1R(s) la ejecución del algoritmo para un valor concreto del parámetro, indicando $s \in \{1, 2, \ldots, 15\}$ dicho valor. Las pruebas fueron realizadas sobre 10 bases de datos de diversa complejidad del almacén UCI, las cuales únicamente presentaban atributos continuos.

La figura 4.8 muestra un esquema del diseño de las pruebas efectuadas. Para cada base de datos, se aplicaron las dos herramientas, USD-C y 1R(s), siendo en total 16 ejecuciones independientes para generar los respectivos modelos de conocimiento. Tanto la estimación del error (*ER*) como el número de reglas (*NR*) de cada ejecución fueron obtenidos mediante validación cruzada de 10 conjuntos, usando los mismos conjuntos de entrenamiento (*BD_Train_n*) y test (*BD_Test_n*) para todas las ejecuciones.

	1R			USD-C		1R/USD-C	
Base de datos	Mejor *SMALL*	ER	NR	ER	NR	ϵ_{er}	ϵ_{nr}
Breast Cancer	15	9.3	18.9	4.2	37.9	2.21	0.50
Bupa	1	39.3	74.4	36.0	155.1	1.09	0.48
Glass	4	44.5	125,1	37.0	547.0	1.21	0.25
Heart	11	34.0	80.7	37.8	181.3	0.90	0.45
Ionosphere	1	8.6	742.9	7.4	2114.4	1.16	0.35
Iris	15	6.7	17.4	4.0	31.0	1.68	0.56
Letter	13	63.3	176.1	62.9	191.0	1.01	0.92
Pima	1	29.3	198.7	28.3	539.2	1.04	0.37
Soybean	7	7.8	37.7	2.5	50.8	3.12	0.74
Vehicle	1	47.2	365.8	46.1	753.0	1.02	0.49
Media						1.44	0.51

Tabla 4.3: Comparativa entre 1R y USD-C.

La tabla 4.3 recoge los resultados obtenidos por cada clasificador. Aunque el objeto de las pruebas se centra en analizar el error introducido por la discretización, la tabla presenta también el número de reglas, que en este caso coincide con el número de intervalos obtenidos. Para USD-C, los resultados son tomados directamente de la validación cruzada. Sin embargo, para 1R tenemos 15 resultados diferentes, por lo que los valores empleados para la comparativa es la media de todas las ejecuciones.

Así, observando la tabla 4.3, la primera columna presenta las bases de datos empleadas; las siguientes tres columnas dan los resultados arrojados por 1R, mostrando la mejor ejecución (*SMALL*), la tasa de error (*ER*) y el número de reglas (*NR*), respectivamente; la quinta y sexta columnas dan los valores *ER* y *NR* producidos por USD-C; las dos últimas columnas reflejan la proporciones entre los resultados respecto a la tasa de error (ϵ_{er}) y el número de reglas (ϵ_{nr}), las cuales son calculadas como el cociente de los valores obtenidos por 1R entre los de USD-C en cada caso; finalmente, la última fila muestra el promedio de ϵ_{er} y ϵ_{nr}.

Como se puede apreciar, para 9 de las 10 bases de datos, USD-C arrojó mejores resultados respecto a la tasa de error, mientras que en ningún caso redujo el número de reglas obtenido por 1R. Este resultado era de esperar, ya que USD-C se centra en minimizar el error. Recordemos en este punto que el objetivo real de USD no es

SMALL	ER	NR
1	39.8	458.7
2	38.9	272.8
3	39.8	184.9
4	38.4	144.5
5	41.2	118.6
6	46.9	102.8
7	45.4	90.8
8	45.4	81.1
9	49.7	74.2
10	46.4	68.6
11	44.4	63.1
12	44.5	59.2
13	48.2	55.4
14	48.2	51.8
15	50.0	49.4
Media	44.5	125.1

Tabla 4.4: Resultados 1R(s) para la base de datos *glass*.

la clasificación sino la discretización y su posterior aplicación en HIDER. Desde este punto de vista, USD-C mejora un 44 % en promedio el error respecto a 1R, empleando aproximadamente el doble de intervalos. Este aumento en el número de intervalos no resulta significativo respecto a la calidad del modelo final que el algoritmo de aprendizaje ha de generar, pues éste se encargará de unir los intervalos convenientemente con un criterio mucho más robusto que el empleado por los discretizadores. Sin embargo, en cuanto al coste computacional de HIDER, el aumento en el número de intervalos influye negativamente en el tiempo de ejecución, aunque, como veremos, este problema es mitigado por la codificación de los individuos. Por otro lado, la reducción del error implica una mejora de la bondad de los intervalos obtenidos por nuestra propuesta respecto a 1R y, por lo tanto, un aumento de la precisión potencial del modelo.

Por otra parte, respecto al parámetro *SMALL*, podemos deducir que el rendimiento de 1R es dependiente de tal parámetro, ya que este presenta una gran diversidad de valores para las distintas bases de datos. La tabla 4.4 muestra el conjunto de resultados de 1R(s) para una de las bases de datos, en concreto *glass*. Como se observa, la tasa de

error varía entre 38.4 % y 50.0 %, lo que demuestra la influencia de *SMALL* sobre los resultados. Este aspecto nos hace pensar que la estimación del valor para el parámetro depende de las características de tamaño, nivel de ruido y distribución de los datos, entre otras, lo que complica en exceso su automatización. Por tanto, la configuración de *SMALL* quedaría en manos del usuario, lo que desde la perspectiva de la herramienta de aprendizaje no resulta práctico.

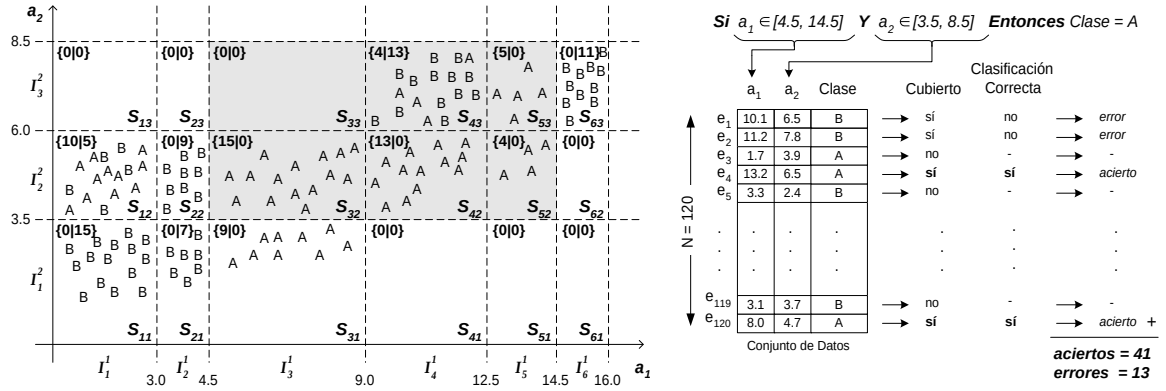
4.6. Aplicación: Editado de Ejemplos USD-E

Los métodos de aprendizaje supervisado que aplican algoritmos evolutivos para generar modelos de conocimiento resultan altamente costosos en tiempo y espacio. Dicho coste es debido fundamentalmente a que el proceso de evaluación de los individuos necesita recorrer completamente el conjunto de datos para medir la bondad de los mismos. Este proceso realiza a menudo operaciones redundantes que pueden ser evitadas. En esta sección describimos otra de las aplicaciones de USD, donde se ha extendido el método de discretización para obtener un algoritmo de editado, que denominamos USD-E [79], el cual reduce la cardinalidad del conjunto de datos con el propósito de disminuir el coste de evaluación los individuos genéticos mediante la extracción, almacenamiento y tratamiento de la información realmente útil para la evaluación.

4.6.1. Motivación

La herramienta HIDER[5] puede operar con dominios continuos gracias a la aplicación previa de algún algoritmo de discretización, en nuestro caso USD. Así, las reglas sólo pueden establecer condiciones usando un determinado conjunto finito de intervalos, a los cuales llamamos *intervalos atómicos*, dado que una vez discretizado un atributo, éstos no podrán ser divididos. En cualquier caso, la discretización de una base de datos convierte el espacio de atributos inicialmente continuo y teóricamente infinito en un espacio discreto y finito de soluciones.

[5]Aunque esta sección se centra en la herramienta HIDER, nuestra propuesta de editado puede ser aplicada a cualquier sistema de aprendizaje evolutivo de reglas de similares características.



(a) División del espacio por USD (b) Evaluación lineal

Figura 4.9: Motivación USD-E.

Ejemplo 4.4. *La figura 4.9(a) muestra un ejemplo para una base de datos con 120 ejemplos, dos clases (A y B), y dos atributos continuos, los cuales han sido discretizados mediante* USD, *obteniéndose 6 intervalos (I_i^1) para a_1 y 3 intervalos (I_j^2) para a_2, respectivamente. Cada par (I_i^1, I_j^2) determina un subespacio ($\mathcal{S}_{ij}$), resultando un total de 18 regiones. Los números entre llaves $\{\varepsilon_A|\varepsilon_B\}$ representan los ejemplos de cada clase incluidos en la región correspondiente.*

Tras la discretización, las regiones derivadas de ésta podrán ser unidas en una regla para formar el antecedente de la misma, pero nunca divididas, ya que sus fronteras de decisión vienen dadas por intervalos atómicos. Por extensión, denominamos a estas regiones *subespacios atómicos*. Para simplificar la explicación, asumiremos que las reglas son conexas, es decir, sus condiciones contendrán siempre un único intervalo, sin importar su tamaño, pudiendo éste contener varios intervalos atómicos adyacentes. Esto no supone una limitación, ya que una regla con condiciones discontinuas en el espacio puede trasformarse directamente en un conjunto de reglas conexas.

Para contabilizar los aciertos y errores de una regla, métodos como GASSIST[17], COGITO[1] o el propio HIDER en sus primeras versiones, realizan un recorrido lineal de los datos de entrenamiento, ejemplo a ejemplo, comprobando si el individuo los clasifica, en cuyo caso verifica si tal clasificación es correcta o no. Esta comprobación implica analizar si los atributos de los ejemplos cumplen las condiciones de la regla. Por tanto, podemos colegir que el coste de evaluación individual[6] es de orden $\Theta(Nm)$,

[6]Coste de evaluación de un único individuo.

siendo N el numero de ejemplos y m el número de atributos del conjunto de datos. Este recorrido se repite G generaciones para los P individuos de la población, por lo que el coste total de evaluación es de orden $\Theta(GPNm)$.

Ejemplo 4.5. *La siguiente regla equivale a la zona sombreada en la figura 4.9(a), y determina que si un ejemplo se encuentra en el subespacio* $\{I_3^1 \cup I_4^1 \cup I_5^1\} \cap \{I_2^2 \cup I_3^2\}$*, éste será de clase A.*

Si a_1 ∈[4.5, 14.5] **y** a_2 ∈[3.5,8.5] ⇒ Clase=A

Para contabilizar los aciertos y los errores de esta regla es necesario tomar cada ejemplo del conjunto de datos y comprobar si es cubierto por la misma, en cuyo caso se coteja la clase, tal como ilustra la figura 4.9(b). En este caso, la regla presenta 41 aciertos y 13 errores. Este cálculo se repite para cada individuo de la población.

El ejemplo anterior deja en evidencia la existencia de una carga computacional innecesaria en el proceso de evaluación individual, debida principalmente a dos aspectos:

1. El conteo redundante de ejemplos para aquellas zonas del espacio compartidas por varias reglas.

2. La exploración de todo el espacio para evaluar reglas que sólo cubren parte del mismo.

Una manera de reducir el coste de evaluación es disminuyendo el número de ejemplos del conjunto de datos $\mathcal{D}$, obteniendo un subconjunto $\mathcal{D}^* \subseteq \mathcal{D}$ que contenga el mismo conocimiento que $\mathcal{D}$, pero con un número de ejemplos N^* menor. Los métodos de reducción del número de ejemplos se engloban dentro de las denominadas técnicas de editado [179].

De este planteamiento nace el algoritmo de editado incremental USD-E, basado en la discretización USD, el cual permite, además de reducir el número de ejemplos, llevar a cabo una exploración más eficiente del conjunto de datos.

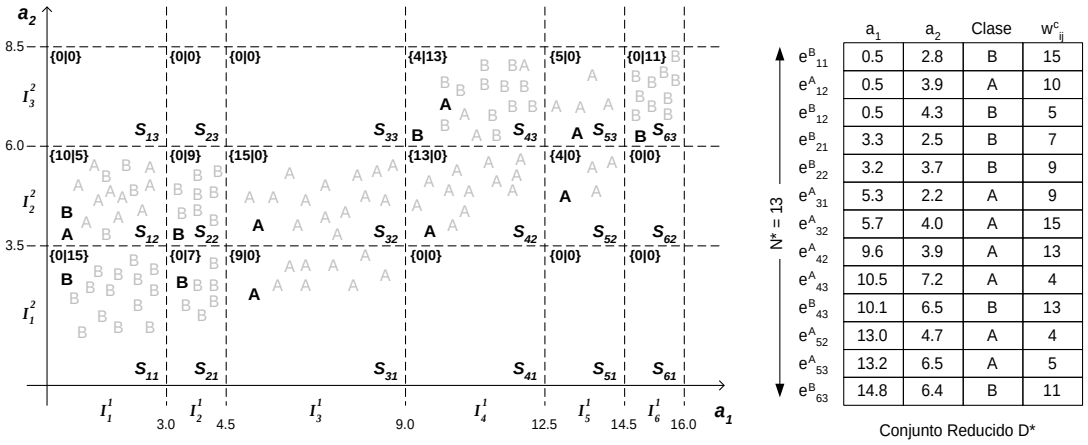
4.6.2. Reducción del número de ejemplos mediante USD-E

Teniendo en cuenta lo expuesto en la sección anterior, podemos deducir que, aunque los ejemplos incluidos en un mismo subespacio atómico pueden ser sintácticamente diferentes, desde el punto de vista de la evaluación de un individuo son semánticamente idénticos. Ello hace posible contabilizar cuántos ejemplos de cada clase coexisten en cada subespacio atómico y almacenar estos valores para su ulterior utilización en la evaluación de los individuos.

Partimos de un conjunto de datos $\mathcal{D}$ con N ejemplos, donde los atributos continuos ya han sido convenientemente discretizados, generando el consiguiente conjunto de subespacios atómicos. Cada ejemplo $e = (a_1, a_2, \ldots, a_m|c)$ está formado por un conjunto de atributos y una clase (*v.g*, $e_1 = (10.1, 6.5 \mid B)$ en la figura 4.9(b)). Por cada subespacio atómico $\mathcal{S}_{ij}$, se contabilizan las instancias de cada clase $\{\varepsilon_A|\varepsilon_B\}$ y se eligen tantos ejemplos representantes (e_{ij}^c) como clases distintas convivan en el subespacio representado, añadiendo éstos al conjunto de datos reducido $\mathcal{D}^*$. Aunque estos representantes no tienen por qué coincidir con algún ejemplo original, por simplicidad, elegimos el primero que se encuentre en el conjunto de datos. Cada ejemplo representante tendrá la misma estructura que un ejemplo original, sólo que añadimos un peso ω_{ij}^c igual al ε_c que contabiliza las instancias de la clase c en el subespacio $\mathcal{S}_{ij}$. Aquellas regiones cuyo ε_c sea 0, no tendrán representante en $\mathcal{D}^*$ para la clase c. Por ejemplo, en la figura 4.9(a), $e_{43}^A = (10.5, 7.2 \mid A, 4)$ y $e_{43}^A = (10.1, 6.5 \mid B, 13)$ representan al subespacio $\mathcal{S}_{43}$, mientras que $e_{53}^A = (13.2, 6.5 \mid A, 5)$ es el único representante para el subespacio $\mathcal{S}_{53}$. Para las regiones vacías, como $\mathcal{S}_{13}$, $\mathcal{D}^*$ no contiene ejemplos.

La figura 4.10 muestra el resultado de aplicar esta técnica de editado al conjunto de datos de la figura 4.9. A la izquierda, se resaltan los ejemplos representantes de cada subespacio atómico, mientras que los eliminados aparecen en color más claro. A la derecha, se representa el conjunto de datos reducido sobre el cual se evaluarán los individuos de la población. Como podemos observar, el conjunto de datos original de $N = 120$ ejemplos ha sido transformado en el conjunto $\mathcal{D}^*$ con tan sólo $N^* = 13$ ejemplos, cada uno con su peso correspondiente.

Una vez concluido el proceso de editado, la evaluación de un individuo puede ser realizada de forma lineal, como ilustraba la figura 4.9(b), sólo que ahora el número



	a_1	a_2	Clase	w^c_{ij}
e^B_{11}	0.5	2.8	B	15
e^A_{12}	0.5	3.9	A	10
e^B_{12}	0.5	4.3	B	5
e^B_{21}	3.3	2.5	B	7
e^B_{22}	3.2	3.7	B	9
e^A_{31}	5.3	2.2	A	9
e^A_{32}	5.7	4.0	A	15
e^A_{42}	9.6	3.9	A	13
e^A_{43}	10.5	7.2	A	4
e^B_{43}	10.1	6.5	B	13
e^A_{52}	13.0	4.7	A	4
e^A_{53}	13.2	6.5	A	5
e^B_{63}	14.8	6.4	B	11

Figura 4.10: Editado mediante USD-E.

de ejemplos es mucho menor. Esta solución sólo resuelve el primero de los problemas comentados en el apartado 4.6.1, ya que sigue siendo necesario un recorrido lineal del conjunto $\mathcal{D}^*$.

Una posible mejora del proceso de evaluación para abordar el problema de la exploración exhaustiva del espacio consiste en aprovechar que el conjunto de datos $\mathcal{D}^*$ se encuentra ordenado por regiones (véanse los subíndices de cada e^c_{ij} en la figura 4.10). Dado el conjunto ordenado de intervalos de cada atributo, es relativamente sencillo obtener los subespacios $\mathcal{S}_{ij}$ cubiertos por una regla, y más concretamente los subíndices i y j, por lo que podríamos aplicar una búsqueda binaria de los ejemplos e^c_{ij}, evitando así el recorrido lineal. En este caso no sería necesario guardar los valores de cada atributo, sino sólo la clase, el peso y los índices i y j.

Nótese que la generalización del método para k clases y m atributos es trivial. Simplemente tendríamos un conjunto de contadores de clase ($\{\varepsilon_{c_1}|\varepsilon_{c_2}|\ldots|\varepsilon_{c_k}\}$) y un conjunto de índices para denotar un subespacio ($\mathcal{S}_{i_1\ldots i_m}$) o ejemplo ($e^c_{i_1\ldots i_m}$).

Generalización para atributos discretos

Es evidente que no podemos limitar la aplicación de USD-E a bases de datos exclusivamente continuas. Por ello, aunque nuestra propuesta sea una extensión del método de discretización USD, el proceso de editado es fácilmente aplicable a atributos de dominio discreto, salvando algunos matices importantes.

Los atributos continuos, aunque sean discretizados, suelen definir un espacio mucho más complejo que los discretos, fundamentalmente debido a dos razones: el número de intervalos suele ser mucho mayor en aplicaciones reales, lo que multiplica el número de subespacios; y, por otra parte, las regiones suelen incluir más de un ejemplo, es decir, desde el punto de vista del aprendizaje, existen ejemplos con el mismo significado. Por ello, nuestra propuesta tiene a priori mayor justificación cuando el conjunto de datos incluye atributos continuos.

En conjuntos de datos exclusivamente discretos, cuando no hay ejemplos repetidos, cada subespacio atómico contiene, a lo sumo, un único ejemplo, es decir, el editado no produciría reducción alguna. Sin embargo, aunque la multiplicidad de ejemplos pueda parecer poco habitual, es relativamente común. Por ejemplo, la aplicación previa de algún método de selección de atributos puede provocar la aparición de ejemplos idénticos si fueron eliminados aquellos atributos que los diferenciaban.

Por tanto, el método de editado es favorable siempre que el conjunto de datos contenga ejemplos similares desde el punto de vista del aprendizaje, soslayándose su aplicación en caso contrario. De cualquier modo, nuestra propuesta no supone un incremento significativo de la carga computacional respecto al coste de un algoritmo evolutivo, por lo que se aconseja su uso si no se dispone de información a priori sobre la multiplicidad de los datos.

4.6.3. Pruebas

Para comprobar empíricamente la reducción del coste computacional del proceso de evaluación, se realizaron pruebas sobre varias bases de datos del almacén UCI. El algoritmo evolutivo de aprendizaje utilizado fue HIDER[7]. Tal algoritmo precisó de algunas modificaciones, aunque mínimas, para adaptar el proceso de evaluación de individuos a los ejemplos con pesos. Para comprobar que el editado no afecta al modelo de conocimiento obtenido, las pruebas fueron realizadas mediante validación cruzada con 10 conjuntos para cada base de datos.

[7]La versión de HIDER utilizada en estas pruebas no incluye la estructura EES para evitar la influencia de ésta en el coste de evaluación.

Características					Editado	Coste Relativo		
Base de Datos	N	m	Tipo	#Clases	N^*	TEjec	TEval	Espacio
Breast Cancer W.	699	9	C	2	263	0.44	0.44	0.38
Hayes Roth	132	4	C	3	28	0.33	0.23	0.21
Iris	150	4	C	3	70	0.59	0.57	0.47
Led7	3200	7	D	10	336	0.31	0.13	0.11
Vote	435	16	D	2	342	0.82	0.74	0.79
Zoo	101	16	D	7	59	0.49	0.38	0.58
Media						0.49	0.41	0.42

Tabla 4.5: Resultados de USD-E.

En total, se llevaron a cabo experimentos sobre 14 bases de datos: *Breast Cancer*, *Bupa*, *Cleveland*, *Glass*, *Hayes Roth*, *Heart*, *Hepatitis*, *Horse Colic*, *Iris*, *Led7*, *Pima Diabetes*, *Tic Tac Toe*, *Vote* y *Zoo*. HIDER fue ejecutado para cada una de ellas, usando el conjunto de datos original ($\mathcal{D}$) y el reducido ($\mathcal{D}^*$) para comparar el coste en tiempo y espacio. Así, se produjo una reducción del número de ejemplos en 8 casos, manteniéndose éste en los 6 restantes. Lógicamente, para estos últimos, el coste de ejecución sufrió un leve incremento usando $\mathcal{D}^*$ debido al preprocesado, aunque éste jamás superó el 5 % respecto al tiempo empleado usando $\mathcal{D}$.

De las 8 bases de datos donde el editado tuvo un efecto positivo, la reducción del conjunto de datos fue superior al 20 % en 6 casos, siendo éstas las presentadas en la tabla 4.5. Las cinco primeras columnas muestran las características de cada base de datos: nombre, número de ejemplos, número de atributos, tipo de los atributos (continuos o discretos) y numero de clases, respectivamente. La siguiente columna da el número de ejemplos tras el proceso de editado. Finalmente, las tres últimas muestran el coste relativo respecto al tiempo de ejecución, el tiempo de evaluación y el espacio de almacenamiento empleados. Estos valores se obtienen dividiendo, en cada caso, el coste empleando $\mathcal{D}^*$ entre el coste usando $\mathcal{D}$. La última fila presenta cada uno de los costes relativos anteriores en promedio.

Observando los resultados en media, el tamaño del conjunto de datos es reducido al 42 %. Ello produce un decremento del tiempo de evaluación hasta el 41 %. Para las bases de datos de la tabla 4.5, podemos afirmar que el editado acelera el proceso de

aprendizaje, usando menos de la mitad de los recursos computacionales, en promedio.

En resumen, concluimos que USD-E produce un descenso del coste computacional, en tiempo y espacio, asociado a la evaluación de individuos durante el aprendizaje, en aproximadamente la mitad de las bases de datos analizadas. Tal disminución es proporcional a la reducción del número de ejemplos resultante del editado, sin que ello repercuta en la calidad del modelo de conocimiento generado. Además, la aplicación de USD-E no supone un aumento significativo del tiempo de ejecución del proceso de aprendizaje completo si dicha reducción no se produce.

4.7. Conclusiones

Como resultado de esta parte de la investigación, hemos obtenido un algoritmo de discretización supervisado no paramétrico que disminuye la cardinalidad de los atributos continuos de una base de datos etiquetada. USD está especialmente enfocado hacia la posterior generación de reglas de decisión por parte de la herramienta HIDER, aunque es aplicable como preprocesado a cualquier algoritmo de aprendizaje supervisado que requiera discretización previa.

USD cumple en gran medida los diferentes objetivos de eficiencia y eficacia marcados al principio. Divide el rango de cada atributo continuo en intervalos disjuntos, intentando que estos intervalos conserven la máxima bondad posible para un número de intervalos aproximadamente del mismo orden que otros métodos de similar propósito. El método es totalmente determinista y no precisa ningún parámetro más que el conjunto de datos, de modo que la calidad de los intervalos no depende de configuración alguna por parte del usuario. Por otra parte, el método tiene un coste computacional reducido, de orden subcuadrático respecto al número de ejemplos, despreciable frente al coste del algoritmo evolutivo.

Las pruebas empíricas realizadas arrojan que USD es un método robusto, ofreciendo un buen rendimiento frente a otros métodos para las bases de datos en ellas tratadas.

Capítulo 5

HIDER

La naturaleza está repleta de razonamientos
que no tuvo nunca la experiencia.
LEONARDO DA VINCI.

5.1. Introducción

Desde el punto de vista de la eficacia, el objetivo de un sistema de aprendizaje diseñado para tareas de clasificación es obtener un modelo capaz de clasificar los ejemplos de un conjunto de datos con el menor error posible, minimizando a su vez la complejidad de la estructura de conocimiento generada. Respecto al método a utilizar para la obtención del modelo de conocimiento, estudios previos a este trabajo han demostrado el buen funcionamiento de la computación evolutiva frente a problemas de aprendizaje automático, y más concretamente en tareas clasificación [45, 66, 98, 172]. En tal caso, cuando un problema de aprendizaje es abordado aplicando técnicas evolutivas, es denominado *aprendizaje evolutivo*. Uno de los mayores inconvenientes de las técnicas basadas en búsquedas probabilísticas es el elevado coste computacional que implica la repetitiva evaluación de las soluciones candidatas (*eficiencia*). Además, estas técnicas operan habitualmente sobre espacios de búsqueda muy grandes, sobre todo cuando el dominio de dicho espacio es continuo, lo cual dificulta la obtención de buenas soluciones (*eficacia*).

En este contexto, el principal objetivo de esta investigación es la mejora en eficiencia y eficacia de las técnicas de aprendizaje evolutivo. Como resultado de nuestro estudio

se ha desarrollado la herramienta denominada HIDER (*Hierarchical Decision Rules*) [4, 77], la cual aplica un algoritmo evolutivo para generar un conjunto jerárquico de reglas de decisión en el marco del aprendizaje supervisado.

El punto de partida de este trabajo fue una familia de algoritmos evolutivos de generación de reglas denominada COGITO [1], descrita en la sección 3.3, cuya versión más reciente[1] es también la más versátil y la que referenciaremos en lo sucesivo. El modelo de conocimiento obtenido por COGITO alcanza una excelente precisión. Sin embargo, la cantidad de recursos que el algoritmo require lo hace computacionalmente muy costoso. Por ello, HIDER hereda la representación del conocimiento usada en COGITO, es decir reglas jerárquicas de decisión, centrando nuestro esfuerzo en acelerar la obtención de las mismas así como en aumentar la calidad del modelo tanto en precisión como en complejidad.

En general, la aplicación de algoritmos evolutivos presenta dos factores críticos: la *codificación* de los individuos de la población genética y la *evaluación* de éstos. Ambos factores, entre otros, influyen en la eficacia y en la eficiencia del algoritmo, siendo por tanto los aspectos donde centramos la mayor parte de nuestro esfuerzo.

En primer lugar, la elección de una codificación adecuada puede reducir considerablemente el espacio de búsqueda, acelerando la convergencia del algoritmo a la vez que puede aumentar la probabilidad de encontrar buenas soluciones. En concreto, COGITO utiliza una codificación híbrida (véase la sección 3.3.1), cuya componente real hace que el espacio de búsqueda sea teóricamente infinito para dominios continuos. Esto nos motivó a diseñar una codificación que minimizara, o al menos disminuyera, el cardinal del conjunto de posibles soluciones sin que ello produjera pérdida en la precisión en las mismas, dando como resultado la denominada *Codificación Natural* descrita más adelante. Parte del éxito obtenido con esta codificación es debido a la aplicación previa del algoritmo de discretización USD, descrito en el Capítulo 4.

Respecto a la evaluación de los individuos, hay que tener en cuenta no sólo qué función de evaluación se debe utilizar, sino también el proceso seguido para aplicar la misma. En este sentido, COGITO usa una función de evaluación que asigna un valor de bondad a un individuo según el número de aciertos y errores que éste cometa sobre

[1] Esta versión de COGITO utiliza la codificación híbrida, la cual posibilita el tratamiento de bases de datos con atributos continuos y discretos.

los datos de entrenamiento. Para ello, explora secuencialmente el conjunto de datos, tomando cada ejemplo y comprobando la correcta clasificación de éstos. Como adelantamos en la sección 4.6.1 del capítulo anterior, este método de evaluación resulta altamente costoso en términos de tiempo y espacio. Este problema de eficiencia nos impulsó a desarrollar una estructura de indexación de los datos que redujera el coste de evaluación. En este sentido, HIDER incorpora la denominada *Estructura de Evaluación Eficiente* o EES (*Efficient Evaluation Structure*). Dicha estructura indexa el conjunto de datos de forma que se aprovecha la semántica del individuo (regla de decisión codificada) que en ese momento se esté evaluando para discriminar aquellos ejemplos que no son clasificados o cubiertos por el mismo. De este modo, sólo son contabilizados los ejemplos estrictamente necesarios, lo que influye directamente en la reducción del tiempo de ejecución.

5.2. Representación del conocimiento

El propósito general de HIDER es extraer el conocimiento inherente de un conjunto de datos etiquetados[2] y generar una estructura que modele tal conocimiento con el fin de utilizarla para clasificar ejemplos desconocidos[3] y colegir propiedades de las diferentes clases. Comúnmente, dicha estructura se representa en forma de *árboles de decisión* o *reglas de decisión* (véase sección 2.3). En general, la representación del conocimiento puede ser evaluada según dos criterios: su precisión en la clasificación y su complejidad. Ambos aspectos tienen gran relevancia, ya que de nada sirven estructuras que clasifiquen con una elevada tasa de error, del mismo modo que una extremada complejidad impediría la comprensión de las mismas. En este sentido, los árboles de decisión presentan el problema de que tienden a crecer en aplicaciones reales, lo cual ha llevado a algunos autores a trasformar dichos árboles en reglas [142]. Por ello, nos decantamos por las reglas de decisión como la representación que HIDER debía utilizar y, en concreto, reglas jerárquicas donde existe un orden predeterminado de evaluación de las mismas.

[2]El conjunto de datos contiene un atributo de decisión o clase. Así, cada ejemplo tiene un valor denominado *etiqueta* que indica a qué clase pertenece.

[3]Ejemplos no etiquetados.

Respecto a los tipos de reglas de decisión, COGITO adoptó tres representaciones geométricas diferentes [5]: hiperrectángulos paralelos a los ejes, hiperrectángulos oblicuos e hiperelipses. Dado que ninguna de estas tres opciones proporciona una mejora global de las reglas respecto a las otras y, por otro lado, las dos últimas complicaban el diseño de la codificación además de resultar computacionalmente más costosas, nos inclinamos por aquella que ofrecía la representación más sencilla e intuitiva, es decir, las reglas hiperrectangulares paralelas a los ejes.

5.2.1. Árboles de decisión vs. reglas jerárquicas

Una de las herramientas de clasificación más usadas es C4.5 [145], la cual es referenciada ampliamente en la bibliografía junto con sus variantes [141, 142, 147, 148]. Se trata de un algoritmo recursivo que divide el espacio definido por los diferentes atributos de la base de datos estableciendo cortes en los valores que mayor ganancia de información proporcionan. En cada paso, se establece un corte que divide el espacio en dos partes que son procesadas por separado de forma recursiva. Este modo de buscar las regiones presenta algunos inconvenientes, los cuales se resumen en que muchas reglas de la misma clase contiguas en el espacio no pueden ser unidas en una única regla. La Figura 5.1 ilustra este aspecto, donde se muestran las regiones que C4.5 y HIDER establecen para un conjunto de datos con dos atributos (AT_1 y AT_2) y dos clases ($\bullet$ y $\times$). Cada punto representa a un ejemplo etiquetado con una determinada clase. Como se puede observar, el árbol que genera C4.5 tiene cinco hojas, que equivalen a cinco reglas distintas. Este árbol se genera a partir de los cuatro cortes que el método establece de forma secuencial (numerados del 1 al 4). Es decir, C4.5 genera cuatro reglas para clasificar los ejemplos de tipo $\times$ y una para los de tipo $\bullet$. Por el contrario, HIDER genera sólo dos reglas para realizar la clasificación. La regla $R1$ clasifica todos los ejemplos de clase $\bullet$, mientras que $R2$ hace lo propio para los de clase $\times$. Esta reducción en la complejidad de la representación del modelo es debida a la naturaleza jerárquica del conjunto de reglas, ya que permite que las regiones definidas por las reglas más profundas en la jerarquía incluyan a aquellas situadas en los niveles superiores.

La ventaja de las reglas jerárquicas de decisión se hace más patente cuanto mayor es el número de clases y cuanto más difusa es la distribución de los ejemplos en esas

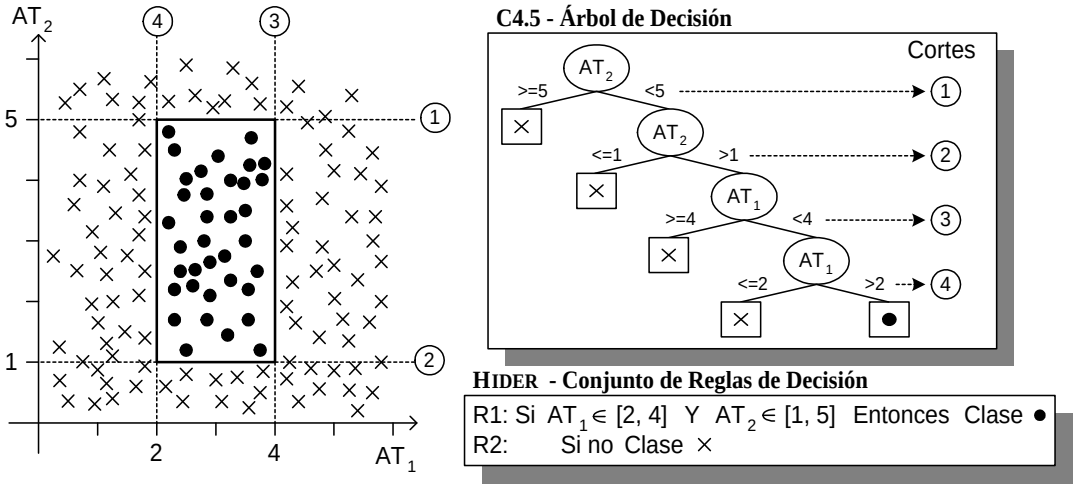
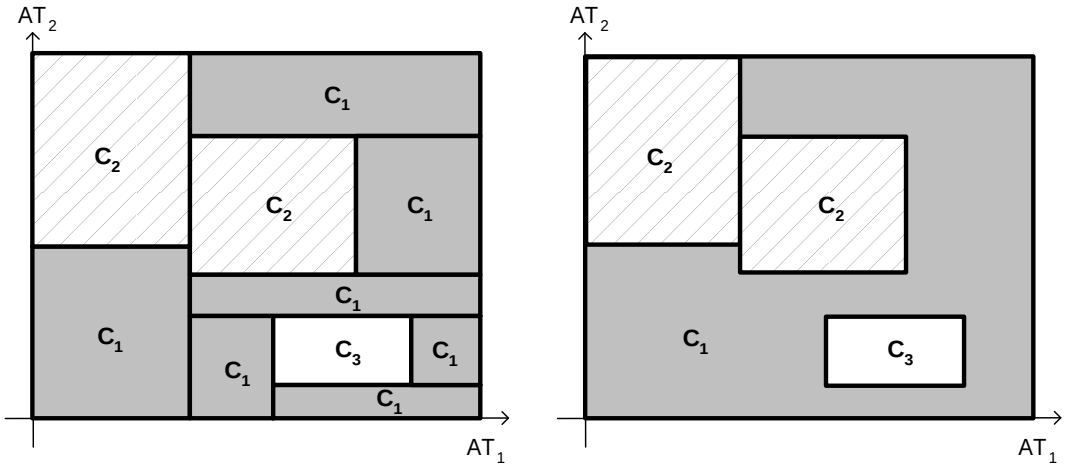


Figura 5.1: Árbol de Decisión vs. Reglas de Decisión.

clases. La figura 5.2 presenta otro ejemplo que compara la división del espacio que las reglas generadas por ambas herramientas produce para una bases de datos de dos atributos y tres etiquetas de clase (C_1, C_2 y C_3). En este caso, ambos algoritmos obtienen dos reglas para C_2 y una para C_3. Sin embargo, para la clase C_1, C4.5 genera siete regiones frente a la única regla generada por HIDER.



(a) C4.5 (b) HIDER

Figura 5.2: División del espacio: C4.5 vs. HIDER.

En resumen, la reducción del cardinal del conjunto de reglas, sin que se produzca pérdida de precisión en la clasificación, así como la obtención de regiones incluidas en

otras, fueron las principales metas que nos propusimos al inicio de esta investigación, y que han motivado el desarrollo de HIDER.

5.3. Codificación Natural

Como se ha mencionado con anterioridad, la codificación es uno de los aspectos críticos en la aplicación de algoritmos evolutivos. La elección de una codificación adecuada puede reducir sensiblemente el tamaño del espacio de búsqueda, disminuyendo así el numero de posibles soluciones y, por tanto, acelerando la convergencia del algoritmo.

La codificación híbrida mezcla genes binarios y reales dependiendo si el dominio de valores que representan es discreto o continuo respectivamente. El tamaño del espacio de búsqueda viene determinado por el número de genes de los individuos, comúnmente llamado longitud, así como por el cardinal del alfabeto de símbolos de esos genes. En general, para individuos homogéneos de longitud L donde todos los genes tiene el mismo alfabeto Ω, el tamaño del espacio de búsqueda es $|\Omega|^L$. Por ejemplo, si usamos sólo codificación binaria, cuyo alfabeto es $\Omega_B = \{ 0, 1\}$, el espacio de búsqueda tendrá un tamaño de 2^L. Sin embargo, cuando los individuos contiene genes con distinto alfabeto, el tamaño efectivo del espacio de búsqueda viene dado por la ecuación 5.1, donde $|\Omega_i|$ es el cardinal del alfabeto para el i-ésimo gen.

$$S = \prod_{i=1}^{L} |\Omega_i| \tag{5.1}$$

Si se utiliza codificación real pura [56, 149, 165], el cardinal del alfabeto de símbolos es teóricamente infinito, lo cual hace que el espacio de búsqueda también lo sea. Ello afecta directamente a la eficiencia y eficacia del algoritmo. Por un lado hace que coexistan gran cantidad de valores diferentes cuyo significado es idéntico desde el punto de vista del problema, lo que repercute en la diversidad de individuos en la población debido principalmente a que la aplicación de los operadores genéticos pueden producir nuevos individuos sintácticamente distintos a los padres pero semánticamente similares a estos. Por otra parte, la gran cantidad de soluciones que conviven en el mismo espacio dificulta el encontrar aquellas de mayor calidad. Ambos aspectos ralentizan

la evolución del modelo y, por tanto, no se asegura la obtención de la solución en un tiempo finito.

Para poder abordar el problema en un entorno que ofrezca garantías de solución, la codificación "ideal" debe cumplir el siguiente conjunto de propiedades:

1. **Completitud**: todo fenotipo ha de poder codificarse correctamente.
2. **Consistencia**: todo genotipo ha de representar un fenotipo válido.
3. **Coherencia**: ningún fenotipo diferente podrá ser codificado.
4. **Uniformidad**: todo fenotipo estará representado por la misma cantidad de genotipos.
5. **Unicidad**: todo fenotipo debe estar representado por un único genotipo.
6. **Simplicidad**: la función de codificación debe ser fácil de aplicar en ambos sentidos.
7. **Localidad**: pequeñas modificaciones en el genotipo se corresponderán con pequeñas modificaciones en el fenotipo.
8. **Minimalidad**: la longitud de la codificación ha de ser la menor posible.

En vista de los problemas de la codificación híbrida, derivados fundamentalmente de la componente real, y con el propósito de mejorar la codificación de COGITO, nos planteamos un nueva codificación más compacta y en la que no hubiera que realizar constantes conversiones para la aplicación de los operadores genéticos. Principalmente centramos nuestros esfuerzos en dos de las propiedades de la codificación "ideal": *unicidad* (todos los elementos están representados por una sola codificación) y *minimalidad* (la longitud de la codificación debe ser la menor posible). Con este propósito desarrollamos la denominada *Codificación Natural*, donde cada gen representa una condición sobre los valores de un atributo, ya sea continuo o discreto. Sin embargo, al contrario que otras codificaciones, en la natural cada gen es un único número natural.

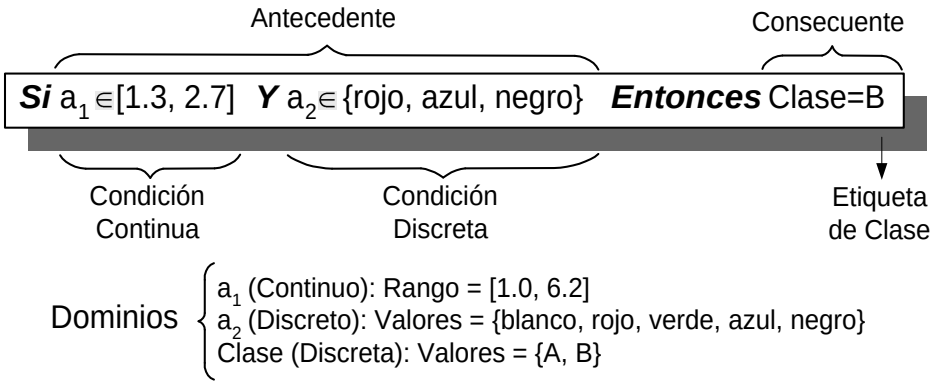


Figura 5.3: Regla de Decisión.

5.3.1. Individuo Natural

Cada individuo de la población representa una única regla de decisión que describe la relación entre los valores de los atributos y las etiquetas de clase. El *antecedente* o *descripción de la regla* (parte izquierda) es una conjunción de condiciones que restringe los valores que los atributos de un ejemplo pueden tomar para que éste sea clasificado con la etiqueta de clase expresada en el *consecuente* (parte derecha). La forma de las condiciones depende del tipo de dominio del atributo sobre el que actúa. Cuando un atributo (a_i) es discreto, la condición toma la forma $a_i \in \{v_1, v_2, \ldots, v_k\}$, donde los valores $\{v_1, v_2, \ldots, v_k\}$ no son necesariamente todos aquellos que el atributo puede tomar. Así, la condición se evaluará como cierta cuando el a_i tome cualquiera de los valores del conjunto. Por otro lado, cuando el atributo (a_j) es continuo, los valores válidos no forman un conjunto finito sino un rango real, por lo que la condición tiene la forma $a_j \in [li_j, ls_j]$, donde li_j y ls_j son los límites inferior y superior del intervalo que define el rango en el que el valor de a_j debe encontrarse para que la condición sea evaluada como cierta. Respecto a la clase, es importante señalar que HIDER opera sobre conjuntos de datos donde cada ejemplo es etiquetado con una única etiqueta de clase discreta. Por ello, una regla sólo podrá clasificar ejemplos de la misma clase, siendo el consecuente de la forma $Clase = E$, donde E es una etiqueta discreta. La figura 5.3 muestra un ejemplo de una regla que clasifica con clase *B* a aquellos ejemplos que cumplan las dos condiciones establecidas en el antecedente.

Cada condición de la regla es codificada por un único gen, el cual es un número natural de rango finito independientemente del tipo de dominio de dicha condición. Por tanto, para un conjunto de datos con m atributos incluyendo la clase, los individuos tendrán longitud fija $L = m$, donde el alfabeto de símbolos de todos los genes será finito y, como consecuencia, el tamaño del espacio de búsqueda también lo será (véase la ecuación 5.1). Además, al contrario de la codificación híbrida, el método de codificación natural imposibilita la existencia de soluciones similares con representaciones diferentes, lo que reduce aún más el tamaño efectivo del espacio.

Dado que la semántica de una condición depende del dominio, el método de codificación de los valores de ambos tipos de atributos en un número natural así como su interpretación son radicalmente diferentes.

Atributos Discretos

Para exponer de manera clara la codificación natural para dominios discretos, estudiaremos inicialmente el caso de un único atributo, generalizando el método para espacios mayores más adelante.

El problema que aquí nos planteamos se puede resumir en la siguiente cuestión: ¿cómo representar un conjunto finito de valores con un único número natural? Una respuesta trivial es la enumeración de los valores del atributo, comenzando en 1.

Ejemplo 5.1. *Supongamos que tenemos un atributo a_i discreto con un conjunto de valores nominales A={blanco, rojo, verde, azul, negro}, entre los cuales no existe una relación de orden. La posible representación sería $\Omega_i = \{1, 2, 3, 4, 5\}$, siendo blanco=1 y negro=5. Nótese que esta codificación obliga que las condiciones sean del tipo $a_i = v$, con $v \in \Omega_i$.*

Esta solución complicaría extremadamente la aplicación de los operadores genéticos, sobre todo si no existe un orden preestablecido en los valores del atributo. Por ejemplo ¿cómo mutar el color rojo? o ¿qué descendencia genera el cruce entre verde y azul? Además, la simple enumeración no permite la posibilidad de que un gen represente varios valores simultáneamente, multiplicándose el número de reglas posibles, ya que una condición como $a_i \in \{rojo, negro\}$ daría lugar a dos reglas distintas, una con $a_i = rojo$ y otra $a_i = negro$, pero idénticas para el resto de condiciones.

La problemática de la simple enumeración se soluciona asignando un número natural a cada posible combinación de valores. Partiendo de la codificación binaria que asigna un bit a cada posible valor, denotando con 1 y 0 la presencia o ausencia del valor en la condición respectivamente, la codificación natural transforma esa cadena binaria en su representación decimal. Así, un gen discreto es un número natural que identifica un conjunto de valores discretos y pertenece al intervalo $[0, 2^{|A|} - 1]$, donde $|A|$ es el cardinal del conjunto de valores posibles del atributo. La Tabla 5.1 muestra un ejemplo de codificación natural para el atributo discreto descrito en el ejemplo 5.1.

Valores Discretos					Codificación
blanco	rojo	verde	azul	negro	Natural
0	0	0	0	0	–
0	0	0	0	1	**1**
0	0	0	1	0	**2**
0	0	0	1	1	**3**
⋮	⋮	⋮	⋮	⋮	⋮
0	1	0	1	1	**11**
⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	0	**30**
1	1	1	1	1	**31**

Tabla 5.1: Codificación natural de un atributo discreto.

Como se puede observar, el código 0 es omitido en el conjunto de valores codificados al carecer éste de sentido, ya que denotaría la ausencia de todos los valores del atributo en la condición y, por lo tanto, ningún ejemplo podría ser clasificado por esa regla. Nótese también que, en general, el último código $(2^{|A|} - 1)$ significa que todos los valores están en la condición, en otras palabras, ese atributo no se tendrá en cuenta a la hora de clasificar ejemplos aplicando la regla correspondiente, pues dicho atributo puede tomar cualquier valor.

El hecho de que exista una relación directa entre la codificación binaria y la natural discreta puede inducir a pensar que es necesario una reconversión al código binario para aplicar los operadores genéticos. Sin embargo, como se describe en la sección 5.3.3, tanto el cruce como la mutación son operaciones algebraicas simples de bajo coste

computacional que transforman los genes naturales sin necesidad de decodificar tales valores.

La clase es el atributo sobre el cual se van a tomar las posteriores decisiones, denominándose también atributo de decisión. HIDER opera con conjuntos de datos con clase exclusivamente discreta, por lo que, a priori, podríamos considerar la clase como una cualidad discreta más y usar así el mismo método de codificación. Sin embargo, cada regla sólo clasifica ejemplos para una clase, no siendo necesario contemplar todas las posibles combinaciones de etiquetas. Por ello, se ha optado por un método más sencillo para representar la clase en los individuos naturales que es la enumeración de las etiquetas. Por motivos de implementación, dicha enumeración comienza en 0.

Ejemplo 5.2. *Supongamos un conjunto de datos con tres atributos, todos discretos, y una clase. El conjunto de valores para cada uno de ellos es:* a_1*={pequeño, mediano, grande},* a_2*={blanco, rojo, verde, azul, negro},* a_3*={sí, no} y Clase={A, B, C}. La figura 5.4 muestra un ejemplo de codificación natural de una regla para este conjunto de datos.*

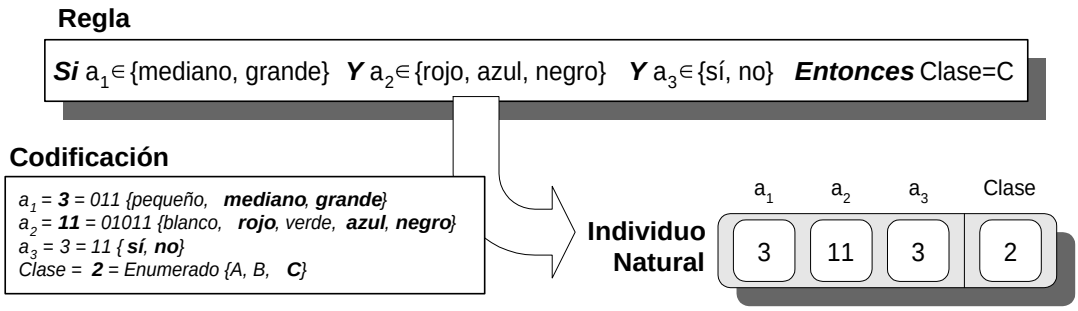


Figura 5.4: Ejemplo de Regla de Decisión Discreta.

Nótese el hecho de que la condición para el atributo a_3 no impone restricción alguna al contener todos los valores posibles que éste puede tomar. Cuando esto ocurre, dicha condición se suele omitir en la regla para simplificar su comprensión, aunque no se puede eliminar del individuo.

Atributos Continuos

El método de codificación natural de atributos continuos es más complejo que en el caso de los discretos, debido principalmente a la necesidad de disminuir la cardinalidad del conjunto de posibles valores que un atributo puede tomar. Para llevar a cabo esta reducción se hace imprescindible la aplicación de un método de discretización, el cual devolverá un conjunto finito de intervalos o, en general, un conjunto de cortes o posibles límites de esos intervalos. Podría pensarse que, una vez discretizados los valores de un atributo, es posible aplicar la codificación natural para atributos discretos. Sin embargo, la relación de orden e inclusión existente entre los intervalos imposibilita esta solución para el tipo de reglas que pretendemos obtener.

Ejemplo 5.3. *Supongamos que tenemos un atributo continuo a_1 que ha sido discretizado, dando el conjunto de cortes $A_1 = \{1.0, 1.5, 2.0, 2.5, 3.0\}$. Por otro tenemos el atributo discreto a_2, cuyos posibles valores son $A_2 = \{blanco, rojo, verde, azul, negro\}$. Una posible regla sería "Si $a_1 \in [1.5, 3.0]$ y $a_2 \in \{rojo, negro\}$ Entonces C". Estas dos condiciones tienen interpretaciones totalmente distintas. La primera exige que a_1 tenga valores comprendidos entre 1.5 y 3.0, pero no necesariamente esos, pudiendo tomar valores que ni siquiera están en el conjunto de cortes. Por el contrario, la segunda obliga que a_2 sea $rojo$ o $negro$, no permitiendo que tome otros valores, aunque éstos estén en posiciones intermedias en el conjunto de datos A_2. Por tanto, la aplicación de la codificación natural discreta sobre atributos continuos discretizados no es válida.*

El primer problema al que nos enfrentamos es la elección o diseño de un método de discretización que no produzca pérdida de precisión en las reglas. Parece obvio que dicho método ha de ser supervisado, de modo que los intervalos o cortes que genere dependan de la clase que cada ejemplo toma en el conjunto de datos. Por ello optamos por la aplicación del algoritmo USD [74] (*Unparametrized Supervised Discretization*) descrito en el Capítulo 4, que devuelve un conjunto de cortes, los cuales serán los posibles límites de los intervalos. Una propiedad importante de esta discretización es que no supone penalización en la precisión de las potenciales reglas debido a que USD maximiza la bondad de los intervalos que obtiene.

Cortes	2.5	3.9	4.7	5.0	6.2
1.4	$\mathbf{1} \equiv [1.4, 2.5]$	$\mathbf{2} \equiv [1.4, 3.9]$	$\mathbf{3} \equiv [1.4, 4.7]$	$\mathbf{4} \equiv [1.4, 5.0]$	$\mathbf{5} \equiv [1.4, 6.2]$
2.5	–	$\mathbf{7} \equiv [2.5, 3.9]$	$\mathbf{8} \equiv [2.5, 4.7]$	$\mathbf{9} \equiv [2.5, 5.0]$	$\mathbf{10} \equiv [2.5, 6.2]$
3.9	–	–	$\mathbf{13} \equiv [3.9, 4.7]$	$\mathbf{14} \equiv [3.9, 5.0]$	$\mathbf{15} \equiv [3.9, 6.2]$
4.7	–	–	–	$\mathbf{19} \equiv [4.7, 5.0]$	$\mathbf{20} \equiv [4.7, 6.2]$
5.0	–	–	–	–	$\mathbf{25} \equiv [5.0, 6.2]$

Tabla 5.2: Tabla de codificación para un atributo continuo.

Una vez obtenidos los cortes, el método de codificación natural asigna un número natural a cada posible combinación de límites. La Tabla 5.2 muestra un ejemplo de codificación natural para un atributo continuo cuyos cortes calculados son {1.4, 2.5, 3.9, 4.7, 5.0, 6.2}. Ésta se denomina *tabla de codificación*, pues es la llave para pasar de la representación natural a los intervalos correspondientes. Si k es el número de cortes, las filas de la tabla están etiquetadas con los $k-1$ primeros cortes (desde el primero hasta el $(k-1)$-ésimo) y las columnas con los $k-1$ últimos (desde el segundo al último). Cada elemento de la tabla n_{ij} es un número natural que identifica al intervalo $[i, j]$. Los elementos se numeran empezando desde el 1, de izquierda a derecha y desde arriba a bajo. Aquellos números n_{ij} que codifiquen a intervalos no válidos, es decir todos los $[i, j]$ donde $i \geq j$, no son considerados y se representan por "$-$". El natural (en negrilla) asignado a cada combinación válida de intervalos en la tabla de codificación es el valor que el gen natural tomará en el individuo.

Evidentemente, la aplicación del algoritmo de discretización junto a la codificación de los distintos intervalos ha reducido el tamaño del espacio de búsqueda respecto a la codificación real e híbrida. En general, el rango de valores de un gen natural continuo para k cortes será finito e igual a $[1, (k-1)^2]$, aunque no todos los valores son válidos. Por ejemplo, de los 25 códigos posibles en la tabla de codificación 5.2, sólo 15 son válidos, siendo omitidos los números 6, 11, 12, 16, 17, 18, 21, 22, 23 y 24, pues representan intervalos no válidos. Aunque el estudio de la reducción del espacio de búsqueda se llevará a cabo en la sección 5.3.2, es interesante analizar aquí el número de valores distintos y válidos del conjunto de naturales (Ω_c) proporcionados por esta codificación. El

número de intervalos válidos y, por tanto, el cardinal de Ω_c es exactamente

$$|\Omega_c| = \binom{k}{2} = \frac{k(k-1)}{2} \tag{5.2}$$

donde k es el número de cortes obtenidos tras la discretización. ¿Hubiera sido mejor asignar un código natural a cada valor distinto del atributo en el conjunto de datos? De la ecuación 5.2 podemos deducir que la codificación natural es ventajosa cuando $\frac{k(k-1)}{2} < v$, siendo v el número de valores distintos en el conjunto de datos para el atributo en estudio. Entonces, teniendo en cuenta que el número de cortes siempre es menor o como mucho igual a v ($k \leq v$), tenemos

$$k(k-1) < 2v \Rightarrow k^2 < 2v + k \leq 3v \Rightarrow k < \sqrt{3v} \tag{5.3}$$

Por tanto, si $k < \sqrt{3v}$, la codificación natural reduce el espacio respecto a la simple enumeración de valores distintos.

Una vez que se tiene la tabla de codificación, la obtención del intervalo correspondiente a un determinado valor natural es directa. Esta transformación la denominamos *decodificación*, y se reduce a encontrar las expresiones que, dado un número natural, calculen su fila y columna en la tabla. Una vez obtenidas, la fila nos dará el límite inferior del intervalo y la columna el superior.

Definición 5.1 (Fila y columna). *Sea n un valor del gen natural y k el número de cortes para un atributo continuo. Se denominan f y c a la fila y la columna, respectivamente, correspondientes a n en la tabla de codificación para dicho atributo, siendo su cálculo*

$$\begin{aligned} f &= \left\lfloor \frac{n-1}{k-1} \right\rfloor + 1 \\ c &= (n-1)\,\%(k-1) + 1 \end{aligned} \tag{5.4}$$

donde k es el número de cortes; $\lfloor \cdot \rfloor$ es la parte entera por defecto; y % es el resto de la división entera.

Del mismo modo que el proceso de decodificación proporciona el intervalo asociado a un número natural, el proceso de *codificación* transforma un intervalo en su código

natural mediante la fila y la columna correspondientes a los límites inferior y superior respectivamente. Esta transformación también es directa aplicando la expresión

$$n = (f-1)(k-1) + c \tag{5.5}$$

El tamaño de la tabla de codificación depende del número de cortes, concretamente es $(k-1)^2$. Si el número de cortes es elevado, el almacenamiento de la tabla en memoria sería computacionalmente muy costoso en espacio. Sin embargo, si analizamos la codificación y decodificación de genes, así como la estructura de la tabla, podemos colegir que no es necesario crear ni conservar la tabla de codificación en memoria. Por un lado, las filas y columnas de la tabla comparten los $(k-2)$ cortes centrales del conjunto total del cortes. Por otra parte, tanto la codificación como la decodificación de un gen natural continuo se lleva a cabo mediante el cálculo de expresiones algebraicas simples. Por tanto, la única información necesaria es el valor de los cortes, los cuales son almacenados en un vector de k posiciones, evitando así tener que conservar la tabla completa. La figura 5.5 ilustra esta idea mostrando el mapeo de la tabla de codificación 5.2. No obstante, aunque dicha tabla sea innecesaria en la implementación, seguiremos usándola para posteriores ejemplos con el propósito de clarificar la exposición de este trabajo.

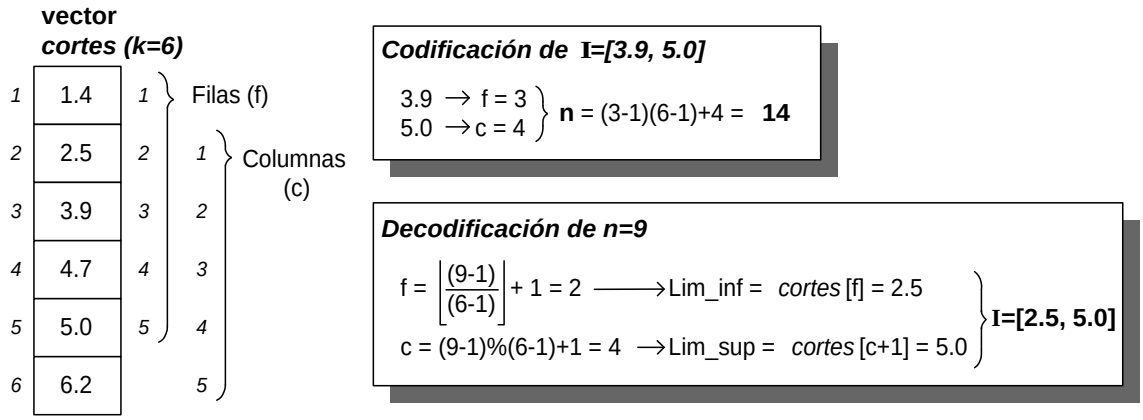


Figura 5.5: Mapeo de la tabla de codificación (tabla 5.2).

5.3.2. Reducción del espacio de búsqueda

La aplicación del método de codificación natural genera individuos de longitud fija, donde cada gen es un número natural que representa una condición de la regla de decisión. Si comparamos esta representación con otras como la híbrida, un individuo natural es mucho más compacto. La Figura 5.6 ilustra la diferencia entre la codificación híbrida y la codificación natural usada en HIDER, mostrando dos individuos que codifican la misma regla. El atributo a_1 es continuo y a_2 es discreto, siendo codificados de acuerdo con las Tablas 5.2 y 5.1 respectivamente.

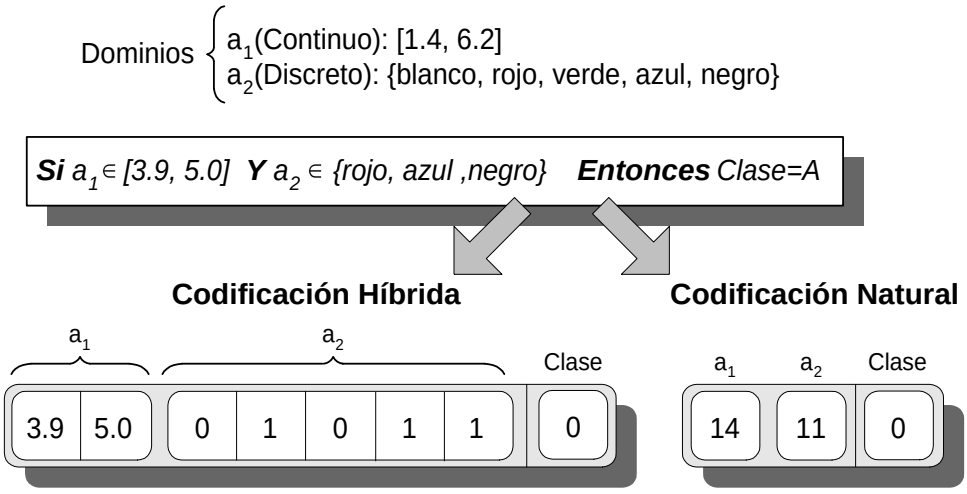


Figura 5.6: Codificación Híbrida vs. Codificación Natural.

Como se puede observar, la codificación natural reduce la longitud de los individuos respecto a la híbrida, ya que esta última precisa de ocho genes para codificar una regla mientras que la codificación natural sólo necesita tres (uno para cada cualidad y uno para la clase). En general, si NC y ND es el número de atributos continuos y discretos respectivamente en el conjunto de datos y NV es el número total de valores discretos teniendo en cuenta todas las cualidades discretas, entonces la longitud de un individuo natural es $L_n = NC + ND + 1$, frente a la longitud de la codificación híbrida $L_h = 2 \times NC + NV + 1$. El último sumando en ambas expresiones corresponde al gen que representa la clase.

Como se puede colegir de la ecuación 5.1, la longitud de los individuos (L) es un

factor determinante en el tamaño del espacio de búsqueda junto al cardinal del alfabeto de símbolos de cada gen (Ω_i). No obstante, cabe discernir entre el espacio definido por las características discretas y continuas, ya que son estas últimas las verdaderas causantes de la disminución de soluciones candidatas usando codificación natural. Si a_i es un atributo discreto con A_i valores distintos, la codificación híbrida precisa de un número de genes $L_i = A_i$, cuyo alfabeto es $\Omega_i^h = \{0, 1\}$. Con las mismas condiciones, la codificación natural utiliza sólo un gen, aunque en este caso el alfabeto viene definido como $\Omega_i^n = \{n \in \mathbb{N} \mid 0 \leq n \leq (2^{A_i} - 1)\}$. Basándonos en la ecuación 5.1 (véase página 116), la ecuación 5.6 muestra el tamaño del subespacio que a_i define aplicando codificación híbrida o binaria (S_i^h), mientras que la ecuación 5.7 hace lo propio aplicando la natural (S_i^n). Comparando ambas expresiones, observamos que $S_i^h = S_i^n$, lo cual demuestra que no se produce reducción efectiva del tamaño del espacio para atributos discretos.

$$S_i^h = \prod_{j=1}^{L_i} |\Omega_j| = \prod_{j=1}^{L_i} 2 = 2^{L_i} = 2^{A_i} \tag{5.6}$$

$$S_i^n = \prod_{j=1}^{L_i} |\Omega_j| = \prod_{j=1}^{1} 2^{A_i} = 2^{A_i} \tag{5.7}$$

Esto no ocurre en el caso de cualidades con dominio continuo, donde el tamaño del espacio si es influenciado realmente por la disminución del número de genes. Concretamente, la longitud de un individuo natural cuando sólo intervienen este tipo de atributos es $L^n = NC$, frente a $L^h = 2 \times NC$ usada por la representación híbrida. Sin embargo, aunque esta disminución de la longitud individual puede suponer un aspecto notable en la reducción del tamaño del espacio de búsqueda, el factor más influyente en dicha reducción es la simplificación del alfabeto de símbolos de las cualidades continuas tras la aplicación del algoritmo de discretización USD. Concretamente, el número de símbolos para un atributo concreto viene determinado por el número de cortes que USD calcula para al mismo. Como muestra la tabla 5.2, el cardinal efectivo del alfabeto ($|\Omega_c|$), contando solamente aquellos valores válidos de la representación, es igual al número de intervalos que los cortes pueden definir (véase la ecuación 5.2).

Por tanto, el tamaño del subespacio de búsqueda definido por un atributo continuo no sólo es finito sino relativamente pequeño, como más adelante demuestran las pruebas realizadas.

5.3.3. Operadores Genéticos Naturales

La codificación natural representa cada atributo con un numero natural con independencia del tipo de dominio. Sin embargo, puesto que el significado y la interpretación de esos números sí es diferente para atributos continuos y discretos, los operadores genéticos también lo son, por lo que estudiaremos ambos tipos de dominios por separado.

Un aspecto importante es el hecho de que los individuos naturales tienen longitud fija y cada gen tiene una posición determinada dentro del individuo, siendo cada uno de éstos totalmente independiente del resto de genes. Esto afecta directamente los operadores genéticos, sobre todo al de cruce, ya que el i-ésimo gen de un individuo sólo podrá intervenir en el cruce interactuado con el i-ésimo gen del otro padre, produciendo igualmente el gen de la posición i de la descendencia. Además, la aplicación de los operadores sobre un gen no afecta al resto del individuo. Por ello, podemos particularizar los operadores hablando de mutación y cruce entre genes para simplificar la comprensión de los mismos.

Atributos Discretos

Con el propósito de clarificar la descripción de los operadores, analicemos el caso para un único gen discreto, para generalizar con mayor número de genes más tarde. En concreto, basaremos el estudio en el atributo codificado en la tabla 5.1, cuyos valores posibles son $A = \{blanco, rojo, verde, azul, negro\}$.

Mutación

Partimos de un número natural cuyos bits en representación binaria denotan la presencia o ausencia de un valor en una condición. La mutación consiste en cambiar el valor del gen por otro símbolo del alfabeto que represente un conjunto de valores distinto al inicial donde simplemente se ha agregado o suprimido un valor. Esta mutación aplicada en la codificación binaria tiene una implementación trivial consistente en seleccionar un bit al azar y cambiar su valor de 0 a 1 o al revés, según el caso. La figura 5.7 ilustra este tipo de mutación.

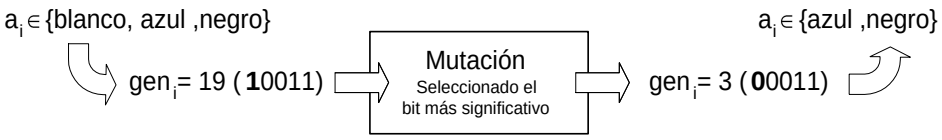


Figura 5.7: Ejemplo de mutación natural discreta.

Una primera aproximación a la solución del problema sería transformar el número natural en binario y aplicar la mutación directamente sobre los bits para luego reconvertir el nuevo conjunto de bits en su representación natural. Evidentemente, aunque esta solución es válida, también es muy ineficiente, ya que se produce un coste computacional aditivo por la realización de las conversiones. ¿Cómo aplicar la mutación sin realizar conversiones de binario a natural o viceversa? La respuesta no es simple.

Estudiemos qué situaciones pueden darse y qué valores deberían obtenerse aplicando la mutación en cada una de ellas. Como tenemos cinco valores en el conjunto A, el gen puede tomar 2^5 valores distintos, entre 0 y 31. Para cada posible valor del gen, se pueden dar cinco posibles resultados tras la mutación de un único bit. La tabla 5.3 ilustra las posibles mutaciones que un gen puede sufrir dependiendo del bit que sea alterado. La primera columna muestra el valor inicial del gen, mientras que las siguientes cinco columnas dan el valor resultante tras la mutación al invertir el k-ésimo bit (bit_k) de la representación binaria inicial, siendo el bit_5 el más significativo y el bit_1 el menos significativo. Es importante señalar que, como se ha razonado anteriormente, aunque un gen natural nunca podrá tomar valor 0, el estudio ha sido realizado sin discriminar este valor, dejando tal exclusión para la posterior implementación de la codificación. Así, la interpretación de la tabla es: si el gen es 0, la mutación cambia el gen al valor 16 si el bit seleccionado es el bit_5, así hasta el valor 1 si el bit elegido es el menos significativo. La lectura para el resto de filas de la tabla es similar.

Llegado este punto, nos planteamos la siguiente cuestión: ¿existe una expresión que dado un número natural n obtenga directamente el número resultante de cambiar el bit_k de la representación binaria de n? La respuesta es sí.

Definición 5.2 (Mutación natural discreta del k-ésimo bit). *Sea A el conjunto de valores posibles de un atributo discreto, y $|A|$ su cardinal. Sea k un número entero entre 1 y $|A|$. Se define mutación natural discreta del k-ésimo bit de un gen natural n y se denotará*

Gen		Conjunto de posibles mutaciones				
Valor		$+Sig$				$-Sig$
Natural		bit_5	bit_4	bit_3	bit_2	bit_1
0	$\rightarrow$	16	8	4	2	1
1	$\rightarrow$	17	9	5	3	0
2	$\rightarrow$	18	10	6	0	3
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
30	$\rightarrow$	14	22	26	28	31
31	$\rightarrow$	15	23	27	29	30

Tabla 5.3: Posibles mutaciones de un gen natural discreto.

como $mut_k(n)$, como resultado de invertir el bit k de la representación binaria de n, y cuya expresión es

$$mut_k(n) = \left(n + 2^{k-1}\right) \%2^k + 2^k \left\lfloor \frac{n}{2^k} \right\rfloor \tag{5.8}$$

donde % es el operador módulo que obtiene el resto de la división entera; y $\lfloor\_\rfloor$ es el operador suelo, es decir, la parte entera por defecto.

Ejemplo 5.4. *Si el valor del gen es $n = 30$ (11110 en binario) para el atributo usado como ejemplo, dicho número representa la condición $a_i \in \{blanco, rojo, verde, azul\}$. El conjunto de posibles mutaciones es $\{14, 22, 26, 28, 31\}$, como muestra la tabla 5.3. Estos valores son obtenidos aplicando $mut_k(n)$, con $k \in \{1, 2, 3, 4, 5\}$. La figura 5.8 muestra los cálculos para cada uno de los k posibles.*

Como muestra el ejemplo anterior, se ha conseguido mutar cualquier valor de un gen natural aplicando una simple expresión algebraica, seleccionando previamente qué valor del conjunto discreto se quiere alterar mediante el operando k.

Definición 5.3 (Probabilidad de mutación por valor). *Se define la probabilidad de mutación por valor ($\mathcal{P}mv$) como la probabilidad de que cada valor del conjunto A sea seleccionado para ser mutado, es decir, la probabilidad de aplicar la mutación natural discreta del k-ésimo bit (mut_k) para un k concreto. Si elegimos aleatoriamente η valores distintos para k, probabilidad de mutación por valor es $\mathcal{P}mv = \frac{\eta}{|A|}$.*

$$mut_1(30) = (30 + 2^0)\ \%2^1 + 2^1 \left\lfloor \frac{30}{2^1} \right\rfloor = 31 \rightarrow \{blanco, rojo, verde, azul, negro\}$$
$$mut_2(30) = (30 + 2^1)\ \%2^2 + 2^2 \left\lfloor \frac{30}{2^2} \right\rfloor = 28 \rightarrow \{blanco, rojo, verde\}$$
$$mut_3(30) = (30 + 2^2)\ \%2^3 + 2^3 \left\lfloor \frac{30}{2^3} \right\rfloor = 26 \rightarrow \{blanco, rojo, azul\}$$
$$mut_4(30) = (30 + 2^3)\ \%2^4 + 2^4 \left\lfloor \frac{30}{2^4} \right\rfloor = 22 \rightarrow \{blanco, verde, azul\}$$
$$mut_5(30) = (30 + 2^4)\ \%2^5 + 2^5 \left\lfloor \frac{30}{2^5} \right\rfloor = 14 \rightarrow \{rojo, verde, azul\}$$

Figura 5.8: Ejemplo de posibles mutaciones discretas.

Definición 5.4 (Mutación natural discreta). *La mutación natural discreta consiste en calcular un número natural a partir de otro de forma que el primero represente la mutación de η bits seleccionados al azar en la representación binaria del segundo sin realizar conversiones de natural a binario o viceversa. Normalmente η es 1, alterándose un único valor de la condición original, siendo por tanto a probabilidad de mutación por valor $\mathcal{P}mv = \frac{1}{|A|}$.*

Aunque el operador de mutación natural da buenos resultados por sí sólo, en ocasiones es interesante introducir otro tipo de mutación que elimine por completo una condición del individuo. Esto es debido a que en el conjunto de datos pueden existir atributos que no aporten información en la toma de decisiones, e incluso que, aún siendo útiles para clasificar ejemplos de algunas clases, no lo sean para otras. Por ello, la eliminación de las condiciones establecidas sobre los valores de esas cualidades puede disminuir la complejidad del modelo reduciendo la longitud de las reglas. Para este caso definimos el siguiente operador de mutación.

Definición 5.5 (Mutación generalizada discreta). *La mutación natural generalizada discreta asigna al gen el numero natural $2^{|A|} - 1$ con independencia del valor original del gen, siendo $|A|$ el cardinal del conjunto de posibles valores del atributo.*

Nótese que, al asignar al gen el máximo valor del rango de posibles códigos, la condición correspondiente a ese número natural es $a_i \in A$, es decir, el atributo puede tomar cualquier valor, por lo que esa condición es eliminada de la regla final al evaluarse siempre como cierta. Por ejemplo, según la tabla 5.1, si el gen toma el valor 31,

la condición es $a_i \in \{blanco, rojo, verde, azul, negro\}$. El operador de mutación generalizada es aplicado con probabilidad muy baja para no repercutir negativamente en la precisión de las reglas. No obstante, si su aplicación resulta beneficiosa en ciertos individuos, éstos tenderán a replicarse y reproducirse, favoreciendo la evolución del modelo.

Cruce

El cruce natural para atributos discretos es una particularización del cruce uniforme [170] definido para la codificación binaria. Al igual que en la mutación, nuestro propósito es que el cruce pueda ser aplicado sin necesidad de trasformaciones entre naturales y binarios. De hecho, el cruce se basa en la mutación natural definida anteriormente. Cada gen que interviene en el cruce proporciona un conjunto de candidatos. Estos candidatos son el resultado de unir el conjunto de posible mutaciones del gen con el propio gen. Así, la descendencia de dos genes se calcula como la intersección de los conjuntos de candidatos que cada gen aporta. Cuando los padres no ofrecen candidatos comunes, se calculan nuevos candidatos para cada padre mutando los conjuntos iniciales hasta que la intersección no es vacía. La Figura 5.9 muestra ejemplos de los operadores naturales para atributos discretos basados en la Tabla 5.1. El gen codificado con el número natural 11 tiene como código binario el 01011. El bloque a la derecha de la llave da las posibles mutaciones que este gen puede sufrir (en negrita el bit que muta en cada caso). El ejemplo para el gen codificado con el 19 es similar. Así, los conjuntos de posibles mutaciones de 11 y 19 son $\{10, 9, 15, 3, 27\}$ y $\{18, 17, 23, 27, 3\}$, respectivamente, mientras que el cruce entre ambos genera el conjunto de genes $\{3, 27\}$, pues es la intersección de los dos conjuntos anteriores.

Para dar una definición formal del cruce natural para atributos discretos es necesario definir una serie de conceptos previos.

Definición 5.6 (Clase Mutación). *Definimos clase mutación de un gen natural discreto n, y la denotamos con $Mut(n)$, como el conjunto de las k posibles mutaciones que puede sufrir n.*

$$Mut(n) = \bigcup_{k=1}^{|A|} mut_k(n) \tag{5.9}$$

$$\begin{array}{c} 11 = 01011 \\ \{rojo, azul, negro\} \end{array} = \begin{cases} 0101\mathbf{0}{=}10{\equiv}\{rojo, azul\} \\ 010\mathbf{0}1{=}\ 9\ {\equiv}\{rojo, negro\} \\ 01\mathbf{1}11{=}15{\equiv}\{rojo, verde, azul, negro\} \\ \mathbf{0}0011{=}\ 3\ {\equiv}\{azul, negro\} \\ \mathbf{1}1011{=}27{\equiv}\{blanco, rojo, azul, negro\} \end{cases}$$

$$\begin{array}{c} 19 = 10011 \\ \{blanco, azul, negro\} \end{array} = \begin{cases} 1001\mathbf{0}{=}18{\equiv}\{blanco, azul\} \\ 100\mathbf{0}1{=}17{\equiv}\{blanco, negro\} \\ 10\mathbf{1}11{=}23{\equiv}\{blanco, verde, azul, negro\} \\ 1\mathbf{1}011{=}27{\equiv}\{blanco, rojo, azul, negro\} \\ \mathbf{0}0011{=}\ 3\ {\equiv}\{azul, negro\} \end{cases}$$

$$mutaciones(11) = \{10, 9, 15, 3, 27\}$$
$$mutaciones(19) = \{18, 17, 23, 27, 3\}$$

$$cruce(11, 19) \in \{mutaciones(11) \oplus 11\} \bigcap \{mutaciones(19) \oplus 19\} =$$
$$= \{10, 9, 15, 3, 27, 11\} \bigcap \{18, 17, 23, 27, 3, 19\} = \{3, 27\}$$

Figura 5.9: Mutación y cruce para atributos discretos.

Definición 5.7 (Mutación de un conjunto). *Sea $\mathcal{Z}$ un conjunto de genes, se define mutación de un conjunto, denotado por $Mut(\mathcal{Z})$, como el conjunto que contiene todas las clases mutación de cada uno de los elementos del conjunto $\mathcal{Z}$.*

$$Mut(\mathcal{Z}) = \bigcup_{n \in \mathcal{Z}} Mut(n) \tag{5.10}$$

Definición 5.8 (Clausura de la mutación). *Sea $\mathcal{Z}$ un conjunto de genes, definimos clausura de la mutación de un conjunto, denotado por $\overline{Mut}(\mathcal{Z})$, como la mutación de la unión de $\mathcal{Z}$ con la mutación del propio $\mathcal{Z}$.*

$$\overline{Mut}(\mathcal{Z}) = Mut(Mut(\mathcal{Z}) \cup \mathcal{Z}\} \tag{5.11}$$

Definición 5.9 (Clase mutación de orden-j). *Si $\mathcal{Z}$ es un conjunto de genes naturales discretos, definimos la clase mutación de orden-j, denotada por $[Mut(\mathcal{Z})]^j$, como:*

$$\begin{aligned} [Mut(\mathcal{Z})]^0 &= \mathcal{Z} \\ [Mut(\mathcal{Z})]^1 &= \overline{Mut}(\mathcal{Z}) \\ &\vdots \\ [Mut(\mathcal{Z})]^j &= \overline{Mut}([Mut(\mathcal{Z})]^{j-1}) \end{aligned} \tag{5.12}$$

El cruce natural discreto entre dos genes naturales será un número perteneciente al conjunto resultante de la intersección entre las clases mutación de orden-j del los

padres para el primer orden-j que evite que dicha intersección sea vacía. La definición formal es la siguiente.

Definición 5.10 (Cruce natural discreto). *Sean n_i^x y n_i^y dos genes naturales pertenecientes a dos individuos de la población, x e y respectivamente, y que codifican al i-ésimo atributo. Sea también el conjunto $Q^t = \{[\overline{Mut}(Mut(n_i^x))]^t \cap [\overline{Mut}(Mut(n_i^y))]^t\}$, con $t > 0$. Si denotamos por n_i^d el gen de la descendencia, entonces definimos cruce natural, $cruce(n_i^x, n_i^y)$, como:*

$$cruce(n_i^x, n_i^y) = n_i^d \in \{Q^t \mid \forall s : 0 \leq s < t \cdot Q^s = \emptyset, Q^t \neq \emptyset\} \tag{5.13}$$

Ejemplo 5.5. *Supongamos que se quieren cruzar dos individuos, x e y, cuyos genes para el i-ésimo atributo según la tabla 5.1 son $n_i^x = 22$ y $n_i^y = 11$ respectivamente. El i-ésimo gen de cada descendiente resultado de cruzar x e y tomará un valor del conjunto $\{2, 7, 14, 19, 26, 31\}$, siendo la selección de tal valor aleatoria, aunque distinta para cada hijo.*

$n_i^x = 22 \Rightarrow Mut(22) = \{6, 18, 20, 22, 23, 30\}$
$n_i^y = 11 \Rightarrow Mut(11) = \{3, 9, 10, 11, 15, 27\}$

$[\overline{Mut}(Mut(22))]^0 = \{6, 18, 20, 22, 23, 30\}$
$[\overline{Mut}(Mut(11))]^0 = \{3, 9, 10, 11, 15, 27\}$
$\hookrightarrow Q^0 = [\overline{Mut}(Mut(22))]^0 \cap [\overline{Mut}(Mut(11))]^0 = \emptyset$

$[\overline{Mut}(Mut(22))]^1 = \{2, 4, 6, 7, 14, 16, 18, 19, 20, 21, 22, 23, 26, 28, 30, 31\}$
$[\overline{Mut}(Mut(11))]^1 = \{1, 2, 3, 7, 8, 9, 10, 11, 13, 14, 15, 19, 25, 26, 27, 31\}$
$\hookrightarrow Q^1 = [\overline{Mut}(Mut(22))]^1 \cap [\overline{Mut}(Mut(11))]^1 = \{2, 7, 14, 19, 26, 31\} \neq \emptyset$

$$cruce(22, 11) \in Q^1 = \{2, 7, 14, 19, 26, 31\}$$

Mutación y cruce de la clase

Aunque la clase es un atributo discreto, su codificación es diferente a la de éstos, siendo una simple enumeración de las diferentes etiquetas presentes en el conjunto de

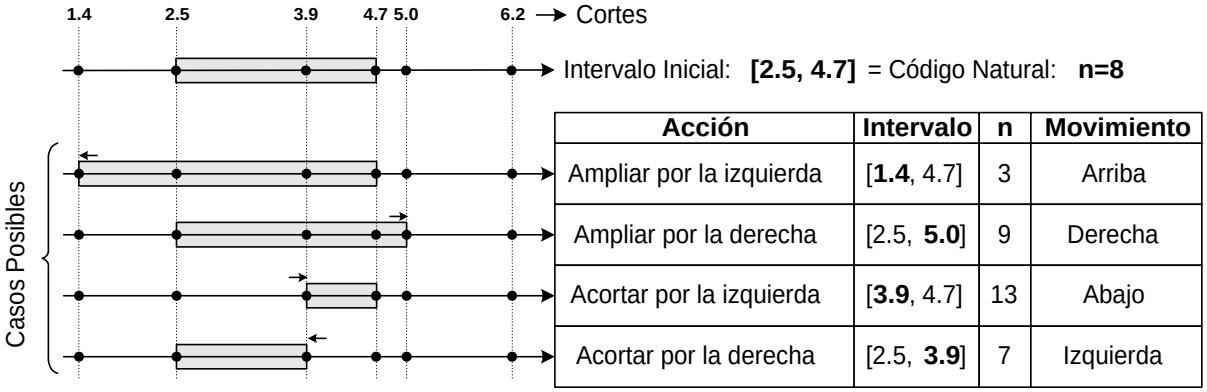
datos. Por ello, los operadores genéticos son totalmente diferentes y mucho más simples. La mutación consiste en un cambio de valor dentro del rango de valores de la clase. Aunque es fácilmente aplicable, la mutación de la clase puede ser muy perjudicial, sobre todo en las últimas generaciones donde los individuos ya están altamente especializados. Por ello, aunque podría tener sentido en las primeras generaciones, en este trabajo hemos optado por no aplicar mutación a la clase. En cuanto al cruce, la descendencia hereda la etiqueta de clase de uno de los padres.

Atributos Continuos

Al igual que en el caso de atributos discretos, estudiaremos los operadores genéticos continuos para un único gen, apoyando la exposición es el ejemplo de la tabla 5.2, y generalizando los distintos conceptos a medida que sea necesario.

En general, un gen natural sólo puede tomar ciertos valores que hemos denominado "válidos", los cuales codifican unos intervalos concretos. Ello implica que, tras la aplicación de cualquier operador genético, el valor del gen siempre será uno de los códigos válidos de la tabla de codificación. Por tanto, el cambio de un código natural, localizado en una determinada fila y columna en la tabla, no es más que un desplazamiento en dicha tabla cambiando las coordenadas. Sin embargo, no todas las posibles variaciones de valor son permitidas. El cambio mínimo que puede darse en un intervalo es desplazar uno de los límites hacia el corte inmediatamente anterior o posterior, pudiéndose dar las cuatro situaciones descritas por la figura 5.10, basada en el ejemplo de codificación de la tabla 5.2.

La parte izquierda de la figura 5.10 representa gráficamente los cuatro posibles casos derivados a partir del intervalo inicial $[2.5, 4.7]$, cuyo número natural asociado es $n = 8$. Para cada caso (*acción*), la tabla de la derecha describe: el intervalo resultante (*intervalo*), resaltando el límite que varía; el número natural asociado al nuevo intervalo (*n*); y el *movimiento* en la tabla de codificaciones para pasar del código natural inicial al nuevo. Por ejemplo, tomando la primera acción posible, para ampliar el intervalo por la izquierda, el límite inferior ha de cambiar su valor al corte inmediatamente anterior, dando como resultado el intervalo $[1.4, 4.7]$. Si buscamos en la tabla 5.2 tal intervalo,



Acción	Intervalo	n	Movimiento
Ampliar por la izquierda	[**1.4**, 4.7]	3	Arriba
Ampliar por la derecha	[2.5, **5.0**]	9	Derecha
Acortar por la izquierda	[**3.9**, 4.7]	13	Abajo
Acortar por la derecha	[2.5, **3.9**]	7	Izquierda

Figura 5.10: Posibles movimientos simples del gen n=8 en la tabla 5.2.

éste resulta ser el número natural 3, situado en la misma columna pero en la fila anterior al código inicial 8. En otras palabras, un desplazamiento hacia arriba en la tabla de codificaciones produce una ampliación del intervalo hacia la izquierda. Asimismo, los otros tres posibles movimientos producen las otras tres posibilidades de cambio en los límites del intervalo.

Definición 5.11 (Transiciones). *Si representamos la tabla de codificaciones como un diagrama de estados donde cada nodo representa un código natural y cada arco dirigido o cambio de estado un movimiento posible, entonces denominamos transiciones al conjunto de todos los posibles cambios de estado.*

La figura 5.11 muestra el conjunto de transiciones (arcos dirigidos) para diferentes número de cortes (k). El último diagrama representa las transiciones para el ejemplo usado en esta sección donde el número de cortes es $k = 6$. Nótese que sólo son posibles transiciones entre elementos situados por encima de la diagonal principal de la tabla de codificación, ya que el resto de estados representan códigos naturales no válidos. Como veremos a continuación, tanto el operador de mutación como el de cruce están basados en este conjunto de transiciones.

Mutación

Desde el punto de vista del cambio en el fenotipo, es decir, la transformación de un intervalo al ser mutado, el operador de mutación natural es parecido al usado en la codificación híbrida, con la salvedad de que los límites del intervalo sólo pueden tomar

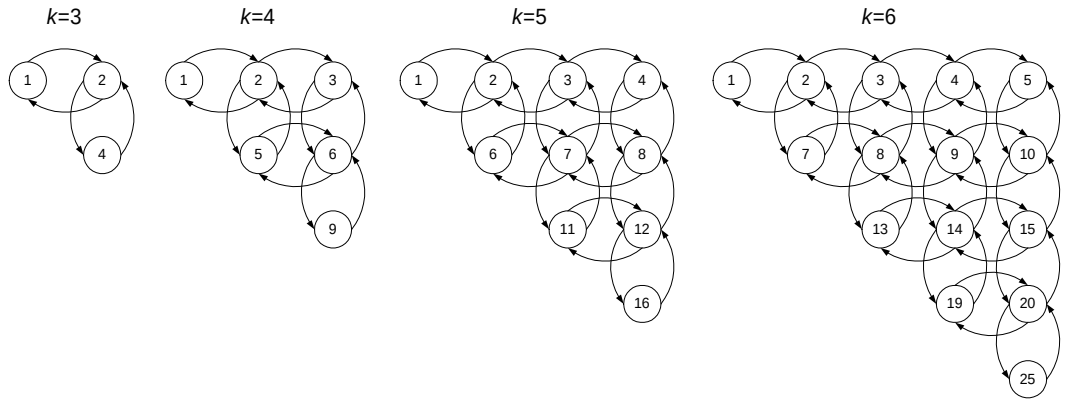


Figura 5.11: Transiciones.

valores dentro del conjunto de cortes. En concreto, la mutación consiste en cambiar a uno de los intervalos más cercanos al gen que queremos mutar, lo cual se reduce a aplicar alguna de las cuatro trasformaciones posibles sobre tal gen: derecha, izquierda, superior o inferior de la figura 5.11. Sin embargo, no siempre es posible aplicar todas estas transformaciones, ya que un gen no podrá trasformarse más allá de los límites de la tabla de codificación ni por debajo de la diagonal principal. Por ello, es necesario definir algunos conceptos antes de formalizar la definición de mutación natural continua.

Definición 5.12 (Limites). *Sea n el valor de un gen natural. Se denomina limites de n a los valores en el grafo que limitan las transiciones en las cuatro posibles direcciones, denotando cada uno de estos límites como: liz (izquierdo), lde (derecho), lsu (superior) y lin (inferior). El cálculo de estos límites se reduce a las siguientes expresiones:*

$$\begin{aligned} liz(n) &= (k-1)(f-1)+f \\ lde(n) &= (k-1)f \\ lsu(n) &= c \\ lin(n) &= (k-1)(c-1)+c \end{aligned} \tag{5.14}$$

donde f y c son la fila y columna correspondientes a n, respectivamente, calculadas mediante las expresiones de la ecuación 5.4; y k es el número de cortes.

Definición 5.13 (Movimientos). *Definimos movimientos a la izquierda, derecha, superior e inferior de un valor natural n como los posibles valores que puede alcanzar aplicando una única transformación válida.*

$$\begin{aligned} iz(n) &= max(liz(n), n-1) \\ de(n) &= min(lde(n), n+1) \\ su(n) &= max(lsu(n), n-k+1) \\ in(n) &= max(liz(n), n+k-1) \end{aligned} \tag{5.15}$$

Definición 5.14 (Conjunto de movimientos). *Si n un gen natural, denominamos conjunto de movimientos horizontal ($\overline{hor}$) a todos los movimientos que pueden realizarse en la fila de n. Análogamente, el conjunto de movimientos vertical ($\overline{ver}$) son los posibles movimientos en la columna de n. Ambos conjuntos incluyen el propio valor n.*

$$\begin{aligned} \overline{hor}(n) &= \bigcup_{i=1}^{k-1} max(liz(n), (k-1)(f-1)+i) \\ \overline{ver}(n) &= \bigcup_{i=1}^{k-1} max(lsu(n), (k-1)(i-1)+c) \end{aligned} \tag{5.16}$$

Definición 5.15 (Mutación natural continua). *La mutación natural continua, denotada como $mut(n)$, consiste en cambiar el valor de un gen n por uno de los movimientos generados a partir de él. Formalmente*

$$mut(n) \in \{\{iz(n) \cup de(n) \cup su(n) \cup in(n)\} - \{n\}\} \tag{5.17}$$

Nótese que la mutación natural se reduce al cálculo de una expresión aritmética de bajo coste computacional. Tal y como se ha definido la mutación, la aplicación de la misma obtiene un intervalo donde uno de los límites se ha ampliado o acortado un corte a la derecha o a la izquierda. Podemos generalizar el operador permitiendo que un gen pueda sufrir más de una transición en la misma dirección.

Definición 5.16 (Movimientos de orden m). *Sea n el valor de un gen y k el número de cortes. Se definen los movimientos de orden m como los valores que se pueden alcanzar*

aplicando m transiciones al gen n en la misma dirección.

$$\begin{aligned} iz^m(n) &= max(liz(n), n-m) \\ de^m(n) &= min(lde(n), n+m) \\ su^m(n) &= max(lsu(n), n-mk+m) \\ in^m(n) &= max(liz(n), n+mk-m) \end{aligned} \tag{5.18}$$

Definición 5.17 (Mutación natural de orden m). *Definimos mutación natural de orden m de un gen n, y la denotamos como $mut^m(n)$, como el cambio del valor de n por uno de los movimientos de orden m*

$$mut^m(n) \in \{\{iz^m(n) \cup de^m(n) \cup su^m(n) \cup in^m(n)\} - \{n\}\} \tag{5.19}$$

Si seleccionamos un m al azar, utilizando la mutación de orden m se consigue que uno de los límites del intervalo se desplace hasta cualquiera de los cortes, y no siempre hasta el siguiente o anterior. Por ejemplo, de acuerdo con la tabla de codificación 5.2 y las transiciones de la figura 5.11, el gen $n = 20$ podría mutar hacia arriba hasta los valores 15, 10 y 5 para m igual a 1, 2 y 3 respectivamente.

Al igual que en el caso de los atributos discretos, en ocasiones es conveniente aplicar un operador de mutación que elimine la condición de la regla. En el caso de los atributos continuos, es posible eliminar completamente la condición o simplemente uno de sus límites, llevando éste al extremo del rango. A este tipo de mutación la denominamos *mutación al extremo* o *generalizada*.

Definición 5.18 (Mutación generalizada continua). *Se denomina mutación generalizada continua a aquella que amplía al máximo uno de los límites del intervalo, siendo su expresión*

$$mutg(n) \in \{liz(n) \cup lsu(n)\} \tag{5.20}$$

Para ampliar el intervalo por ambos límites, asignándoles los extremos del rango de valores continuos, el gen toma el valor $k-1$, que es siempre el número natural de la esquina superior derecha de la tabla de codificación.

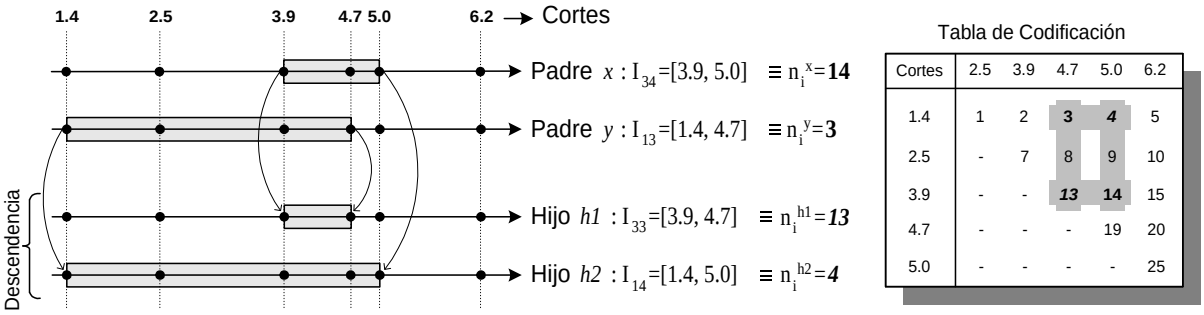


Figura 5.12: Ejemplo de cruce natural continuo.

Cruce

El cruce natural continuo entre dos valores se resuelve calculando el valor más próximo a ambos genes en la tabla de codificación. Si observamos las transiciones de la figura 5.11, el punto más próximo entre dos valores será aquel que exija menor número de movimientos en una misma dirección. Podemos inducir que la descendencia entre dos genes está formada por aquellos genes que se encuentren en la intersección entre la fila y la columna de cada uno de los padres.

Ejemplo 5.6. *Supongamos que queremos cruzar dos individuos, x e y, cuyo i-ésimo atributo es continuo y se codifica según la tabla 5.2. El valor natural de los genes es $n_i^x = 14$ y $n_i^y = 3$, los cuales codifican a los intervalos $I_{34} = [3.9, 5.0]$ y $I_{13} = [1.4, 4.7]$ respectivamente. Los dos subíndices de los intervalos indican la fila y la columna correspondientes. El cruce entre ambos individuos es ilustrado por la figura 5.12. La descendencia de x e y está formada por dos hijos, cuyos genes para el atributo i se han formado cruzando n_i^x y n_i^y. El hijo $h1$ tiene $n_i^{h1} = 13$, que resulta de la intersección entre la fila de x (3) y la columna de y (3), y cuyo intervalo asociado es $I_{33} = [3.9, 4.7]$. Nótese que $h1$ ha heredado el límite inferior de x y el superior de y. Análogamente, el hijo $h2$ toma $n_i^{h2} = 4$ equivalente al intervalo $I_{14} = [1.4, 5.0]$, donde los límites inferior y superior son heredados de y y x respectivamente.*

El método de cruce descrito anteriormente no contempla dos situaciones excepcionales. La primera es que no siempre se producen dos hijos, ya que una de las intersecciones puede resultar por debajo de la diagonal principal de la tabla. En ese caso el descendiente es rechazado al tener un código natural no válido, generando un único

hijo. La otra situación especial se da cuando ambos padres están en la misma fila o columna, es decir, uno de los intervalos está incluido en el otro. En general, si los números se encuentran en la misma fila (esto es $\left\lfloor \frac{n_i^x}{k-1} \right\rfloor = \left\lfloor \frac{n_i^y}{k-1} \right\rfloor$), el mayor valor numérico representa al intervalo mayor; por el contrario, si están en la misma columna (esto es $n_i^x \,\%(k-1) = n_i^y \,\%(k-1)$), es el menor valor numérico el que representa al mayor intervalo. Estos casos, en los que $f(n_i^x) = f(n_i^y)$ o $c(n_i^x) = c(n_i^y)$ según las expresiones de la equación 5.4, la descendencia tomará uno de los valores entre n_i^x y n_i^y, ambos incluidos, situados en la misma fila o la columna que los padres, según el caso.

Una vez descrito de manera informal el cruce natural continuo, veamos la definición del mismo que contempla todos los posibles casos.

Definición 5.19 (Cruce natural continuo). *Sean n_i^x y n_i^y dos genes naturales que codifican al i-ésimo atributo de dos individuos de la población, x e y respectivamente; y n_i^d el gen correspondiente en la descendencia. El cruce natural continuo entre dos valores se define como:*

- *Si ambos genes no comparten la misma fila ni la misma columna, entonces*

$$Cruce(n_i^x, n_i^y) = \{n_i^d \in \{(\overline{hor}(n_i^x) \cap \overline{ver}(n_i^y)) \cup (\overline{hor}(n_i^y) \cap \overline{ver}(n_i^x))\}\} \tag{5.21}$$

- *Si n_i^x y n_i^y, están localizados en la misma fila, entonces*

$$Cruce(n_i^x, n_i^y)\Big|_{f(n_i^x)=f(n_i^y)} = \{n_i^d \in \overline{hor}(n_i^x) |\ min(n_i^x, n_i^y) \leq n_i^d \leq max(n_i^x, n_i^y)\} \tag{5.22}$$

- *Si n_i^x y n_i^y están localizados en la misma columna, entonces*

$$Cruce(n_i^x, n_i^y)\Big|_{c(n_i^x)=c(n_i^y)} = \{n_i^d \in \overline{ver}(n_i^x) |\ min(n_i^x, n_i^y) \leq n_i^d \leq max(n_i^x, n_i^y)\} \tag{5.23}$$

Ejemplo 5.7. *Veamos un ejemplo de cada situación posible para clarificar cómo se resuelve el cruce en cada una de ellas aplicando la definición de cruce natural (5.19) sobre la codificación de la tabla 5.2.*

$$Cruce(14,5) = n_i^d \in \{(\overline{hor}(14) \cap \overline{ver}(5)) \cup (\overline{hor}(5) \cap \overline{ver}(14))\} =$$
$$= \{(\{13,14,15\} \cap \{5,10,15\}) \cup (\{1,2,3,4,5\} \cap \{4,9,14,19\})\} = \{\mathbf{15},\mathbf{4}\}$$

$$Cruce(14,2) = n_i^d \in \{(\overline{hor}(14) \cap \overline{ver}(2)) \cup (\overline{hor}(2) \cap \overline{ver}(14))\} =$$
$$= \{(\{13,14,15\} \cap \{2,7\}) \cup (\{1,2,3,4,5\} \cap \{4,9,14,19\})\} = \{\mathbf{4}\}$$

$$Cruce(5,2)\Big|_{f(5)=f(2)=1} = n_i^d \in \{\overline{hor}(5) \mid 2 \leq n_i^d \leq 5\} = \{\mathbf{2},\mathbf{3},\mathbf{4},\mathbf{5}\}$$

$$Cruce(10,20)\Big|_{c(10)=c(20)=5} = n_i^d \in \{\overline{ver}(10) \mid 10 \leq n_i^d \leq 20\} = \{\mathbf{10},\mathbf{15},\mathbf{20}\}$$

En resumen, todos los operadores genéticos definidos para la codificación natural, ya sean para atributos discretos como continuos, son operaciones algebraicas que realizan transformaciones entre genes naturales sin necesidad de ningún tipo de decodificación. Asimismo, los genes obtenidos tras la aplicación de estos operadores siempre producen códigos válidos con una representación fenotípica coherente.

5.3.4. Evaluación de individuos naturales

Aunque el proceso de evaluación se detalla en secciones posteriores, es interesante exponer aquí cómo se evalúan los individuos codificados con codificación natural.

La evaluación consiste en medir y cuantificar la calidad o bondad de cada individuo de la población. Dicha bondad es usada en el proceso de selección como medida comparativa para establecer cuáles son los mejores individuos. Generalmente, cuando se aplican técnicas evolutivas en tareas de clasificación, la función de evaluación se basa, entre otros parámetros, en el número de ejemplos cubiertos por la regla, así como en los aciertos y errores que ésta comete al clasificarlos. Una regla cubre a un ejemplo cuando éste satisface todas las condiciones establecidas en el antecedente. Si además, el ejemplo está etiquetado con la misma clase que la regla, entonces decimos que lo clasifica correctamente, contabilizando un acierto. Si por el contrario, la clase no coincide, decimos que el ejemplo ha sido clasificado incorrectamente y se contabiliza como un error.

Cuando las reglas son codificadas usando codificación híbrida, la clasificación de

un ejemplo es directa, ya que simplemente consiste en comprobar que los valores de los atributos de éste pertenecen a los intervalos o conjuntos de valores que cada condición indica y cotejar la clase. Éste es el proceso que sigue COGITO. Sin embargo, en el caso de HIDER donde los individuos son naturales, este proceso no es tan evidente. Dado que la codificación natural de la clase es una simple enumeración de las etiquetas, comprobar si la regla acierta o no una vez cubierto un ejemplo es inmediato. Pero, ¿cuándo un individuo natural cubre a un ejemplo? La solución trivial consistiría en decodificar cada gen para obtener el intervalo o conjunto de valores correspondiente a cada condición y verificar su cumplimiento. No obstante, aunque esta conversión es de bajo coste computacional, la gran cantidad de evaluaciones que se realizan durante la ejecución del algoritmo evolutivo hace que tal coste sí sea significativo globalmente.

La solución que evita la decodificación de los individuos pasa por trasformar también los ejemplos según la codificación natural, representando cada atributo como los genes de un individuo. Si el atributo es discreto, el atributo codificado toma el número natural correspondiente al código binario donde el bit que representa dicho valor es 1 y el resto 0 (véase tabla 5.1). Esta conversión es directa y viene dada por la expresión $2^{|A|-p-1}$, donde $|A|$ es el número de valores distintos y p es la posición del valor en el conjunto ordenado de valores, empezando en 0. Por ejemplo, para el atributo discreto del ejemplo 5.1, donde el conjunto de valores posibles es $A = \{blanco, rojo, verde, azul, negro\}$, la codificación del color $rojo$ es $2^{5-1-1} = 8\,(01000)$. Si el atributo es continuo, su valor natural será igual al código natural del mínimo intervalo que lo contenga en la tabla de codificación correspondiente. Por ejemplo, si pretendemos codificar el valor 2.6 para el atributo de la tabla 5.2, el atributo codificado tomará el valor 7 correspondiente al intervalo $[2.5, 3.9]$. Nótese que los intervalos mínimos son aquellos situados en la diagonal principal de la tabla de codificación. Por último, la clase del ejemplo se codifica igual que si fuera una regla, es decir, enumerando las etiquetas.

Como veremos a continuación, una vez transformados todos atributos, comprobar si un individuo cubre al ejemplo codificado es inmediato. Esta transformación no implica pérdida de información desde el punto de vista del aprendizaje en HIDER, ya que éste siempre genera reglas donde los límites de los intervalos para atributos continuos

pertenecen a un conjunto de cortes fijo, independientemente de dónde se sitúen los valores de los ejemplos para tales atributos. Teniendo en cuenta esto, y que no se añaden nuevos ejemplos durante la ejecución del algoritmo evolutivo, el conjunto de datos es codificado completamente al inicio, soslayando el original y usando los nuevos ejemplos durante todo proceso de aprendizaje.

Una vez transformado el conjunto de datos, dado un individuo $r = (r_1, r_2, \ldots, r_m | C_r)$ y un ejemplo $e = (e_1, e_2, \ldots, e_m | C_e)$, entonces r cubre a e ($cubre(r, e)$) según la expresión 5.24.

$$cubre(r, e) \Leftrightarrow \forall k : 1 \leq k \leq |atributos| \bullet cub(r_k, e_k) \tag{5.24}$$

$$cub(r_k, e_k) = \begin{cases} f(e_k) \leq f(e_k) \wedge c(r_k) \geq c(e_k) & si\ el\ a_k\ es\ continuo \\ r_k \,\&\, e_k \neq 0 & si\ el\ a_k\ es\ discreto \end{cases}$$

donde r_k es el gen que representa la k-ésima condición de r; e_k es la codificación del k-ésimo atributo de e; $f(\cdot)$ y $c(\cdot)$ son la fila y la columna respectivamente, calculadas según las expresiones de la ecuación 5.4; y $\&$ es la conjunción binaria. Así, podemos afirmar que:

$$\begin{aligned} Si\ \neg cubre(r, e) &\Rightarrow r\ no\ clasifica\ a\ e \\ Si\ cubre(r, e) \wedge C_r = C_e &\Rightarrow r\ clasifica\ correctamente\ a\ e\ (acierto) \\ Si\ cubre(r, e) \wedge C_r \neq C_e &\Rightarrow r\ clasifica\ incorrectamente\ a\ e\ (error) \end{aligned}$$

Ejemplo 5.8. *Siguiendo los ejemplos anteriormente usados, donde un atributo continuo (a_1) y otro discreto (a_2) son codificados según las tablas 5.2 y 5.1 respectivamente, la regla "Si $a_1 \in [2.5, 5.0]$ Y $a_2 \in \{rojo, azul, negro\}$ Entonces $Clase = A$" se codifica como el individuo $r = (9, 11 \mid 0)$. Con estos genes, el individuo r cubrirá a todos aquellos ejemplos codificados cuyo primer atributo sea 7, 13 ó 19, y cuyo segundo atributo tome 1, 2 u 8. La figura 5.13 ilustra este ejemplo.*

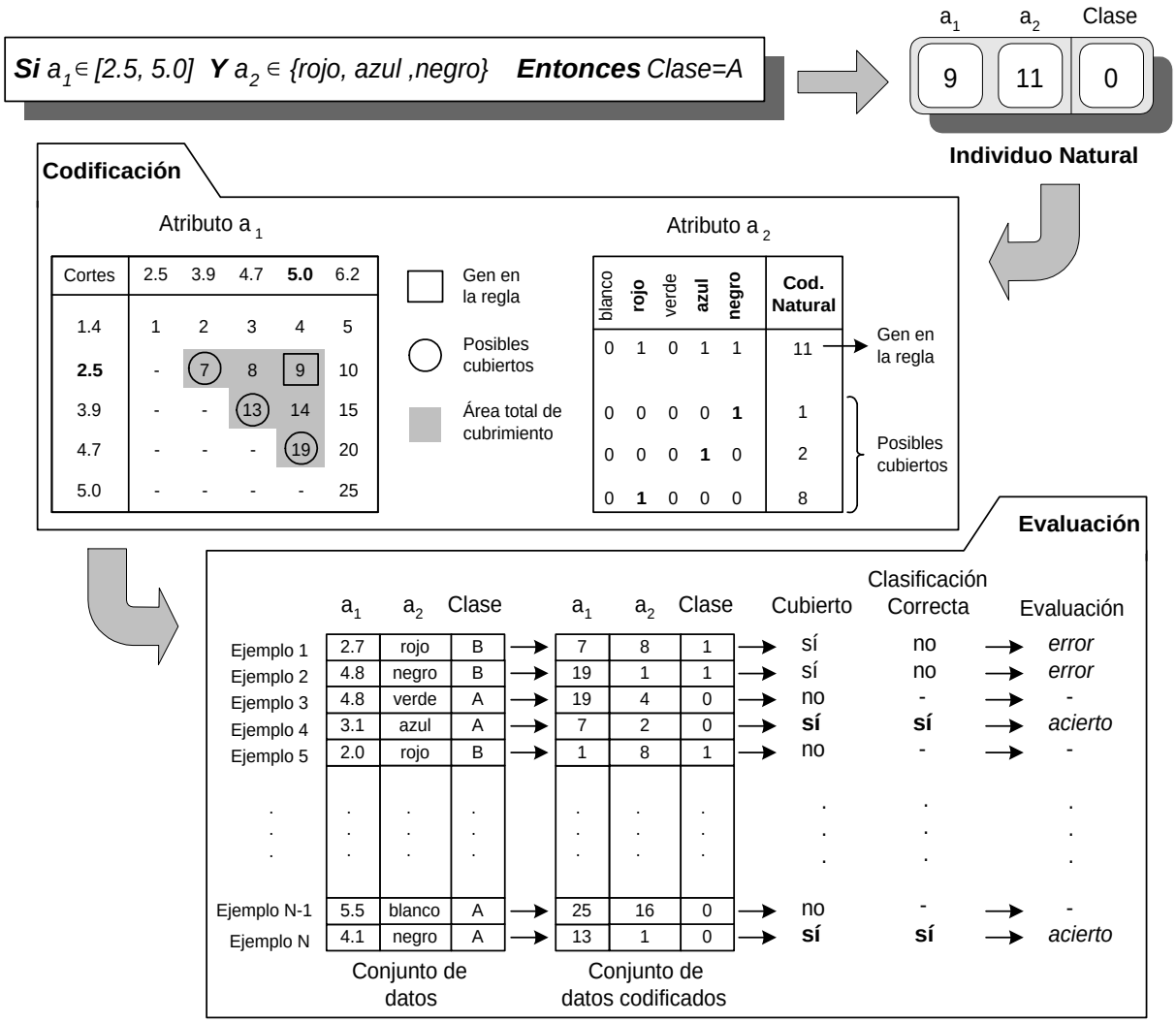


Figura 5.13: Ejemplo de cubrimiento.

5.4. Estructura de Evaluación Eficiente

La evaluación de los individuos de la población genética es, junto a la codificación, uno de los factores críticos en la aplicación de algoritmos evolutivos. En el caso concreto del aprendizaje supervisado de reglas, la evaluación influye notablemente no sólo en la calidad del modelo de conocimiento generado, sino también en el coste computacional del algoritmo.

Como se ha mencionado anteriormente, la función de evaluación cuantifica la bondad de cada individuo de la población a partir del número de aciertos y errores que

éstos cometen al clasificar los ejemplos del conjunto de datos. Para contabilizar esos aciertos y errores, el método usado tradicionalmente por los algoritmos de aprendizaje evolutivo, incluido COGITO, consiste en realizar un recorrido lineal de los datos de entrenamiento para cada individuo de la población. Como describimos en la sección 3.3.3 (véase página 81), este proceso de evaluación es muy sencillo de aplicar pero altamente costo, en concreto $\Theta(GPNm)$, donde G es el número de generaciones, P es el tamaño de la población, N es el número de ejemplos del conjunto de datos y m el número de atributos de éstos. Algunos autores [148, 162] han concentrado su esfuerzo en mejorar el proceso de aprendizaje. Otros han abordado el problema desde la perspectiva de la escalabilidad [167]. Sin embargo, la organización apropiada de la información podría contribuir también a la reducción del coste computacional. Este aspecto, no menos importante que los anteriores, ha sido quizá más descuidado.

Ejemplo 5.9. *Un algoritmo evolutivo que lleve a cabo 300 generaciones, con 100 individuos por población, necesita ejecutar al menos 30.000 evaluaciones. Si tal algoritmo es aplicado sobre un conjunto de datos con 1000 ejemplos y 10 atributos, éste haría un total de* 3×10^8 *comprobaciones, de las cuales muchas podrían evitarse.*

Si analizamos los términos que intervienen en el coste de evaluación ($\Theta(GPNm)$), habitualmente N es significativamente mayor que los otros tres, sobre todo en tareas de minería de datos donde el número de ejemplos a procesar es muy grande. Este término lo aporta el recorrido lineal de los N ejemplos del conjunto de datos. Sin embargo, para contabilizar los aciertos y los errores que un individuo comete, bastaría con procesar sólo aquellos ejemplos que son cubiertos por la regla, es decir, clasificados correcta o incorrectamente, y no todo el conjunto de datos. Esta idea nos induce a pensar que es posible reducir el coste de evaluación individual si los ejemplos son distribuidos dependiendo del valor que toman sus atributos. Así, se podrían discriminar los que no cumplen las condiciones de una regla sin tener que procesarlos.

Existen en la literatura numerosas propuestas sobre estructuras de datos y organización de los mismos que esencialmente aceleran la búsqueda de información en espacios multidimensionales, como los denominados MAM (*Multidimensional Access Methods*)[67]: *Point Access Methods* (Grid File [133], KDB-tree [161], LSD-tree [87] , BV-tree [65], etc) y *Spatial Access Methods* (K-D-Tree [20], R-tree [84], P-tree [97], SDK-tree

[134], etc). No obstante, dada la particularidad del problema que abordamos, los MAM no aportan, por sí solos, una solución a dicho problema. Habría que adaptar entonces alguno de estos métodos a la evaluación de reglas de decisión, es decir, deberíamos modificar los métodos de construcción y utilización de alguna de estas estructuras para que facilitara el modo de acceder a la información que viene dado por la semántica de una regla de decisión. Esa modificación supondría el coste añadido de aplicar una estructura de datos para dar solución a un problema para el cual no fue diseñada. Por ejemplo, el uso de un AD-Tree [128] podría parecer apropiada para dar solución al problema planteado. Sin embargo, si queremos evaluar una regla de decisión cualquiera usando esta estructura, tendríamos que construir un AD-Tree de máxima densidad (no esparcido) para contemplar todas las posibles consultas además de los índices de los ejemplos que cada consulta cumple. Esto supondría un alto coste computacional en términos de espacio, debido fundamentalmente a gran cantidad de información redundante.

Dado el problema del alto coste de evaluación, y teniendo en cuenta que las estructuras existentes no son suficientemente adecuadas, desarrollamos la *Estructura de Evaluación Eficiente* [76, 78], en adelante EES (*Efficient Evaluation Structure*), la cual acelera del proceso de evaluación de reglas de decisión durante la aplicación del algoritmo evolutivo. Esta estructura indexa el conjunto de datos de forma que se aprovecha la semántica de las reglas de decisión para discriminar aquellos ejemplos que no son cubiertos por las mismas. De este modo, sólo son procesados los ejemplos estrictamente necesarios.

Aunque la EES ha sido ideada y diseñada para su uso en algoritmos evolutivos, en general, es aplicable a cualquier sistema que se base en una búsqueda probabilística en el espacio y una posterior evaluación de las hipotéticas soluciones [75]. No obstante, la implementación concreta de la estructura está estrechamente relacionada con la codificación que el algoritmo evolutivo utilice. En este sentido, se han desarrollado dos implementaciones de la EES: la denominada EES-H, diseñada para la codificación híbrida y aplicada a COGITO; y la EES-N, desarrollada para la codificación natural e integrada en HIDER. Aunque ambas estructuras difieren sólo en aspectos de implementación, la EES-N aprovecha ciertas características de la codificación natural que la

hacen más eficiente frente a la EES-H, aunque el proceso de evaluación sea el mismo. A pesar de ello, la implementación híbrida resulta más intuitiva y fácil de interpretar que la natural, por lo que la descripción general de la estructura, su construcción, así como el método de evaluación serán descritos usando la EES-H, para más tarde extrapolar todos estos aspectos a la EES-N.

5.4.1. EES Híbrida

Partiendo de un conjunto de datos etiquetados, EES indexa dicho conjunto de datos y distribuye los índices de forma que sea posible realizar un recorrido por atributos en lugar de hacerlo por ejemplos. El objetivo de la estructura se puede resumir en que dadas las condiciones establecidas por una regla, el recorrido de la EES devuelva los índices de los ejemplos que cumplen tales condiciones para luego evaluar la regla sólo con esos ejemplos y no con todos.

La estructura de datos debe ser capaz de almacenar esta información con independencia del tipo de atributo (continuo o discreto). Para atributos continuos, es conveniente aplicar algún método de discretización que disminuya la cardinalidad del conjunto de valores que este tipo de atributos puede llegar a tomar. Si el método de generación de reglas emplea algún tipo de discretización concreta para transformar los valores continuos en intervalos, EES debe ser construida a partir de esos mismos intervalos. Éste es el caso de HIDER, que usa el método USD para poder aplicar la codificación natural. Esto no supone limitación alguna por parte de la estructura de datos, ya que ésta es totalmente flexible frente a la discretización usada. La única restricción que se exige al método de discretización es que los intervalos generados sean disjuntos. En cualquier caso, podríamos mantener el rango teóricamente infinito y la estructura seguiría siendo válida.

En general, para cada atributo a_i denotaremos por Ω_i al conjunto finito de valores que a_i puede tomar. En el caso de que a_i sea un atributo discreto, Ω_i contendrá valores que representaremos como V_{ij} (con $1 \leq j \leq |\Omega_i|$). Por el contrario, si se trata de un atributo continuo, Ω_i contendrá intervalos que llamaremos I_{ij} (con $1 \leq j \leq |\Omega_i|$), cuyos límites inferior y superior denotaremos por li_{ij} y ls_{ij} respectivamente. De este modo, EES almacenará la información de los atributos continuos de forma similar a como lo

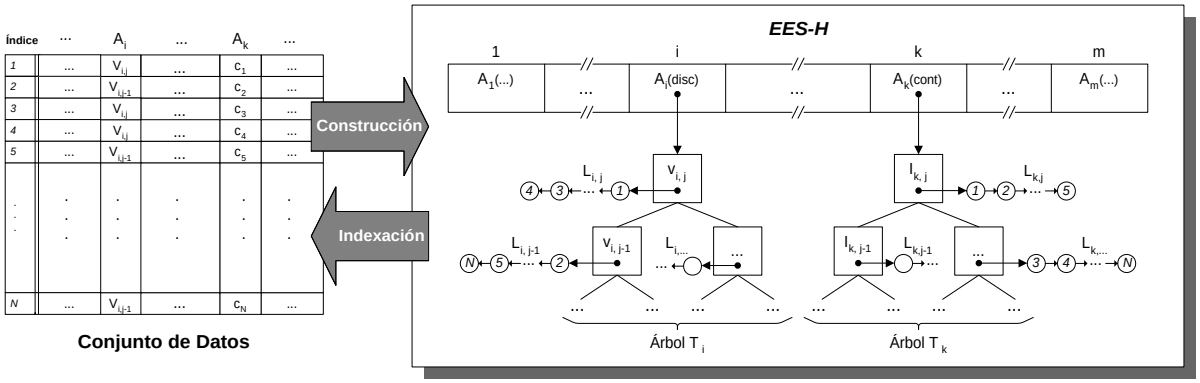


Figura 5.14: Esquema general del la estructura EES-H.

hace para los discretos: se guardarán los límites inferior y superior de cada intervalo, en lugar de los valores individuales que son almacenados para los atributos discretos. La figura 5.14 muestra el esquema general de la estructura de datos para un conjunto con m atributos.

Básicamente, EES-H es un vector de árboles de búsqueda binarios y balanceados, de forma que el i–ésimo elemento del vector contendrá información sobre el i–ésimo atributo (a_i) del conjunto de datos. En concreto, se almacenan los diferentes valores o intervalos que a_i puede tomar en el árbol, que denotaremos por T_i. Por simplicidad, la Figura 5.14 presenta únicamente dos de los m árboles que contiene la estructura. Si el atributo a_i es continuo, se almacenarán los límites del intervalo I_{ij} correspondiente (li_{ij} y ls_{ij}). En caso de que sea discreto, se almacenará el valor V_{ij}. Además de V_{ij} o I_{ij}, cada nodo N_{ij} del árbol T_i contiene una lista (L_{ij}) de índices que indican las posiciones de los ejemplos dentro del conjunto de datos. Si a_i es discreto, los índices contenidos en la lista L_{ij} corresponderán a los ejemplos cuyo i–ésimo atributo toma el j–ésimo valor (V_{ij}) dentro del conjunto Ω_i de posibles valores de dicho atributo. Si por el contrario a_i es continuo, los índices contenidos en L_{ij} corresponderán a aquellos ejemplos cuyo valor para el i–ésimo atributo está incluido en el j–ésimo intervalo (I_{ij}) dentro del conjunto Ω_i de posibles intervalos de dicho atributo. Es importante señalar que los árboles se encuentran ordenados, de forma que cualquier búsqueda sobre ellos tiene coste logarítmico[4].

[4]Este coste es relativo a la EES-H. La versión natural EES-N elimina este coste al no tener que realizar búsquedas debido a que la codificación natural posibilita el acceso directo a los nodos.

Una vez descritas las características generales de la estructura, vamos a detallar los dos aspectos fundamentales de la misma: cómo se construye y cómo se usa una vez construida para la evaluación de potenciales reglas de decisión.

Construcción

Inicialmente tenemos un conjunto etiquetado de datos con N ejemplos indexados de 1 a N, cada uno de ellos con m atributos de cualquier tipo, entre los cuales no incluimos la clase. Para cada atributo a_i, se crea un árbol de búsqueda balanceado cuyos nodos contienen los valores V_{ij} o intervalos I_{ij}, según sea el atributo discreto o continuo respectivamente. Cuando un árbol T_i ha sido completado, éste es insertado en el vector de árboles en la posición correspondiente (i), pasándose a tratar el siguiente atributo (a_{i+1}).

Una vez que todos los árboles han sido creados e insertados, se pasa a completar las listas de cada nodo. Para ello, se realiza un recorrido lineal del conjunto de datos, pasando por todos los ejemplos. Durante el tratamiento de cada ejemplo se almacenará su índice o posición en la lista correspondiente de cada uno de los árboles, dependiendo del valor de los atributos. Para cada atributo a_i de un ejemplo concreto, se busca en el árbol T_i el nodo correspondiente al valor de dicho atributo. Si el atributo es discreto, el valor deberá coincidir con el valor del nodo. En caso de que sea continuo, el nodo correspondiente al valor del atributo será aquél cuyo intervalo contenga dicho valor. Ahora podemos entender por qué es necesario que el método de discretización genere intervalos disjuntos, puesto que si no fuera así, podría existir algún valor en el conjunto de datos perteneciente a varios intervalos y por tanto corresponderle varios nodos del árbol. Una vez que el nodo es localizado, se inserta en la lista de dicho nodo el índice del ejemplo que se esté tratando, pasando a procesarse el siguiente atributo de ese ejemplo. Cuando todos los atributos de un ejemplo han sido tratados, decimos que el ejemplo ha sido insertado en la estructura, y pasamos a procesar el siguiente ejemplo.

En el momento en el que todos los ejemplos han sido insertados, la estructura contiene la misma información que el conjunto de datos, exceptuando la clase de cada ejemplo. Sin embargo, esta información es accesible directamente durante el uso de la

estructura, ya que ésta almacena los índices de los ejemplos. El coste computacional del proceso de construcción de la estructura de datos es $\Theta(NMlog_2|\overline{\Omega}|)$, donde $\overline{\Omega}$ es el número medio de nodos de los árboles.

Evaluación de reglas usando EES

El principal objetivo que buscamos con la utilización de la estructura de datos es no tener que procesar aquellos ejemplos cuyos valores no son cubiertos por la regla que se está evaluando. Cada nodo N_{ij} del árbol contiene en la lista L_{ij} los índices de aquellos ejemplos que satisfacen la condición $a_i \in I_{ij}$ o $a_i \in V_{ij}$, según el caso. Si tomamos un nodo N_{ij} de cada árbol T_i, la intersección de las listas L_{ij} será el conjunto de los índices de ejemplos que satisfacen que cada uno de sus atributos a_i toman valores que son cubiertos por cada nodo N_{ij} correspondiente. La ventaja que ofrece la estructura radica en que las intersecciones se realizan de forma incremental, es decir, primero se realiza la intersección de la lista para el atributo a_1 y la lista para el atributo a_2. Si dicha intersección no es vacía, se busca la lista para el atributo a_3 y se realiza una nueva intersección entre esta lista y el resultado de la intersección anterior. Este proceso se repite hasta completar todos los atributos o bien hasta que una de las intersecciones resulte vacía. Si el proceso se completa, la lista resultante contendrá los índices de los ejemplos que cumplen las condiciones correspondientes a los nodos N_{ij} elegidos. Si estos nodos N_{ij} son buscados según las condiciones establecidas por una regla de decisión, estaremos evaluando dicha regla, siendo la lista final el conjunto de índices de ejemplos cubiertos. La figura 5.15 muestra el algoritmo de evaluación de reglas de decisión usando la estructura EES.

Un aspecto importante a tener en cuenta a la hora de usar la estructura EES es el hecho de que las condiciones establecidas por una regla pueden incluir varios nodos del árbol correspondiente, es decir, varios intervalos consecutivos o varios valores discretos, según el tipo de atributo. Esto implica que la intersección que se realiza en cada paso se lleva a cabo entre la lista acumulada de intersecciones anteriores y la unión de las listas de los nodos cubiertos por la condición correspondiente a la iteración en curso. En la Figura 5.15, esta unión la lleva a cabo la función $UnionDeListas$, actuando de forma diferente según el tipo de atributo al que afecte la condición. Si el atributo

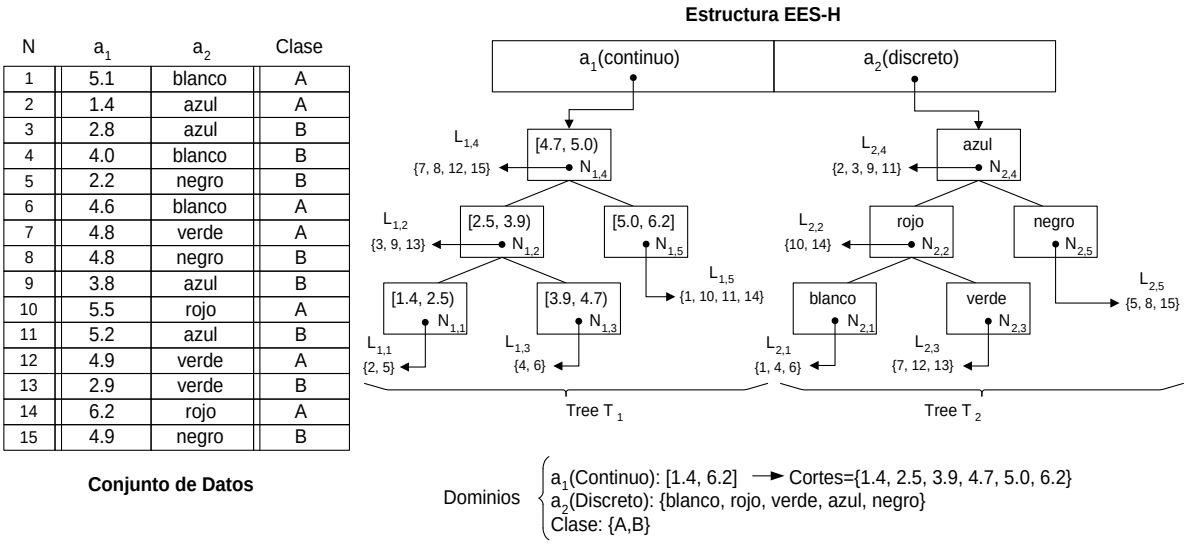
Función Evaluar(R,E,L)
Entrada: R: Regla de Decisión; E: EES
Salida: L: Lista de Índices (ejemplos cubiertos por R)
Comienzo
$i := 1$
$L_i := UnionDeListas(R, E, i)$
Mientras $i < NumeroDeAtributos(E) \wedge L_i \neq \emptyset$
$i := i + 1$
$L_i := L_{i-1} \bigcap UnionDeListas(R, E, i)$
Fin Mientras
$L := L_i$
Fin Evaluar

Function UnionDeListas(R,E,k)
Entrada: R: Regla de Decisión; E: EES
k: Entero (posición del atributo)
Salida: Lu: Lista de Índices (ejemplos cubiertos por R_k)
Comienzo
$Lu := \emptyset$
$T_k := E[k]$ (árbol de la posición k)
Si a_k es Continuo
$\forall\, j : \{I_{kj} \in T_k \wedge I_{kj} \subseteq R_k\} \bullet Lu := Lu \bigcup L_{kj}$
Si no (a_k es Discreto)
$\forall\, j : \{V_{kj} \in T_k \wedge V_{kj} \in R_k\} \bullet Lu := Lu \bigcup L_{kj}$
Fin Si
Fin UnionDeListas

Figura 5.15: Algoritmo de evaluación usando EES.

es continuo, los intervalos que intervienen en la condición son siempre consecutivos, por lo que se busca en el árbol el primer nodo que está incluido en la condición que la regla establece, y se realiza un recorrido en *inorden* del árbol partiendo desde dicho nodo, deteniéndose al encontrar el primer nodo que no es cubierto por la condición. En el caso de atributos discretos, es necesario buscar todos los valores que la regla establece para dicho atributo en el árbol correspondiente, ya que éstos no tiene por qué ser consecutivos.

Es fácil colegir que, usando la estructura EES, sólo son tratados los ejemplos que van siendo cubiertos por las condiciones de regla en evaluación, puesto que en cada



N	a_1	a_2	Clase
1	5.1	blanco	A
2	1.4	azul	A
3	2.8	azul	B
4	4.0	blanco	B
5	2.2	negro	B
6	4.6	blanco	A
7	4.8	verde	A
8	4.8	negro	B
9	3.8	azul	B
10	5.5	rojo	A
11	5.2	azul	B
12	4.9	verde	A
13	2.9	verde	B
14	6.2	rojo	A
15	4.9	negro	B

Conjunto de Datos

Figura 5.16: Ejemplo de EES-H.

iteración, el numero de índices contenidos en las listas se va decrementando o, a lo sumo, se queda igual. Si en alguna iteración intermedia, la intersección resulta vacía, el proceso se detiene, puesto que esa regla ya no podrá cubrir a ningún ejemplo. Si esa misma regla hubiera sido evaluada mediante el recorrido lineal del conjunto de datos, éste hubiera sido tratado completamente aunque la regla no cubriera ningún ejemplo. La existencia de reglas que no cubren ningún ejemplo del conjunto de datos es relativamente frecuente durante el proceso evolutivo, sobre todo en las primeras generaciones, por lo que el hecho de detener la evaluación en una iteración intermedia resulta altamente ventajoso.

Ejemplo 5.10. *La figura 5.16 muestra el caso de un conjunto de datos con 15 ejemplos, 2 atributos y 2 etiquetas de clase. El primer atributo (a_1) es continuo, por lo que se aplica el algoritmo de discretización y éste devuelve un conjunto de 6 cortes que delimitan 5 intervalos disjuntos. Por tanto, árbol T_1 tiene 5 nodos (N_{1j}, con $1 \leq j \leq 5$), uno por intervalo. El segundo atributo (a_2) es discreto y puede tomar 5 valores distintos, lo que implica que el árbol T_2 también tiene 5 nodos (N_{2j}, con $1 \leq j \leq 5$). Con estos datos y una vez construida la estructura, la figura 5.17 ilustra la evaluación de dos reglas usando la estructura EES-H anterior.*

Regla R1: Si $a_1 \in [4.7, 6.2)$ Y $a_2 \in \{verde, azul\}$ Entonces $Clase = A$
↪ Recorrido de la EES:
$a_1 \in [4.7, 6.2) \Rightarrow L_1 = L_{1,4} \cup L_{1,5} = \{1, 7, 8, 10, 11, 12, 14, 15\}$
$a_2 \in \{verde, azul\} \Rightarrow L_2 = L_{2,3} \cup L_{2,4} = \{2, 3, 7, 9, 11, 12, 13\}$
↪ $L = L_1 \cap L_2 = \{7, 11, 12\}$
↪ Evaluación:
$\{7, 12\} \rightharpoonup Clase = A \Rightarrow Aciertos = 2$
$\{11\} \rightharpoonup Clase = B \Rightarrow Errores = 1$

Regla R2: Si $a_1 \in [2.5, 3.9)$ Y $a_2 \in \{rojo, negro\}$ Entonces $Clase = B$
↪ Recorrido de la EES:
$a_1 \in [2.5, 3.9) \Rightarrow L_1 = L_{1,2} = \{3, 9, 13\}$
$a_2 \in \{rojo, negro\} \Rightarrow L_2 = L_{2,2} \cup L_{2,5} = \{5, 8, 10, 14, 15\}$
↪ $L = L_1 \cap L_2 = \emptyset$
↪ Evaluación: R2 no cubre ningún ejemplo

Figura 5.17: Ejemplo de evaluación de reglas mediante la EES-H de la figura 5.16

Como podemos observar, la regla R1 cubre 8 ejemplos con la primera condición y 7 con la segunda, de los cuales sólo 3 (7, 11 y 12) son comunes a ambas condiciones. Con estos índices, se consulta en el conjunto de datos la clase de cada uno y se compara con la de la regla, dando como resultado 2 aciertos y 1 error. El uso de la EES ha permitido discriminar los ejemplos no cubiertos sin tener que procesarlos. Respecto a la regla R2, ésta no cubre ningún ejemplo, dando la lista final vacía. Nótese que si el conjunto de datos tuviera más atributos, la evaluación de R2 hubiera concluido igualmente con las dos primeras condiciones, sabiendo en este punto que no podrá cubrir ningún ejemplo y deteniendo el proceso. La evaluación lineal recorre el conjunto de datos tanto si la regla cubre como si no cubre algún ejemplo, lo que pone de manifiesto las ventajas de usar la EES frente al recorrido secuencial tradicional.

5.4.2. EES Natural

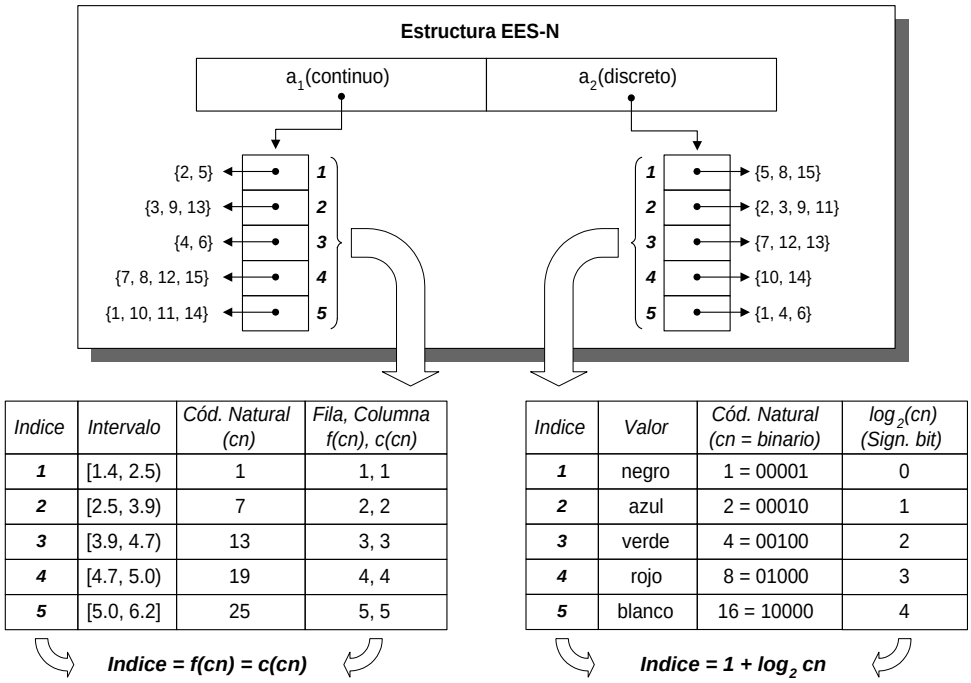
En general, el planteamiento de la estructura EES para acelerar la evaluación de los individuos es independiente de la codificación que el algoritmo evolutivo utilice.

No obstante, una implementación adaptada a la codificación puede mejorar la eficiencia del recorrido de la estructura, haciendo aún más rápido el proceso de evaluación. En este sentido, hemos desarrollado una versión de la EES orientada a la codificación natural denominada EES-N. Cuando una determinada condición en la regla incluía varios nodos del árbol correspondiente en la EES-H, era necesario realizar una búsqueda de tales nodos para llevar a cabo la unión de las listas. El coste computacional de esta búsqueda es de orden logarítmico, ya que se trataba de árboles binarios ordenados y balanceados. Sin embargo, la variante EES-N aprovecha ciertas características de la codificación natural para que el acceso a las listas sea directo, eliminando así el coste de la búsqueda de nodos.

La EES-N no distribuye las listas de índices en árboles sino en vectores ordenados. Si el atributo es continuo, cada celda del vector representa un intervalo de mínima anchura, es decir, los representados en la diagonal principal de la tabla de codificación. Nótese que los números naturales correspondientes a estos intervalos mínimos son los únicos que un ejemplo codificado[5] puede tomar. Así, cada celda contiene la lista de índices de aquellos ejemplos codificados cuyo código natural sea igual al que la celda representa. Para los atributos discretos, la idea es la misma, aunque con valores en vez de intervalos. En ambos casos, los vectores se encuentran ordenados ascendentemente por el código natural correspondiente.

La figura 5.18 muestra la EES-N que representa el mismo ejemplo que la EES-H de la figura 5.16, así como dos tablas con la interpretación de cada vector, aunque esta información no es necesario almacenarla. Como podemos observar, lo único que almacenan los vectores son las listas, ya que existe una relación directa entre el código natural y el índice de cada lista en los vectores. Esto hace que, dado un individuo, sea posible acceder directamente a las celdas que cubre sin necesidad de buscarlas. En el caso de los atributos continuos, el índice en el vector coincide con la fila y la columna del código natural, siendo $Indice = f(cn) = c(cn)$. Tanto la fila como la columna pueden ser calculadas directamente mediante las expresiones de la ecuación 5.4. Para los discretos, el índice en el vector coincide con la posición del bit activo en representación binaria, empezando por el menos significativo, siendo pues $Indice = 1 + log_2\ cn$.

[5]En la sección 5.3.4, se detalla cómo los ejemplos también son codificados para hacer más eficiente la evaluación de individuos naturales.



Indice	Intervalo	Cód. Natural (cn)	Fila, Columna f(cn), c(cn)
1	[1.4, 2.5)	1	1, 1
2	[2.5, 3.9)	7	2, 2
3	[3.9, 4.7)	13	3, 3
4	[4.7, 5.0)	19	4, 4
5	[5.0, 6.2]	25	5, 5

Indice = f(cn) = c(cn)

Indice	Valor	Cód. Natural (cn = binario)	log_2(cn) (Sign. bit)
1	negro	1 = 00001	0
2	azul	2 = 00010	1
3	verde	4 = 00100	2
4	rojo	8 = 01000	3
5	blanco	16 = 10000	4

Indice = 1 + log_2 cn

Figura 5.18: Ejemplo de EES-N.

Por tanto, la EES-N evita la búsqueda de las celdas (nodos en la EES-H) para realizar la unión de las listas, como muestran los algoritmos de la figura 5.19.

5.4.3. Experimentos

Con independencia de las pruebas hechas para la herramienta HIDER completa, se han llevado a cabo experimentos diseñados ex profeso para probar empíricamente la eficiencia de la estructura EES frente al recorrido lineal habitualmente usado. En concreto, los experimentos han consistido en la evaluación de varios conjuntos de reglas de decisión para 15 bases de datos diferentes del UCI Repository [24]. Para cada una de ellas, han sido generados grupos de reglas de forma aleatoria mediante un método que asegura la distribución uniforme de éstas. Posteriormente, dichas reglas fueron evaluadas usando el método lineal y la estructura EES, comparándose los resultados obtenidos.

Para cada regla, el método de evaluación lineal recorre la base de datos, que previamente ha sido almacenada en una tabla, tratando todos y cada uno de los ejemplos de

Función UnionDeListas_Continuos(v,n,k)
Entrada: v: vector de listas; n: Gen Natural; k: Número de cortes
Salida: Lu: Lista de Índices
Comienzo
$f := \lfloor \frac{n-1}{k-1} \rfloor + 1$ (fila de n)
$c := (n-1)\,\%(k-1) + 1$ (columna de n)
$Lu := \emptyset$
Desde $i = f$ **hasta** c
$Lu := Lu \cup v[i]$
Fin Desde
Fin UnionDeListas_Continuos

Función UnionDeListas_Discretos(v,n)
Entrada: v: vector de listas; n: Gen Natural
Salida: Lu: Lista de Índices
Comienzo
$val := n$
$Lu := \emptyset$
Mientras $val \neq 0$
$mp := \lfloor log_2\ val \rfloor$ (mayor potencia de 2 = posición del bit más sig. a 1)
$Lu := Lu \cup v[val]$
$val := val - 2^{mp}$
Fin Mientras
Fin UnionDeListas_Discretos

Figura 5.19: Unión de listas usando EES-N.

ésta. Asimismo, la comprobación para cada ejemplo de que los valores de sus atributos cumplen las condiciones de la regla se realiza igualmente de forma lineal. Sin embargo, no siempre es necesario comprobar todos los atributos. Si durante el tratamiento de un determinado ejemplo, uno de sus valores no cumple la condición que la regla establece para el atributo correspondiente, ese ejemplo deja de procesarse, puesto que ya no podrá cumplir la regla independientemente de los valores que tomen el resto de atributos, pasándose a tratar el siguiente ejemplo.

Dado que la estructura EES presenta mayor ventaja en la evaluación de reglas que no cubren ejemplos, quisimos comparar en qué medida este tipo de reglas influía en la diferencia de eficiencia entre la estructura y el método lineal. Por ello se generaron y evaluaron dos tipos de reglas: *válidas* y *no válidas*. Llamamos regla válida a aquella

regla que al menos cubre un ejemplo de la base de datos. En contraposición, decimos que una regla es no válida cuando no cubre ningún ejemplo de la base de datos. Según estas definiciones, una regla válida hará que la estructura sea recorrida completamente, mientras que para una regla no válida el proceso de evaluación en la estructura de datos se detendrá antes de recorrer ésta completamente.

Como se ha mencionado con anterioridad, las pruebas realizadas han consistido en la generación de diferentes conjuntos de reglas aleatorias, en concreto 11 grupos de 1000 reglas para cada base de datos. Cada uno de estos grupos tiene un porcentaje de reglas válidas diferente que varía uniformemente desde el 0 % al 100 %, es decir, en el primer conjunto todas la reglas serán no válidas, en el segundo habrá un 10 % de válidas y un 90 % de reglas no válidas, y así sucesivamente hasta el undécimo grupo que contendrá el 100 % de reglas válidas. Por tanto, se han generado grupos de reglas estratificados para cada base de datos, estudiándose la relación entre el tiempo de evaluación y el porcentaje de reglas válidas. En principio, la evaluación de aquellos conjuntos de reglas donde la proporción de reglas válidas es del 100 % resulta menos ventajosa para la estructura de datos, puesto que la evaluación de cada una de ellas recorrerá la estructura completamente. Por el contrario, la evaluación de aquellos conjuntos donde todas las reglas son no válidas, el proceso se detendrá antes de haber recorrido la estructura EES, lo que hará que el procesamiento sea aún más rápido. Fácilmente podemos intuir que las pruebas con mayor proporción de reglas no válidas hará que la evaluación lineal sea también más rápida, puesto que, como se expuso anteriormente, el recorrido lineal también fue optimizado.

La Tabla 5.4 compara los resultados obtenidos tras los experimentos. Para cada base de datos (primera columna), se muestran el tiempo medio de evaluación empleado por la estructura EES (segunda columna) y por el método lineal (tercera columna), así como la mejora que EES obtiene frente a dicha evaluación lineal (última columna). Tal mejora viene dada por la ecuación 5.25, que representa el porcentaje de tiempo que ahorra la estructura EES respecto al tiempo empleado por la evaluación directa sobre el conjunto de datos.

$$Mejora = 100 \times \frac{\overline{T}(Eval.\,Lineal) - \overline{T}(Eval.\,EES)}{\overline{T}(Eval.\,Lineal)} \tag{5.25}$$

Base de Datos	$\overline{T}$ (Eval. EES)	$\overline{T}$ (Eval. Lineal)	Mejora(%)
bupa liver disorder	1.464	4.370	66.5
breast cancer (Wisconsin)	5.572	13.421	58.5
cars	2.173	4.870	55.4
cleveland	3.460	6.042	42.7
glass	1.634	2.884	43.3
hayes-roth	0.720	1.499	52.0
heart desease	3.018	6.228	51.5
iris	0.517	1.536	66.3
led7	20.689	34.905	40.7
letter	222.556	842.784	73.6
pima indian	4.283	11.309	62.1
soybean	1.661	2.808	40.8
tic-tac-toe	7.623	11.031	30.9
vehicle	8.937	18.785	52.4
wine	2.084	4.120	49.4
Media			52.4

Tabla 5.4: Tiempo medio de evaluación: EES vs. recorrido lineal.

Como se puede observar, para todas las bases de datos el tiempo medio empleado por la EES es sensiblemente inferior al del recorrido lineal. Esto hace que la mejora sea siempre positiva, es decir, que la EES reduce el tiempo de evaluación para todos los casos tratados. Si calculamos el promedio de las mejoras para las 15 bases de datos (última fila de la tabla), obtenemos como resultado una mejora media del 52.4 %, es decir, la estructura tarda prácticamente la mitad del tiempo en evaluar las reglas.

Estos resultados experimentales son mostrados gráficamente por la figura 5.20, la cual contiene 15 gráficas, una por cada base de datos tratada. Cada una de estas gráficas representa la variación del tiempo de evaluación en segundos (eje de ordenadas) conforme aumenta el porcentaje de reglas válidas (eje de abscisas), trazando dos curvas: la línea gris muestra el tiempo de evaluación empleado por el método lineal, mientras que la línea de color negro muestra el tiempo empleado usando la EES. Como podíamos intuir, las gráficas desvelan que el comportamiento de la EES es muy favorable, quedando la curva que representa el tiempo empleado por ésta siempre por debajo para todas las bases de datos estudiadas. En otras palabras, la EES resulta ser más eficiente para todos los casos con independencia del tipo de regla (válida o no válida).

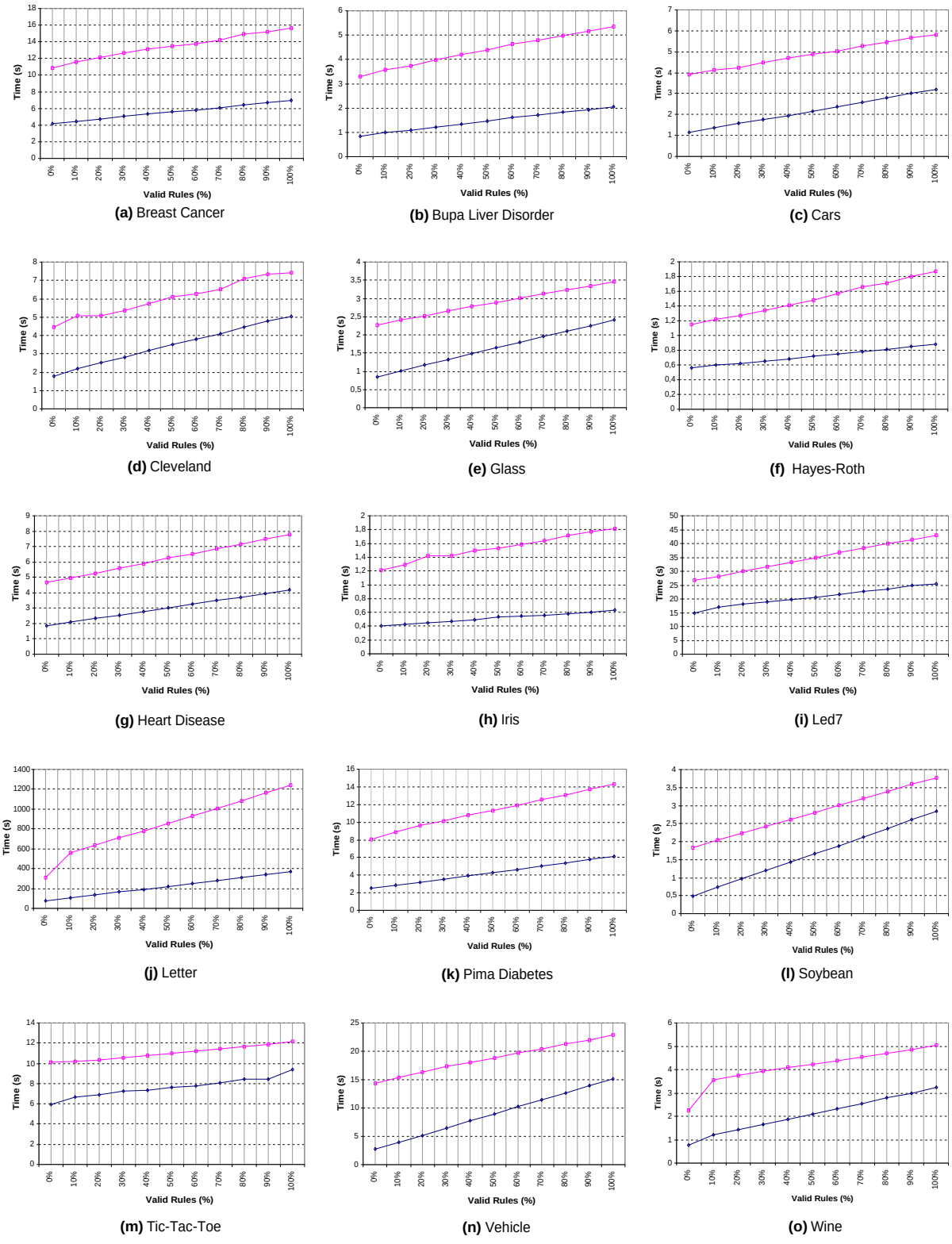


Figura 5.20: Gráficas de tiempo de evaluación EES vs. recorrido lineal.

Por otra parte, como era de esperar, el tiempo de evaluación aumenta conforme crece el porcentaje de reglas válidas para ambos métodos de evaluación. En este sentido, es interesante el hecho de que en 8 de las 15 bases de datos (figura 5.20: a, b, f, g, h, i, j, k), la curva que representa el tiempo invertido por la EES presenta menor pendiente que la curva para el método lineal. Esto indica que, para esas bases de datos, la evaluación con la estructura es más robusta frente a la validez de las reglas que el método lineal.

5.5. Algoritmo

HIDER obtiene un conjunto de reglas de decisión jerárquicas a partir de un conjunto de datos etiquetados. Para ello, sigue un algoritmo evolutivo de cubrimiento secuencial [127], el cual produce, en cada iteración, una única regla que es usada para eliminar ejemplos del conjunto de datos [172]. El proceso se repite hasta que todos los ejemplos han sido cubiertos, es decir, hasta que el conjunto de datos es vacío o el número de ejemplos restantes no se considera significativo. Si comparamos este algoritmo con otros basados en las propuestas de Michigan y Pittsburgh, HIDER reduce el espacio de búsqueda, ya que busca una única regla entre las posibles soluciones en cada iteración.

El pseudocódigo de nuestra propuesta viene dado por la figura 5.21. El algoritmo se divide en dos partes: el procedimiento principal HIDER, el cual construye el conjunto de reglas; y la función auxiliar $AlgEvo$, que devuelve una regla cada vez que es llamada y que implementa el algoritmo evolutivo propiamente dicho. Inicialmente, el conjunto de reglas R está vacío y en cada iteración se añade (mediante el operador $\oplus$) la regla obtenida por la función $AlgEvo$. El parámetro D es el conjunto de datos, el cual es codificado al inicio para acelerar las evaluaciones (véase la sección 5.3.4). Esta transformación da como resultado el conjunto de ejemplos codificados D^* que será usado durante la toda ejecución. A continuación, se construye la estructura EES a partir de D^*, cuyo propósito es mejorar la eficiencia del método de evaluación, tal y como se describió en la sección 5.4. La variable n almacena el número inicial de ejemplos de D^*, ya que éste será reducido en cada iteración. Dicha reducción se produce eliminando aquellos ejemplos de D^* que son cubiertos por la descripción de la regla r, que denotamos

Procedimiento HIDER(D, R)
Entrada: D: Conjunto de datos
Salida: R: Conjunto de reglas
Comienzo
D^* :=Codificar(D) (conjunto de ejemplos codificados)
EES :=Construir(D^*)
$R := \emptyset$
$n := |D^*|$
mientras $|D^*| > n \times fpe$ (fpe =factor de poda de ejemplos)
$r :=$ AlgEvo(D^*, EES)
$R := R \oplus \{r\}$
$D^* := D^* - \{e \in D^* \mid e \subseteq \Delta_r\}$
fin mientras
fin HIDER

Function AlgEvo(D^*, EES)
Entrada: D^*: Conjunto de ejemplos codificados; EES: EES-N
Salida: r: Regla
Comienzo
Inicializar(P)
Desde $i = 1$ **hasta** $num\_generaciones$
Evaluación(P, EES)
Reemplazo(P)
fin desde
Evaluación(P, EES)
r := el_mejor_de(P)
end AlgEvo

Figura 5.21: Pseudocódigo de HIDER.

por Δ_r. El parámetro fpe (factor de poda de ejemplos) controla el número de ejemplos que aún no han sido cubiertos durante el proceso. Este factor evita la generación de reglas que cubran pocos ejemplos al final del proceso. La condición de terminación del bucle se alcanzada cuando el número de ejemplos que restan en el fichero de entrenamiento ($|D^*|$) no supera el porcentaje establecido por fpe sobre el número inicial de ejemplos (n).

El módulo evolutivo $AlgEvo$ inicializa la población P y ejecuta el bucle que lleva a cabo la evolución de ésta según $num\_generaciones$. En cada iteración, el procedimiento $Evaluación$ asigna un valor de bondad a cada individuo de la población actual P, aplicando la función de evaluación descrita más adelante (ecuación 5.26) a los ejemplos resultantes del recorrido de la estructura de evaluación eficiente EES. Tras esto, el procedimiento $Reemplazo$ genera la nueva población mediante la selección, réplica y recombinación de individuos. Cuando el número de generaciones preestablecido es alcanzado, la población final es de nuevo evaluada para seleccionar el mejor individuo de ésta. Tal individuo es devuelto al módulo principal para borrar los ejemplos cubiertos, ser incluido en el conjunto de reglas R y continuar el proceso.

Una vez descrito el funcionamiento del algoritmo de manera gemeral, veamos con mayor nivel de detalle las características del mismo.

5.5.1. Inicialización de la población

El método que se encarga de inicializar la población P ($Inicializar$) toma un conjunto aleatorio de ejemplos del conjunto D^* y genera un individuo por cada ejemplo seleccionado de forma que cada individuo clasificará correctamente ese ejemplo. Para el antecedente de cada individuo, se toman los valores de los atributos del ejemplo correspondiente. Si el atributo es continuo se genera un intervalo aleatorio, en codificación natural, que cubra al valor que toma en el ejemplo. En caso de que sea discreto, se genera un gen natural aleatorio que contenga al valor que el atributo toma en el ejemplo seleccionado. Para que estos individuos clasifiquen correctamente al ejemplo usado para su gestación, se le asigna la misma clase que dicho ejemplo.

5.5.2. Función de Evaluación

La función de evaluación se encarga de medir la calidad de los individuos respecto de la clasificación que éstos realizan. En otras palabras, asigna un valor numérico, que denominamos bondad, a un individuo dependiendo de los aciertos y los errores que éste cometa al clasificar los ejemplos del conjunto de datos. Un individuo comete un error cuando cubre a un ejemplo pero no lo clasifica con la misma clase de éste. En caso contrario, si la clase coincide, entonces decimos que el individuo tiene un acierto. Encontrar una función de evaluación apropiada no es una tarea trivial, ya que ésta puede depender de las características particulares del problema a resolver.

El algoritmo evolutivo maximiza la función de evaluación φ de forma que un individuo r es mejor cuanto mayor sea $\varphi(r)$. La función de evaluación que HIDER aplica viene dada por la ecuación 5.26.

$$\varphi(r) = N - EC(r) + A(r) + cobertura(r) \tag{5.26}$$

donde N es el número de ejemplos que quedan por cubrir en el fichero de entrenamiento; $EC(r)$ es el error de clase, es decir, el número de ejemplos que pertenecen a la región definida por la regla pero que no comparten la misma clase que dicha regla; $A(r)$ es el número de ejemplos correctamente clasificados; y la $cobertura(r)$ es el volumen normalizado de la región cubierta por la regla.

Como se describió en la sección 5.4, $EC(r)$ y $A(r)$ son calculados mediante el recorrido de la EES-N para aumentar la eficiencia del proceso de evaluación.

La cobertura añade un valor en el rango $(0, 1]$ que ayuda a las reglas a expandirse rápidamente para encontrar más ejemplos, ya que, en igualdad de aciertos y errores, se potencian aquellas reglas que cubren un área mayor. El cálculo de la cobertura se lleva a cabo dividiendo el volumen de la región definida por la regla por el volumen total del espacio de búsqueda. Si el atributo a_i es continuo, $[li_i, ls_i]$ representa el intervalo asociado con el atributo a_i en la descripción de la regla y $[Li_i, Ls_i]$ el rango de valores continuos de a_i. Por el contrario, si a_i es discreto, k_i es el número de valores discretos contemplados por la condición de la regla sobre el atributo a_i y $|A_i|$ el número de valores diferentes para tal atributo. Entonces, la cobertura de una regla viene dada por la ecuación 5.27.

$$cobertura(r) = \prod_{i=1}^{m} \frac{cob(r,i)}{rango(r,i)}$$

$$cob(x,i) = \begin{cases} ls_i - li_i & si\ el\ atributo\ es\ continuo \\ k_i & si\ el\ atributo\ es\ discreto \end{cases} \tag{5.27}$$

$$rango(x,i) = \begin{cases} Ls_i - Li_i & si\ el\ atributo\ es\ continuo \\ |A_i| & si\ el\ atributo\ es\ discreto \end{cases}$$

Penalización y Relajación

La función de evaluación puede estar sujeta a dos aspectos denominados penalización y relajación. La penalización consiste en introducir un factor (fp: *factor de penalización*) en la función de evaluación que multiplique los errores contabilizados para una regla si éstos superan un determinado umbral denominado *coeficiente de error permitido* (cep). El cep es un valor que se suele establecer entre el 0 % y el 10 % del total de ejemplos cubiertos. Así, si el porcentaje de errores supera el cep, la función de evaluación que se aplica es

$$\varphi(r) = N - (fp \times EC(r)) + A(r) + cobertura(r) \tag{5.28}$$

Por otro lado, la relajación es el caso contrario a la penalización. Consiste en que, si los errores que una regla comete no superan el cep, se eliminan dichos errores de la función de evaluación, es decir, se multiplica por 0 el número de errores ($EC(i)$). Desde el punto de vista práctico, la relajación podemos interpretarla como una penalización negativa, es decir, si el $fp \in [0,1)$ y $cpe > 0$ estaremos relajando los errores; si por el contrario, $fp \in (1,\infty)$ y $cpe > 0$ estaremos penalizándolos. En el caso de $fp = 1$ ó $cpe = 0$ estaremos aplicando la función de evaluación original (ecuación 5.26).

En general, si relajamos el método obtendremos menor número de reglas sacrificando un poco la tasa de error. Por el contrario, si penalizamos los errores que las reglas cometan, obtendremos mayor número de reglas pero con una tasa de error menor. Tanto el cep como el fp pueden ser configurados por el usuario dependiendo de la base de datos concreta sobre la que se aplique HIDER.

5.5.3. Reemplazo

El reemplazo incluye la selección, réplica y aplicación de los operadores genéticos para producir la nueva población de individuos. Para la selección se ha aplicado en todos los casos el método de la ruleta de la fortuna [44], el cual asigna una probabilidad de selección a cada individuo proporcional a su bondad. Los individuos de la nueva población son obtenidos a partir de la actual según la siguiente política de reemplazo:

1. El mejor individuo pasa a la siguiente población sin ser mutado (elitismo).
2. Una copia mutada del mejor individuo también es replicada.
3. Un porcentaje de individuos son seleccionados y replicados directamente, aplicándoles previamente la mutación según la probabilidad de mutación individual.
4. El resto de la población es obtenida mediante el cruce de pares preseleccionados. Estos nuevos individuos son también mutados según la probabilidad de mutación individual.

5.6. Poda

Muchos algoritmos de clasificación como C4.5 o el propio COGITO incluyen un método de poda que reduce la complejidad de la estructura de conocimiento mediante la eliminación de condiciones o reglas completas. En el ámbito de la generación de reglas de decisión, denominamos *poda parcial* a la supresión de alguna condición de las reglas, mientras que llamamos *poda global* a la eliminación de reglas completas en la estructura jerárquica. HIDER aplica poda parcial indirectamente mediante los operadores de mutación generalizada[6]. Este tipo de mutación hace que el intervalo o conjunto de valores establecido por una determinada condición tome el dominio completo del atributo, por lo que dicha condición no impone restricción alguna y es suprimida de la regla final.

Respecto a la poda global, es importante discernir entre si la poda se realiza a priori, es decir, durante el proceso de generación de reglas, o bien a posteriori, es decir, una vez obtenido el conjunto de reglas. La poda global a priori es controlada mediante el

[6] Véanse páginas 131 y 139.

factor de poda de ejemplos (fpe) (figura 5.21). Como se ha descrito en secciones anteriores, HIDER genera una regla en cada iteración, la cual es usada para eliminar los ejemplos del conjunto de datos cubiertos por tal regla. Este proceso se lleva a cabo mientras se cumpla la condición $|D^*| > n \times fpe$. El factor fpe toma valores en el rango $[0, 1]$ y provoca que cuando el número de ejemplos restantes en el conjunto de datos ($|D^*|$) no supera una determinada proporción (fpe) respecto al número inicial de ejemplos (n), el algoritmo deje de generar reglas. La justificación de esta poda radica en que las reglas generadas a partir de un numero reducido de ejemplos no son determinantes a la hora de clasificar. De hecho, las pruebas empíricas realizadas demuestran que, para ciertas bases de datos, las últimas reglas cubren muy pocos ejemplos. Algunos autores [35, 94] afirman que este tipo de reglas no son deseables, especialmente cuando el dominio incluye ruido. En general, una poda hasta del 5 % de los ejemplos ($fpe = 0.05$) reduce el número de reglas sin provocar pérdidas significativas de precisión.

Por otro lado, la poda global a posteriori consiste simplemente en, una vez generadas todas las reglas, eliminar aquellas que no aportan beneficio al conjunto total de reglas, quedándose así sólo con las mejores y manteniendo la tasa de error. Este tipo de poda es aplicada por C4.5Rules, que poda aquellas las ramas del árbol generado por C4.5 cuya eliminación no aumenta la tasa de error. La aplicación de esta poda en HIDER es trivial, aunque es necesario tener en cuenta la condición jerárquica del conjunto.

5.7. Conclusiones

HIDER es el resultado de la unión de diversas propuestas para mejorar el rendimiento de los algoritmos evolutivos en el aprendizaje supervisado de reglas jerárquicas de decisión, abordando principalmente el diseño de una codificación genética eficaz y un proceso de evaluación eficiente como factores determinantes en la aplicación de este tipo de técnicas. Con el objetivo de comprobar el comportamiento de HIDER en la práctica, llevamos a cabo un amplio conjunto de experimentos. La descripción de éstos junto con los resultados obtenidos son detallados en los Capítulos 6 y 7, resumiendo las conclusiones generales de nuestra investigación en el Capítulo 8.

Capítulo 6

Pruebas

El pensamiento lógico puro no puede brindarnos
ningún conocimiento del mundo empírico.
ALBERT EINSTEIN.

6.1. Rendimiento

Todas las pruebas descritas en esta sección se han llevado a cabo sobre diferentes bases de datos del almacén de aprendizaje automático de la Universidad de California Irvine (UCI *Machine Learning Repository*) [24]. Se han realizado dos estudios comparativos para probar la calidad de HIDER. El primero de ellos compara la herramienta con C4.5 Release 8 y C4.5Rules, en términos de precisión (tasa de error) y complejidad (número de reglas). El segundo estudio analiza la codificación natural frente a la codificación híbrida, cotejando los resultados de HIDER con los obtenidos por la herramienta COGITO. Es importante señalar que tanto la estimación del error como el número de reglas generadas por cada método han sido obtenidos mediante validación cruzada estratificada de 10 conjuntos (*stratified 10-fold cross validation*) [169, 26]. Todos los experimentos fueron realizados usando los mismos conjuntos de entrenamiento y test para todos los algoritmos de modo que los resultados obtenidos fueran comparables.

Otro aspecto notable se refiere a la parametrización del algoritmo evolutivo aplicado por HIDER. Puesto que tanto C4.5 como COGITO no fueron configurados exclusivamente para cada base de datos, y dado que pretendíamos que los experimentos fueran

Parámetro	Descripción	Valor
P	Tamaño de la Población	70
G	Número of Generaciones	100
fpe	Factor de Poda de Ejemplos	0
cep	Coeficiente de Error Permitido	0
fp	Factor de Penalización	1
$\%rep$	Porcentaje de Réplicas	20 %
$\%cru$	Porcentaje de Cruces (100- $\%rep$)	80 %
mut_ind	Probabilidad de Mutación Individual	0.5
mut_gen	Probabilidad de Mutación por Gen	$\frac{1}{\|atributos\|}$
mut_vdisc	Probabilidad de Mutación de Valores Discretos	$\frac{1}{\|valores\|}$

Tabla 6.1: Parámetros de HIDER.

lo más justos posibles, mantuvimos todos los parámetros de HIDER constantes en todas las pruebas. Los valores concretos usados para parametrizar nuestra herramienta fueron los expresados en la tabla 6.1. Nótese en primer lugar que tanto el tamaño de la población como el número de generaciones fueron muy reducidos (70 y 100 respectivamente). Por otra parte, no se aplicó poda global a priori ($fpe = 0$) ni a posteriori, lo que hace al algoritmo más sensible al sobreajuste, pudiendo generar clasificaciones erróneas. La poda puede mejorar en muchos casos el número de reglas sin perjudicar en exceso la tasa de error, sin embargo, el fpe depende de la base de datos utilizada, por lo que decidimos mantener este factor a 0 y hacer un estudio de su influencia independiente a estas pruebas (sección 6.2). Por el mismo motivo, tampoco se aplica penalización ni relajación ($cep = 0$, $fp = 1$). Los parámetros de porcentaje de réplicas ($\%rep$) y cruce ($\%cru$), así como la probabilidad de mutación individual (mut_ind) fueron ajustadas empíricamente. Respecto a la probabilidad de mutación por gen, el valor $mut\_gen = \frac{1}{||atributos||}$ indica que si un individuo ha sido seleccionado para ser mutado, sólo se cambiará el valor de un gen. Si además ese gen representa un atributo discreto, sólo se agregará o substraerá uno de los valores de la condición ($mut\_vdisc = \frac{1}{||valores||}$).

6.1.1. Eficacia: HIDER versus C4.5/C4.5Rules

Con el objetivo de dar una visión global del rendimiento de nuestra herramienta, comparamos HIDER con uno de los sistemas de referencia en aprendizaje supervisado, C4.5 Release 8, el cual es habitualmente utilizado en la bibliografía para este tipo de comparativas. C4.5 genera un árbol de decisión a partir de una base de datos etiquetada, estableciendo condiciones sobre los atributos siguiendo un criterio que maximiza la ganancia de información. Dado que HIDER genera reglas de decisión, hemos incluido en este este estudio comparativo la versión denominada C4.5Rules, la cual construye un conjunto de reglas de decisión a partir del árbol obtenido por C4.5 aplicando criterios de refinado y poda a posteriori.

Las pruebas descritas en esta sección fueron realizadas sobre 18 bases de datos del almacén de UCI, las cuales contienen tanto atributos continuos como discretos. Para cada una de estas bases de datos, medimos la tasa media de error (en porcentaje) y el número medio de reglas generadas por cada método en la 10-validación cruzada. Los resultados obtenidos por HIDER son comparados con los generados por C4.5 en las tablas 6.2 y 6.3, mientras que la comparación con C4.5Rules es explicada en las tablas 6.4 y 6.5 respectivamente. Para dar validez a los resultados obtenidos, se llevó a cabo un test estadístico con una confianza del 95 % con el fin de determinar qué resultados son significativos y cuáles no. En concreto, se aplicó el Test de Student de Diferencia de Medias con $\alpha = 0.05$.

La lectura de la tabla 6.2 (análoga a la de la tabla 6.4) es la siguiente: la primera columna indica la base de datos a la que se le ha aplicado cada método; el siguiente bloque de dos columnas da los resultados de C4.5, es decir, la tasa media de error junto a la desviación estándar (ER$\pm\delta$) y el número medio de reglas junto a la desviación estándar (NR$\pm\delta$). De igual modo, el siguiente bloque de dos columnas presenta los resultados obtenidos por HIDER, con el mismo significado que las dos columnas anteriores, es decir, la tasa media de error y el número medio de reglas así como ambas desviaciones estándar; finalmente, el último bloque de dos columnas muestra los resultados del test estadístico respecto a la tasa media de error (ER) y respecto al número medio de reglas (NR). La interpretación de los símbolos de estas dos columnas es la siguiente: un "$-$" indica que HIDER obtiene peor resultado que C4.5; un "$+$" denota que

	C4.5		HIDER		Test Estadístico	
Base de datos	ER$_{\pm\delta}$	NR$_{\pm\delta}$	ER$_{\pm\delta}$	NR$_{\pm\delta}$	ER	NR
Breast Cancer	6.3$_{\pm 3.0}$	22.9$_{\pm 3.0}$	4.1$_{\pm 2.0}$	2.0$_{\pm 0.0}$	$+$	$+*$
Bupa	33.7$_{\pm 9.3}$	29.7$_{\pm 5.1}$	37.3$_{\pm 11.4}$	4.2$_{\pm 0.8}$	$-$	$+*$
Cleveland	23.5$_{\pm 7.0}$	38.3$_{\pm 5.8}$	25.3$_{\pm 10.5}$	5.9$_{\pm 0.9}$	$-$	$+*$
German	32.9$_{\pm 4.3}$	204.2$_{\pm 8.5}$	27.4$_{\pm 3.9}$	8.0$_{\pm 1.4}$	$+*$	$+*$
Glass	30.8$_{\pm 11.4}$	25.8$_{\pm 2.0}$	35.2$_{\pm 7.8}$	11.7$_{\pm 1.6}$	$-$	$+*$
Hayes-Roth	18.3$_{\pm 5.4}$	14.7$_{\pm 2.5}$	33.1$_{\pm 13.6}$	4.3$_{\pm 1.7}$	$-*$	$+*$
Heart	25.5$_{\pm 5.1}$	33.5$_{\pm 4.5}$	21.9$_{\pm 8.8}$	4.3$_{\pm 0.5}$	$+$	$+*$
Hepatitis	19.4$_{\pm 7.0}$	15.5$_{\pm 1.7}$	16.7$_{\pm 11.0}$	3.7$_{\pm 0.7}$	$+$	$+*$
Horse Colic	19.0$_{\pm 7.7}$	44.4$_{\pm 3.8}$	20.0$_{\pm 7.7}$	11.1$_{\pm 1.9}$	$-$	$+*$
Iris	5.3$_{\pm 6.9}$	5.7$_{\pm 0.5}$	3.3$_{\pm 4.7}$	3.2$_{\pm 0.4}$	$+$	$+*$
Lenses	30.0$_{\pm 21.9}$	5.2$_{\pm 0.9}$	25.0$_{\pm 26.4}$	4.5$_{\pm 0.9}$	$+$	$+$
Mushroom	0.0$_{\pm 0.0}$	17.6$_{\pm 1.0}$	1.2$_{\pm 0.6}$	3.5$_{\pm 0.5}$	$-*$	$+*$
Pima	26.1$_{\pm 5.4}$	24.4$_{\pm 8.1}$	25.7$_{\pm 3.4}$	5.1$_{\pm 0.7}$	$+$	$+*$
Tic-Tac-Toe	14.2$_{\pm 3.4}$	95.1$_{\pm 9.2}$	21.9$_{\pm 5.4}$	6.7$_{\pm 2.3}$	$-*$	$+*$
Vehicle	27.5$_{\pm 3.6}$	74.8$_{\pm 10.0}$	33.8$_{\pm 7.4}$	19.7$_{\pm 3.3}$	$-*$	$+*$
Vote	6.2$_{\pm 3.1}$	15.9$_{\pm 3.0}$	4.4$_{\pm 2.9}$	2.2$_{\pm 0.4}$	$+$	$+*$
Wine	6.7$_{\pm 7.8}$	6.5$_{\pm 0.9}$	8.8$_{\pm 4.2}$	5.6$_{\pm 0.8}$	$-$	$+*$
Zoo	7.0$_{\pm 10.6}$	10.9$_{\pm 1.8}$	4.0$_{\pm 5.2}$	7.9$_{\pm 0.9}$	$+$	$+*$

Tabla 6.2: Comparativa entre C4.5 y HIDER.

HIDER mejora a C4.5; y por último, un "$*$" significa que el resultado es significativo, ya sea positivo o negativo.

Como se puede observar en la tabla 6.2, HIDER mejoró al C4.5 respecto al número de reglas en todas las bases de datos, de las cuales, 17 de estas mejoras resultaron significativas y sólo una no lo fue. Pudiera parecer que este resultado implicaría una pérdida notable de precisión en las reglas, sin embargo, observamos que para la mitad de las bases de datos, HIDER presentó mejoras en la tasa de error, aunque sólo una de éstas tuvo significatividad estadística. De las 9 bases de datos en las que nuestra propuesta presentó pérdidas en la tasa de error, en 4 casos dicha pérdida fue significativa. Para mostrar más claramente la proporción entre los resultados de ambos métodos, la tabla 6.3 muestra una medida de la mejora de la tasa de error (ϵ_{er}) y el número de reglas (ϵ_{nr}), las cuales son calculadas dividiendo los valores obtenidos por C4.5 entre los de HIDER en cada caso. La última fila da la media aritmética de ϵ_{er} y ϵ_{nr}. Aunque en

Base de datos	ϵ_{er}	ϵ_{nr}
Breast Cancer	1.54	11.45
Bupa	0.9	7.07
Cleveland	0.93	6.49
German	1.2	25.53
Glass	0.88	2.21
Hayes-Roth	0.55	3.42
Heart	1.16	7.79
Hepatitis	1.16	4.19
Horse Colic	0.95	4
Iris	1.61	1.78
Lenses	1.2	1.16
Mushroom	0	5.03
Pima	1.02	4.78
Tic-Tac-Toe	0.65	14.19
Vehicle	0.81	3.8
Vote	1.41	7.23
Wine	0.76	1.16
Zoo	1.75	1.38
Media	1.03	6.26

Tabla 6.3: Mejora de HIDER sobre C4.5.

promedio HIDER sólo mejora el error relativo ϵ_{er} en un 3 %, la mejora respecto a ϵ_{nr} es ostensible (526 %), obteniendo éste menos de la quinta parte del número de reglas que C4.5. Destaca el resultado obtenido para las bases de datos *Breast Cancer*, *Tic-Tac-Toe* y *German*, cuya mejora ϵ_{nr} supera el 1000 %.

Respecto a la comparación entre HIDER y C4.5Rules, la tabla 6.4 mantiene la misma estructura que la tabla 6.2, presentando también resultados claramente favorables a HIDER. En lo que concierne al número de reglas, HIDER mejoró en 12 casos, teniendo 11 de éstos casos significación estadística respecto a las reglas generadas por C4.5Rules. De las 6 bases de datos donde C4.5Rules obtuvo menos reglas, sólo 3 casos resultaron significativos. Con respecto a los errores, C4.5Rules y HIDER igualaron sus resultados, teniendo 9 casos a favor cada uno de ellos, 2 de los cuales fueron significativos para cada herramienta. Del mismo modo que con C4.5, la tabla 6.5 muestra la mejora relativa frente a C4.5Rules. En este caso, HIDER mejoró el ϵ_{er} en un 33 %, superando la media

	C4.5Rules		HIDER		Test Estadístico	
Base de datos	ER$_{\pm\delta}$	NR$_{\pm\delta}$	ER$_{\pm\delta}$	NR$_{\pm\delta}$	ER	NR
Breast Cancer	4.9$_{\pm 2.4}$	9.6$_{\pm 1.1}$	4.1$_{\pm 2.0}$	2.0$_{\pm 0.0}$	$+$	$+*$
Bupa	32.0$_{\pm 6.2}$	11.9$_{\pm 2.2}$	37.3$_{\pm 11.4}$	4.2$_{\pm 0.8}$	$-$	$+*$
Cleveland	25.9$_{\pm 14.7}$	12.2$_{\pm 4.6}$	25.3$_{\pm 10.5}$	5.9$_{\pm 0.9}$	$+$	$+*$
German	28.8$_{\pm 3.1}$	6.2$_{\pm 2.2}$	27.4$_{\pm 3.9}$	8.0$_{\pm 1.4}$	$+$	$-$
Glass	18.5$_{\pm 5.9}$	15.0$_{\pm 2.8}$	35.2$_{\pm 7.8}$	11.7$_{\pm 1.6}$	$-*$	$+*$
Hayes-Roth	22.8$_{\pm 3.7}$	11.2$_{\pm 0.9}$	33.1$_{\pm 13.6}$	4.3$_{\pm 1.7}$	$-*$	$+*$
Heart	20.7$_{\pm 7.0}$	11.5$_{\pm 2.0}$	21.9$_{\pm 8.8}$	4.3$_{\pm 0.5}$	$-$	$+*$
Hepatitis	16.9$_{\pm 6.1}$	6.3$_{\pm 3.6}$	16.7$_{\pm 11.0}$	3.7$_{\pm 0.7}$	$+$	$+*$
Horse Colic	17.5$_{\pm 8.2}$	5.0$_{\pm 1.9}$	20.0$_{\pm 7.7}$	11.1$_{\pm 1.9}$	$-$	$-*$
Iris	4.7$_{\pm 7.1}$	5.0$_{\pm 0.0}$	3.3$_{\pm 4.7}$	3.2$_{\pm 0.4}$	$+$	$+*$
Lenses	16.7$_{\pm 22.2}$	4.1$_{\pm 0.3}$	25.0$_{\pm 26.4}$	4.5$_{\pm 0.9}$	$-$	$-$
Mushroom	0.7$_{\pm 2.3}$	17.9$_{\pm 1.9}$	1.2$_{\pm 0.6}$	3.5$_{\pm 0.5}$	$-$	$+*$
Pima	29.7$_{\pm 3.8}$	8.3$_{\pm 3.1}$	25.7$_{\pm 3.4}$	5.1$_{\pm 0.7}$	$+$	$+*$
Tic-Tac-Toe	18.8$_{\pm 16.7}$	11.7$_{\pm 9.7}$	21.9$_{\pm 5.4}$	6.7$_{\pm 2.3}$	$-$	$+$
Vehicle	57.6$_{\pm 14.7}$	4.1$_{\pm 4.1}$	33.8$_{\pm 7.4}$	19.7$_{\pm 3.3}$	$+*$	$-*$
Vote	5.3$_{\pm 3.7}$	7.5$_{\pm 0.7}$	4.4$_{\pm 2.9}$	2.2$_{\pm 0.4}$	$+$	$+*$
Wine	6.7$_{\pm 7.8}$	5.6$_{\pm 0.7}$	8.8$_{\pm 4.2}$	5.6$_{\pm 0.8}$	$-$	$-$
Zoo	29.8$_{\pm 20.7}$	6.3$_{\pm 2.0}$	4.0$_{\pm 5.2}$	7.9$_{\pm 0.9}$	$+*$	$-*$

Tabla 6.4: Comparativa entre C4.5Rules y HIDER.

obtenida en el cotejo con C4.5. Sin embargo, la mejora respecto al número de reglas no fue tan manifiesta, aunque en promedio, HIDER redujo el error con la mitad de reglas aproximadamente (98 %). Nótese que esta comparativa no es totalmente justa con nuestra propuesta, ya que C4.5Rules realiza una poda de las reglas a posteriori, es decir, una vez que ha generado las reglas, elimina aquellas que no aportan beneficio al conjunto total de reglas, quedándose así sólo con las mejores y manteniendo la tasa de error. Como se mencionó con anterioridad, HIDER no aplicó ningún tipo de poda, ni a priori (con el factor fpe) ni a posteriori, para la realización de estos experimentos.

Además de comparar por separado la precisión y la complejidad de los modelos de conocimiento obtenidos por HIDER y C4.5/C4.5Rules, es interesante tener un medida de rendimiento que relacione ambos aspectos para dar una idea más general de la calidad de tales modelos. En este sentido, una medida apropiada que proponemos es el porcentaje de aciertos que cada regla cubre respecto al conjunto total de reglas, y

Base de datos	ϵ_{er}	ϵ_{nr}
Breast Cancer	1.2	4.8
Bupa	0.86	2.83
Cleveland	1.02	2.07
German	1.05	0.78
Glass	0.53	1.28
Hayes-Roth	0.69	2.6
Heart	0.95	2.67
Hepatitis	1.01	1.7
Horse Colic	0.88	0.45
Iris	1.42	1.56
Lenses	0.67	0.91
Mushroom	0.58	5.11
Pima	1.16	1.63
Tic-Tac-Toe	0.86	1.75
Vehicle	1.7	0.21
Vote	1.2	3.41
Wine	0.76	1
Zoo	7.45	0.8
Media	1.33	1.98

Tabla 6.5: Mejora de HIDER sobre C4.5Rules.

que en adelante denominaremos *tasa media de acierto por regla* (A/R). El cálculo de A/R viene dado por la ecuación 6.1, donde ER es la tasa de error y NR el número de reglas.

$$A/R = \frac{100 - ER}{NR} \tag{6.1}$$

Los valores de A/R obtenidos para cada herramienta son expresados por la tabla 6.6. Para cada base de datos, los tres primeros valores numéricos dan la A/R para C4.5, C4.5Rules y HIDER respectivamente, mientras que las dos columnas finales muestran la mejora obtenida por HIDER, siendo tal mejora calculada dividiendo la A/R de C4.5 y C4.5Rules, respectivamente, entre la obtenida por HIDER. Como se puede observar, la comparativa con C4.5 resulta rotundamente favorable a HIDER, consiguiendo una mejora del 540 % en promedio (última fila de la tabla). Por otro lado, la mejora media respecto a C4.5Rules no es tan significativa aunque sigue siendo claramente favorable, siendo ésta del 100 %.

	A/R			Mejora	
Base de datos	C4.5	C4.5Rules	HIDER	C4.5/HIDER	C4.5Rules/HIDER
Breast Cancer	4.1	9.9	48	11.7	4.8
Bupa	2.2	5.7	14.9	6.8	2.6
Cleveland	2	6.1	12.7	6.4	2.1
German	0.3	11.5	9.1	30.3	0.8
Glass	2.7	5.4	5.5	2	1
Hayes-Roth	5.6	6.9	15.6	2.8	2.3
Heart	2.2	6.9	18.2	8.3	2.6
Hepatitis	5.2	13.2	22.5	4.3	1.7
Horse Colic	1.8	16.5	7.2	4	0.4
Iris	16.6	19.1	30.2	1.8	1.6
Lenses	13.5	20.3	16.7	1.2	0.8
Mushroom	5.7	5.5	28.2	4.9	5.1
Pima	3	8.5	14.6	4.9	1.7
Tic-Tac-Toe	0.9	6.9	11.7	13	1.7
Vehicle	1	10.3	3.4	3.4	0.3
Vote	5.9	12.6	43.5	7.4	3.5
Wine	14.4	16.7	16.3	1.1	1
Zoo	8.5	11.1	12.2	1.4	1.1
Media				6.4	2

Tabla 6.6: Tasa media de aciertos por regla (A/R) de C4.5,C4.5Rules y HIDER.

El principal inconveniente de HIDER, y en general de cualquier algoritmo basado en búsquedas probabilísticas, es el elevado tiempo de ejecución que precisa para encontrar buenas soluciones. Mientras que C4.5 invirtió 4 minutos para completar la 10-validación cruzada de las 18 bases de datos, HIDER precisó 1 hora y 34 minutos para las mismas pruebas. Así, si dividimos estos tiempos entre el total de 180 ejecuciones que se llevaron a cabo, C4.5 necesitó algo más de 1 segundo para generar el árbol correspondiente a una única base de datos frente a los 30 que HIDER empleó para producir el conjunto de reglas jerárquicas. Aunque estos resultados temporales juegan en contra de nuestra propuesta, C4.5 ha demostrado dar un rendimiento muy difícil de mejorar, tanto en precisión y complejidad, como en tiempo de ejecución [25, 112]. Teniendo en cuenta que muchos de los problemas abarcados por el campo de la minería de datos no necesitan un procesamiento en tiempo real de la información, podemos

considerar que HIDER arrojó resultados muy satisfactorios desde el punto de vista de a calidad y la eficacia de las estructuras de conocimiento producidas.

En resumen, podemos afirmar que HIDER mantiene unas tasas de error aceptables, reduciendo sensiblemente el número de reglas. Teniendo en cuenta sólo los resultados con significación estadística en las 36 pruebas realizadas, HIDER obtiene prácticamente la misma tasa de error frente a C4.5 y C4.5Rules, reduciendo ésta en 3 pruebas (8.3 %) e incrementándola en 6 (16.6 %). Los resultados respecto al número de reglas son mucho más dispares, alcanzando HIDER mejoras significativas en 28 casos (77.7 %) y pérdidas en sólo 2 (5.5 %). Respecto a la tasa media de acierto por regla, cada regla producida por HIDER acierta aproximadamente 5 veces más que una generada por C4.5 y el doble que una de C4.5Rules, en promedio.

6.1.2. Eficiencia: HIDER versus COGITO

El objetivo principal de estas pruebas es mostrar la mejora en eficiencia que el uso de la codificación natural de HIDER aporta frente a la codificación híbrida que COGITO aplica. Para ello, comparamos el rendimiento ofrecido por ambas herramientas sobre 16 bases de datos heterogéneas del almacén UCI. COGITO aplica un algoritmo evolutivo similar al de HIDER, cuyos parámetros fueron ajustados con los mismos valores que éste (véase la tabla 6.1) a excepción del número de generaciones y el tamaño de la población. El uso de la codificación natural reduce el espacio de búsqueda y acelera la convergencia del algoritmo. Por ello, HIDER sólo precisó 100 generaciones y 70 individuos para obtener mejores resultados que COGITO, el cual necesitó 300 generaciones y 100 individuos.

Al igual que en la sección anterior, el rendimiento de ambos métodos fue determinado mediante validación cruzada estratificada con 10 conjuntos, midiendo la tasa media de error (ER) y el número medio de reglas (NR) para cada base de datos. Estas pruebas arrojaron los resultados que aparecen en la tabla 6.7, la cual, además de la tasa de error y el número de reglas que cada herramienta produjo, muestra la mejora obtenida por HIDER (ϵ_{er} y ϵ_{nr}). Tal mejora es calculada como el cociente de los valores obtenidos por COGITO y HIDER para cada medida. Finalmente, los valores de la última fila representan la media aritmética de las columnas ϵ_{er} y ϵ_{nr}.

	COGITO (híbrida)		HIDER (natural)		Mejora	
Base de Datos	ER	NR	ER	NR	ϵ_{er}	ϵ_{nr}
Breast Cancer	4.3	2.6	4.06	2	1.06	1.3
Bupa	35.7	11.3	37.35	4.2	0.96	2.69
Cleveland	20.5	7.9	25.33	5.9	0.81	1.34
German	29.1	13.3	27.4	8	1.06	1.66
Glass	29.4	19	35.24	11.7	0.83	1.62
Heart	22.3	9.2	21.85	4.3	1.02	2.14
Hepatitis	19.4	4.5	16.67	3.7	1.16	1.22
Horse Colic	17.6	6	20	11.1	0.88	0.54
Iris	3.3	4.8	3.33	3.2	0.99	1.5
Lenses	25	6.5	25	4.5	1	1.44
Mushroom	0.8	3.1	1.18	3.5	0.68	0.89
Pima	25.9	16.6	25.66	5.1	1.01	3.25
Vehicle	30.6	36.2	33.81	19.7	0.91	1.84
Vote	6.4	4	4.42	2.2	1.45	1.82
Wine	3.9	3.3	8.82	5.6	0.44	0.59
Zoo	8	7.2	4	7.9	2	0.91
Media					1.02	1.55

Tabla 6.7: Comparativa entre COGITO y HIDER.

Aunque el propósito de estos experimentos era mostrar la mejora en eficiencia de HIDER, podemos ver que éste también supera a COGITO en eficacia. Pese a que la tasa de error sólo se logró disminuir en la mitad de las bases de datos, el número de reglas sí fue reducido en 12 de los 16 casos. Así, aunque la mejora media respecto al error fue muy reducida (2 %), la mejora en el número de reglas alcanza el 55 % en promedio, aumentando así la calidad del modelo de conocimiento. Es cierto que la mejora en la precisión y la complejidad no es tan sensible como en las comparativas con C4.5 y C4.5Rules, pero no debemos olvidar que HIDER necesitó sólo la tercera parte de las generaciones y tres cuartas partes del tamaño de la población requeridos por COGITO.

La principal aportación de la codificación natural es el aumento de eficiencia del algoritmo evolutivo en la generación de reglas. El uso de esta codificación reduce el coste computacional tanto en tiempo como en espacio. Aunque ambos aspectos están estrechamente relacionados, vamos a analizar por separado cada uno de ellos.

Respecto a la reducción en el tiempo de ejecución, HIDER invirtió 1 hora y 20 minutos en completar todas la pruebas, aproximadamente la cuarta parte del tiempo empleado por COGITO que fue de unas 5 horas y media. La razón de este descenso radica en la reducción del espacio de búsqueda que la codificación natural provoca, principalmente en los dominios continuos donde la previa aplicación del algoritmo de discretización USD decrementa el número de posibles soluciones. Dado que esta discretización maximiza la bondad de los intervalos que genera, su aplicación no ocasiona pérdidas de precisión en las reglas, como comprobamos en la tabla 6.7. La reducción del espacio de búsqueda junto a las características del propio algoritmo hace que éste converja más rápidamente, lo cual posibilita la obtención del modelo en pocas generaciones y con poblaciones relativamente pequeñas. Sin embargo, la rápida convergencia del algoritmo no es el único factor responsable del decremento del tiempo de ejecución, ya que no hay que olvidar que la estructura EES también aporta beneficios en este sentido.

Por otro lado, la longitud de los individuos naturales es menor que la de los híbridos. Esto repercute directamente en el espacio necesario para almacenar las poblaciones. Como recoge la tabla 6.8, el número de genes de los individuos depende directamente del tipo de atributos de la base de datos, más concretamente del número de atributos continuos (NC) y discretos (ND), así como del número total de valores diferentes que estos últimos pueden tomar (NV). Como estudiamos en la sección 5.3.2, la longitud individual[1] usando la codificación natural es $L_n = NC + ND$, mientras que aplicando la codificación híbrida dicha longitud es $L_h = 2 \times NC + NV$. La tabla 6.8 muestra en sus dos últimas columnas las longitudes L_h y L_n para cada una de las bases de datos. Como se puede observar, la codificación natural decrementa notablemente la longitud de los individuos, obteniendo una reducción superior al 63 % en promedio respecto a los individuos híbridos.

Resumiendo, HIDER obtiene mejoras en eficacia frente a COGITO reduciendo fundamentalmente el número de reglas manteniendo la tasa de error. La aplicación de la codificación natural permite que HIDER obtenga estos resultados usando aproximadamente un tercio de los recursos computacionales que COGITO necesita, tanto en términos de tiempo como en espacio.

[1]Este valor de longitud no incluye la clase, ya que ambas codificaciones usan la misma representación para ésta, es decir, un único gen que enumera las diferentes etiquetas.

Base de Datos	NC	ND (NV)	L_h	L_n
Breast Cancer	9	-	18	9
Bupa	6	-	12	6
Cleveland	6	7 (19)	31	13
German	7	13 (54)	68	20
Glass	9	-	18	9
Heart	13	-	26	13
Hepatiti	6	13 (26)	38	19
Horse Colic	7	15 (55)	69	22
Iris	4	-	8	4
Lenses	-	4 (9)	9	4
Mushroom	-	22 (117)	117	22
Pima	8	-	16	8
Vehicle	18	-	36	18
Vote	-	16 (48)	48	16
Wine	13	-	26	13
Zoo	-	16 (36)	36	16
Media			36	13.3

Tabla 6.8: Tamaño de los individuos para codificación híbrida y natural.

6.2. Análisis de influencia de la Poda

Como se indicó al inicio de este capítulo, los experimentos para medir el rendimiento de HIDER frente a C4.5, C4.5Rules y COGITO fueron realizados sin aplicar poda global debido a que el ajuste del fpe (factor de poda de ejemplos) es dependiente de la base de datos. Para comprobar la influencia de este parámetro, se realizaron pruebas empíricas sobre 5 bases de datos del almacén UCI: *Bupa*, *German*, *Heart*, *Horse Colic* y *Pima*. Estas pruebas consistieron en tomar cada base de datos y aplicar HIDER variando el fpe desde el 0 % hasta el 20 % para comprobar la evolución de la tasa de error así como el número de reglas a medida que se incrementa el factor de poda. Al igual que en los experimentos anteriores, se aplicó 10-validación cruzada estratificada para cada valor de fpe.

Los resultados obtenidos son ilustrados por la figura 6.1, la cual muestra las gráficas de variación del error y el número de reglas frente al fpe para cada base de datos. Como

podemos observar, todas las gráficas presentan un comportamiento muy similar. A medida que el porcentaje de poda es incrementado, el número de ejemplos empleados para el entrenamiento es menor. Como era de suponer, esto provoca un descenso en el número medio de reglas, lo que hace que la tasa media de error ofrezca una tendencia creciente. Sin embargo, cabe destacar el hecho de que mientras que la tasa de error presenta un crecimiento más o menos lineal con pendiente suave, el número de reglas decrece bruscamente para valores pequeños del fpe (entre 0 y 5 %), estabilizándose a medida que aumenta el porcentaje de poda.

El uso de un determinado valor del fpe dependerá del área concreta de aplicación así como de las características (ruido, distribución de los datos, etc) de la base de datos sobre la cual se desee adquirir conocimiento. En este sentido, el experto en el área debe decidir qué error se puede tolerar o bien acotar el número de reglas para así determinar un fpe adecuado. Concretamente, en estas pruebas, aunque todas la bases de datos tuvieron un comportamiento similar, el fpe apropiado para cada una de ellas fue diferente. La tabla 6.9 indica los valores del fpe que mejores resultados conjuntos ofrecían respecto a las dos medidas consideradas (ER_p y NR_p), así como la mejora en el error ($\epsilon_{er} = \frac{ER_0}{ER_p}$) y el número de reglas ($\epsilon_{nr} = \frac{NR_0}{NR_p}$) respecto a la ejecución carente de poda (ER_0 y NR_0). Los símbolos que aparecen junto a ϵ_{er} y ϵ_{nr} indican si la mejora ha sido positiva ("$+$"), o si por el contrario se produjo pérdida ("$-$"). Para validar estadísticamente la significatividad de los resultados, aplicamos el test de Student de diferencia de medias con una confianza del 95 %, marcando con "$*$" aquellos valores que resultaron significativos. Para cada medida, δ representa la desviación estándar necesaria para realizar el test estadístico.

Observando los resultados de la tabla 6.9, sólo se obtuvo mejora de la tasa de error en 1 base de datos y pérdidas en 4, aunque ninguno de estos valores resultaron significativos, siendo la pérdida global sólo del 2 % en promedio (última fila). Sin embargo, todas las bases de datos presentaron mejoras significativas en lo que respecta al número de reglas, siendo la mejora media global del 73 %. Por tanto, podemos colegir que la aplicación adecuada de la poda aumenta sensiblemente el rendimiento de HIDER, ya que se disminuye significativamente la complejidad del modelo de conocimiento sin penalizar la exactitud de predicción del mismo.

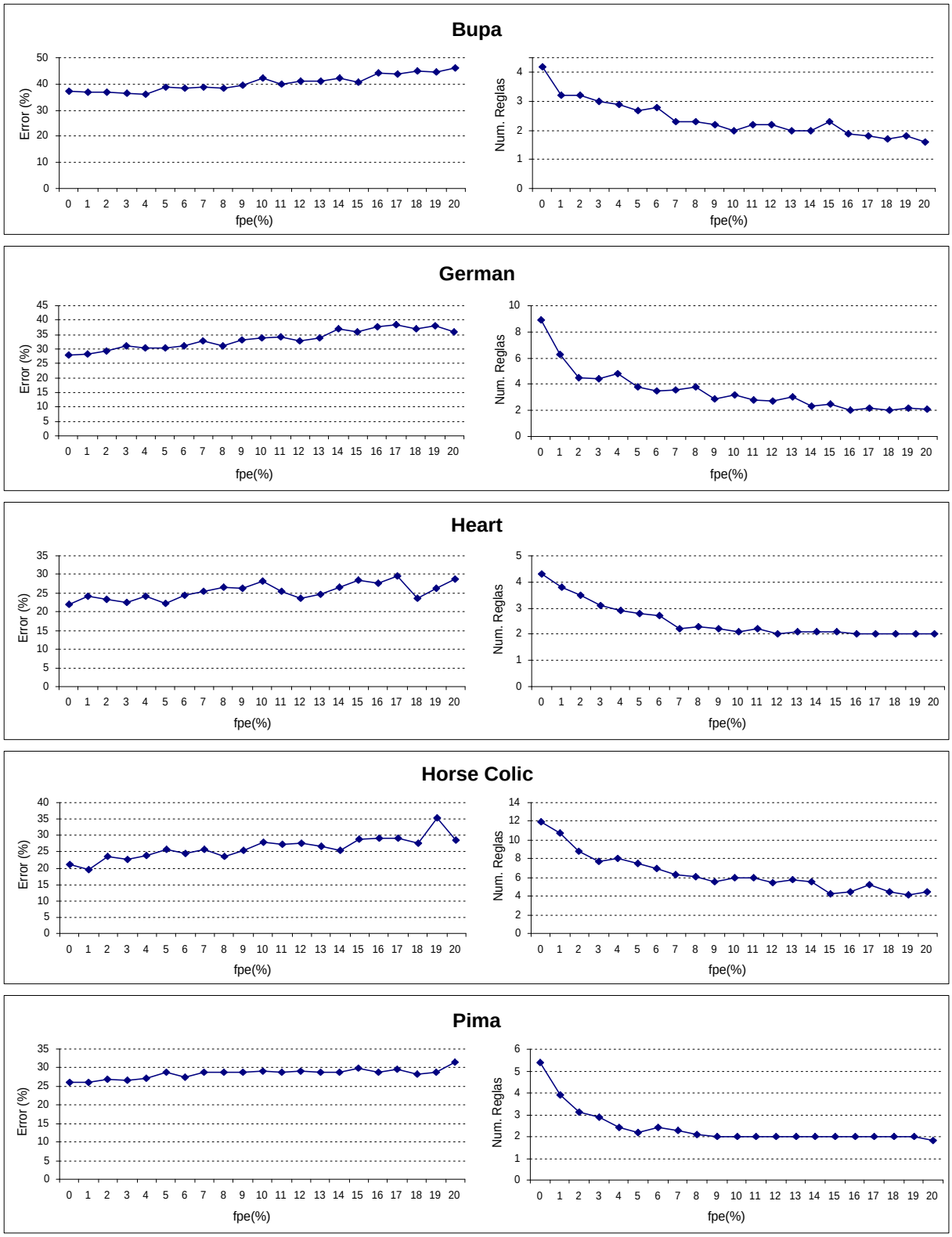


Figura 6.1: Tasa de error y número medio de reglas variando fpe.

	Sin Poda ($fpe = 0$)		Con Poda			Mejora	
Base de Datos	$ER_{0\pm\delta}$	$NR_{0\pm\delta}$	$fpe(\%)$	$ER_{p\pm\delta}$	$NR_{p\pm\delta}$	ϵ_{er}	ϵ_{nr}
Bupa	37.3±10.8	4.2±0.7	4	35.8±10.1	2.9±0.3	1.04 +	1.45 +∗
German	28.0±3.8	8.9±0.9	2	29.4±3.6	4.5±1.2	0.97 −	1.98 +∗
Heart	21.9±8.3	4.3±0.4	5	22.2±9.6	2.8±0.6	0.99 −	1.54 +∗
Horse Colic	21.1±7.2	11.9±1.3	3	22.7±6.7	7.7±1.2	0.93 −	1.55 +∗
Pima	25.9±3.7	5.1±1.0	4	27.2±4.0	2.4±0.4	0.95 −	2.13 +∗
Media						0.98	1.73

Tabla 6.9: Valores óptimos de fpe.

Aunque no es un resultado generalizado, el caso de la base de datos *Pima* resulta cuanto menos interesante. Se trata de una base de datos muy dispersa cuya clasificación es muy compleja. Por ello, una tasa de error entre el 25 y el 30 % se considera aceptable. Observando la figura 6.1, si aplicamos una poda con fpe entre el 9 % y el 19 %, el error siempre se mantiene por debajo del 30 % clasificando los ejemplos de la base de datos con 2 únicas reglas. Puesto que *Pima* sólo tiene dos etiquetas de clase, HIDER logra clasificarla con una regla por etiqueta de clase manteniendo un error aceptable, algo que ningún clasificador presente en la bibliografía conseguido.

6.3. Conclusiones

Tras la realización de las pruebas empíricas y la comparación con C4.5, C4.5Rules y COGITO, HIDER ha demostrado ser un sistema robusto en términos de eficacia y eficiencia, mejorando los resultados de las anteriores herramientas.

La comparativa con C4.5 y C4.5Rules arroja resultados muy satisfactorios, sobre todo en lo que se refiere a la minimización del número de reglas, donde HIDER reduce significativamente la complejidad de la estructura de conocimiento para la gran mayoría de la bases de datos utilizadas sin que se produzcan pérdidas en la exactitud de las predicciones. De hecho, la tasa media de error también es mejorada en términos globales, aunque este aumento de la precisión no es tan sensible como la reducción de la complejidad.

Respecto a la eficiencia, la aplicación de la codificación natural resulta altamente ventajosa frente a la codificación híbrida usada en COGITO. La reducción del espacio

de búsqueda, así como el diseño de los operadores genéticos favorece la aceleración de la convergencia del algoritmo, lo que permite obtener mejores resultados en un tiempo más reducido. En general, HIDER utilizó menos de un tercio de los recursos computacionales empleados por COGITO para obtener mejores modelos de conocimiento, principalmente en lo que se refiere al número de reglas.

Por último, tras el análisis de influencia del factor de poda de ejemplos (fpe), concluimos que, para las bases de datos analizadas, la aplicación de una poda global de 2–5 % reduce aún más el número medio de reglas sin que esto repercuta significativamente en la tasa de error. El ajuste particular del fpe para cada base de datos hubiera establecido diferencias mayores con los modelos generados por las otras herramientas empleadas en estas pruebas.

Capítulo 7

Aplicación de HIDER a un Proceso Industrial

La inteligencia consiste no sólo en el conocimiento,
sino también en la destreza de aplicar los conocimientos en la práctica.
ARISTÓTELES.

Como se introdujo en el Capítulo 1, el objetivo central de nuestra investigación era mejorar las técnicas de aprendizaje evolutivo desde el punto de vista de la eficiencia y la eficacia de los algoritmos que las implementan. Como resultado principal hemos obtenido la herramienta de generación de reglas de decisión HIDER. Normalmente, las pruebas empíricas para establecer comparativas entre las diferentes propuestas son realizadas sobre bases de datos disponibles para la comunidad científica, con el fin de que tales pruebas sean reproducibles y comprobables. Sin embargo, una de nuestras inquietudes era saber cómo se comportaría HIDER frente a algún problema real y poco conocido.

Este capítulo presenta una aplicación práctica de nuestra investigación a un proceso de fabricación química, y es el resultado de la colaboración mantenida entre la empresa ATLANTIC COPPER S.A. y nuestro grupo de investigación formado por profesores. En concreto, nuestro trabajo consiste en generar modelos de conocimiento aplicando HIDER a partir de los datos cuantitativos proporcionados por esta empresa y facilitar así la toma de decisiones en la producción de ácido sulfúrico a partir de los desechos del proceso de obtención del cobre, con el objetivo de optimizar aprovechamiento de ambos productos.

7.1. Planteamiento del problema

ATLANTIC COPPER S.A. es uno de los mayores productores de cobre a nivel mundial. Su actividad económica se centra en la fabricación y comercialización de diversos productos derivados de este material.

La producción de cobre a partir de las materias primas es un proceso complejo que requiere la intervención de varios subsistemas, cada uno de los cuales se encarga de una fase del proceso productivo. Durante la actividad de cada uno de estos subsistemas, se liberan una serie de materiales que, en lugar de constituir material de desecho, pueden servir como base para otros subproductos, entre los que se encuentra el ácido sulfúrico.

Este estudio consiste en la aplicación de técnicas de minería de datos que permitan modelar el comportamiento de ciertos aspectos críticos del sistema de producción de ácido sulfúrico. El propósito es posibilitar la definición de actuaciones por parte de los expertos que sirvan para optimizar el funcionamiento del sistema. Así, los objetivos que se persiguen con la realización este estudio son:

1. Identificar los parámetros que intervienen en el comportamiento del sistema, así como conocer el grado de influencia de cada uno de ellos.

2. Predecir las posibles respuestas del sistema ante diferentes actuaciones sobre los parámetros.

3. Determinar las causas de los periodos de inestabilidad del sistema.

4. Descubrir actuaciones innecesarias o perjudiciales en el control del sistema, así como descubrir nuevas acciones no consideradas.

5. Obtener reglas de funcionamiento que permitan modelar el sistema con un alto grado de confianza.

La consecución de los anteriores objetivos y el análisis de los expertos de la empresa les permitirá obtener:

6. Un mayor conocimiento del sistema que sirva de apoyo en la toma de decisiones.

7. Una optimización del funcionamiento del sistema de producción, y consecuentemente, una reducción de costes de producción y una mayor protección del medio ambiente.

Todos estos puntos están estrechamente interrelacionados, lo que requiere una colaboración continua entre los investigadores y empresa. Por ello, la consecución satisfactoria de estos objetivos pasa por obtener un modelo de conocimiento del sistema útil, preciso y comprensible para los expertos. Nuestro trabajo consiste en la aplicación de HIDER para obtener reglas de decisión que describan el comportamiento del sistema de producción y permitir alcanzar las metas marcadas.

7.2. Proceso productivo del ácido sulfúrico

La primera etapa del proceso de producción del cobre es la fundición, dando como resultado *Ánodo de cobre* como producto principal. Esta transformación libera gases con un alto contenido en azufre, los cuales se aprovechan para producir ácido sulfúrico como subproducto. De este modo no sólo se obtiene otro producto comercializable, sino que se evita el grave impacto medioambiental que la expulsión de estos gases a la atmósfera puede provocar.

La fabricación de ácido sulfúrico involucra tres pasos básicos:

1. Generación de anhídrido sulfuroso (SO_2, dióxido de azufre).

2. Conversión de SO_2 en anhídrido sulfúrico (SO_3, trióxido de azufre).

3. Absorción del SO_3 en una solución de agua para formar el ácido sulfúrico (H_2SO_4).

El primero de estos pasos queda cubierto en el proceso de fundición de minerales sulfurados de cobre, donde se produce gran cantidad de dióxido de azufre. Mediante una *planta de ácido* es posible convertirlo en ácido sulfúrico, utilizando para ello una serie de procesos físicos y químicos (pasos 2 y 3). Además, otros elementos altamente contaminantes como el arsénico, mercurio y selenio son eliminados de los gases durante este proceso, evitando así ser emitidos a la atmósfera.

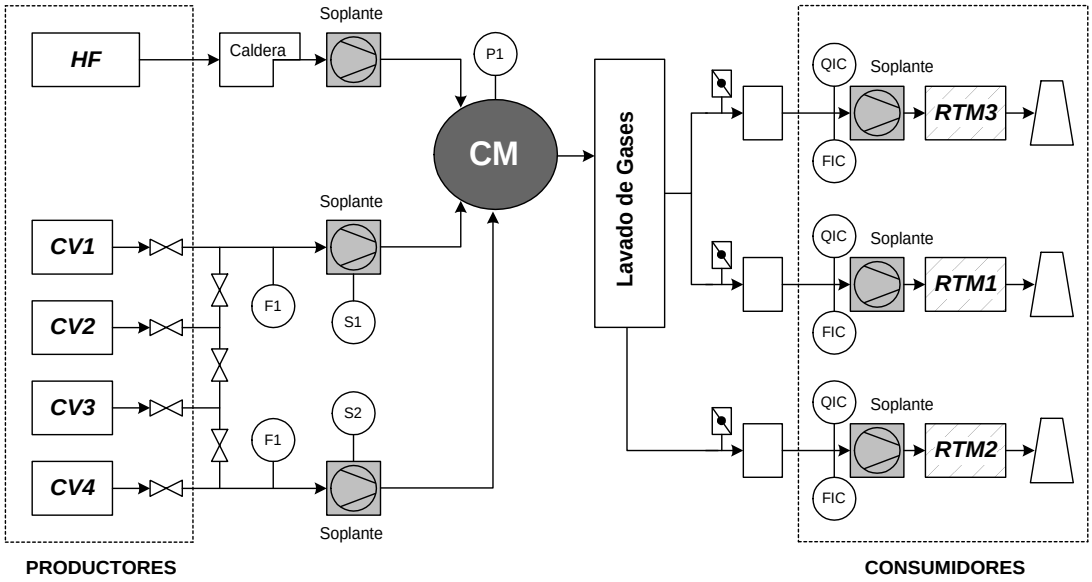


Figura 7.1: Sistema de producción de ácido sulfúrico [136].

7.2.1. Descripción del sistema de producción

Antes de comenzar el estudio, es necesario conocer el funcionamiento básico del sistema de producción de ácido sulfúrico. La figura 7.1 presenta un esquema simplificado del sistema mencionado. Se trata de una red de flujos de gases que contienen azufre en mayor o menor medida.

En el esquema se identifican elementos productores y consumidores de gases. Por un lado, los productores son el *Horno Flash* (*HF*) y cuatro *Convertidores* (*CV1*, *CV2*, *CV3* y *CV4*). Por otro lado, las tres plantas de la fábrica de ácido (*RTM1*, *RTM2* y *RTM3*) son los consumidores. Cada uno de estos elementos constituye un subsistema completo y muy complejo, cuya descripción no es necesaria para este estudio.

Los gases generados en los productores son canalizados hasta la *Cámara de Mezcla* (*CM*), a partir de la cual se distribuyen a los distintos consumidores según sus necesidades. Este flujo de gases es posible gracias a las *soplantes*, que conducen los gases de un punto a otro regulando el caudal de gas a través de los conductos. Se dice que el extremo hacia el cual la soplante actúa tiene *presión*, mientras que el extremo contrario tiene *tiro*.

El funcionamiento del sistema debe garantizar la canalización adecuada de los gases desde los productores hasta los consumidores de modo que el tiro y la presión de los componentes estén compensados. En este sentido, es de vital importancia que en la cámara de mezcla no haya exceso de presión, pues esto puede producir que los gases escapen a la atmósfera, lo cual hay que evitar en todo momento. El exceso de tiro también es perjudicial, ya que puede producir una falta de presión en la cámara y provocar la entrada de aire externo, adulterando la mezcla.

El sistema consta de un gran número de parámetros. Dada la cantidad de situaciones diferentes que se dan continuamente, el ajuste de estos parámetros resulta muy complejo. Ello provoca que, en ocasiones, el sistema se vuelva inestable temporalmente, pudiendo causar un aumento de presión en la cámara de mezcla. En estas circunstancias, la capacidad de respuesta de los técnicos es vital para estabilizar el sistema antes de que se produzcan escapes.

Aunque en la actualidad se toman todas las medidas oportunas para evitar la anterior situación, es deseable prever o conocer las circunstancias que provocan alguna deficiencia en el tiro de la cámara de mezcla.

7.3. Estudio

El estudio global se encuentra en sus primeras fases, por lo que no contamos con un dominio del problema suficientemente amplio que nos permita profundizar en objetivos muy complejos. Por otra parte, las necesidades más urgentes de la empresa pasan por conocer de manera global en qué circunstancias el sistema funciona correctamente, es decir, cuándo las condiciones de presión y tiro en la cámara de mezcla son estables. Por ello, el estudio que aquí presentamos va dirigido a generar un modelo que establezca la relación entre los valores de diferentes parámetros y el funcionamiento de la cámara de mezcla.

El estudio se divide en las siguientes etapas, las cuales podemos identificar claramente con las fases del proceso $\mathcal{KDD}$ (véase figura 2.1, pág. 14):

1. Determinación de Objetivos: entendimiento del dominio de la aplicación y análisis de viabilidad del estudio.

2. Preparación: preprocesado (eliminación de ruido), transformación y reducción de los datos (selección de características y preparación de la clase).

3. Minería de datos: estudios previos con otras herramientas de minería y aplicación de HIDER.

4. Análisis: interpretación del conocimiento extraído.

5. Aplicación: uso del conocimiento descubierto.

Cada entidad participante se ha dedicado a una parte del problema. Concretamente, nuestro trabajo se centra en la fase de minería de datos propiamente dicha. Por ello, sólo se describirán con detalle los aspectos abordados directamente en nuestra investigación, soslayando aquellos no estrictamente necesarios.

7.3.1. Los datos

El sistema consta de una red de sensores instalados en numerosos puntos del mismo, que establecen el estado del proceso en cada una de sus partes. Estos sensores toman medidas periódicamente, generando una gran cantidad de datos. Como resultado del funcionamiento de la red de sensores se obtiene una gran base de datos que contiene la evolución temporal del sistema. Cada ejemplo representa el estado del sistema en un instante determinado, siendo cada atributo una medición (valor continuo) tomada en un sensor concreto.

Es importante señalar que estos datos son dependientes de la configuración de la planta de ácido, por lo que no siempre son comparables los resultados obtenidos para programaciones diferentes del sistema. Por ello, por cada configuración, se obtendrá un conjunto de reglas diferente. Así, en la fase aplicación, se deberá usar el modelo obtenido para una configuración de aprendizaje similar a aquella para la que se pretenda predecir las actuaciones. Tales configuraciones son determinadas por los expertos de la empresa.

Los datos facilitados por la empresa hasta el momento son dos bases de datos, correspondientes a las mediciones tomadas en la planta de ácido durante dos días

completos, concretamente el 18/01/2002 y el 23/01/2002. Para diferenciarlos, denominaremos a estos conjuntos de datos BD18 y BD23 respectivamente. En ambos casos, la frecuencia de medición fue de 2 segundos, lo que nos permite contar con 43.201 ejemplos por cada base de datos. Cada ejemplo contiene un total de 57 atributos (parámetros del $TCM1$ al $TCM57$). Hay que tener en cuenta que la configuración de la planta fue diferente para cada día, por lo que es necesario realizar estudios independientes.

Al tratarse de un sistema real, los datos presentan algunos problemas para su tratamiento: valores ausentes y ruido provocado por el mal funcionamiento de algunos sensores, mediciones poco relevantes y clase continua, entre otras. Esto hace imprescindible el preprocesado de los datos para su preparación antes de la etapa de aprendizaje. Este tratamiento se detalla en [135], siendo las siguientes las actuaciones más significativas:

- Un fallo en la red de sensores el día 23/01/2002, entre las 9:12:36 y las 9:14:54, provocó que BD23 presentara 70 ejemplos con la mayoría de los parámetros ausentes. Estos ejemplos fueron eliminados, dejando la BD23 con 43.131 casos.

- Tras un análisis de la correlación entre los parámetros, fueron eliminados aquellos parámetros altamente correlacionados, además de los que presentaban el mismo valor para todas las instancias.

- Dado que se pretendía realizar predicciones sobre los valores de la cámara de mezcla, la etiqueta de clase venía dada por los parámetros relacionados con ésta. Concretamente, según los expertos de la empresa, el funcionamiento del sistema se considera correcto cuando la diferencia entre los parámetros $TCM42$ y $TCM12$ se encuentra en el rango $[-0.5, 0.5]$, siendo la etiqueta de clase "*OK*". Cualquier otro valor denota mal funcionamiento, siendo en este caso "*NOK*" la etiqueta de clase.

La tabla 7.1 resume las características de las bases de datos una vez preprocesadas y preparadas para el aprendizaje. El conjunto final de parámetros, así como la descripción de los mismos, viene dado por la tabla A.1 (Apéndice A).

Base de datos	Ejemplos	Atrib.	Instancias de Clase	
			OK	NOK
BD18 (18/01/2002)	43.201	29	29.997 (69 %)	13.204 (31 %)
BD23 (23/01/2002)	43.131	26	25.376 (59 %)	17.755 (41 %)

Tabla 7.1: Bases de datos preprocesadas.

7.4. Estudios complementarios

Con idea de obtener una primera visión de la complejidad de los datos respecto a la clasificación, además de tener algunos resultados de otras técnicas con los que poder comparar, decidimos aplicar C4.5 y C4.5Rules a las dos bases de datos. Dado que se trata de una aplicación práctica, no pretendemos testar la calidad de estas herramientas, sino simplemente obtener modelos de conocimiento considerando sus limitaciones. Por ello, se han usado las bases de datos completas como conjunto de entrenamiento y test.

Los resultados completos generados por ambas herramientas son mostrados en el Apéndice A (figuras A.1-A.4), donde no se ilustran los árboles de decisión por razones de espacio. La tabla 7.2 presenta un resumen de estos resultados. Respecto al porcentaje de error (ER), éste es calculado sobre el mismo conjunto de entrenamiento. Por otra parte, como ya habíamos apuntado en capítulos anteriores, los árboles de decisión tienden a crecer en aplicaciones reales. Este hecho se da para nuestro caso de estudio, por lo que los resultados para C4.5 mostrados en la tabla corresponden a los árboles una vez aplicada la poda.

Como se puede observar, los resultados no son satisfactorios desde el punto de vista de la potencial aplicación del conocimiento extraído. Sin embargo, sí pueden resultar útiles para determinar algunas características de los datos. C4.5 comete muy pocos errores, pero a costa de generar árboles muy grandes. Estos no son modelos válidos, pues los árboles deben ser posteriormente interpretados y validados por los expertos. Una posible solución sería considerar únicamente aquellas ramas que más ejemplos cubran. Sin embargo, las hojas con mayor cobertura no sobrepasan los 150 ejemplos, lo

	C4.5		C4.5Rules	
Base de datos	ER	NR	ER	NR
BD18	6.8	1674	27.7	3
BD23	3.5	905	41.0	3

Tabla 7.2: Resultados de C4.5 y C4.5Rules para BD18 y BD23.

que descarta por completo el modelo generado por C4.5. Con respecto a C4.5Rules se da el caso contrario, es decir, clasifica con muy pocas reglas, sólo 3 en ambos casos, lo que dispara el error.

Observando más detenidamente los resultados de C4.5, podríamos pensar a priori que, al igual que éste, C4.5Rules debería comportarse mejor frente a BD23 que con BD18. Sin embargo, obtiene un error mucho mayor para el mismo número de reglas. La razón radica en que la reglas generadas para BD23 cubren a muy pocos ejemplos, por lo que clasifica prácticamente toda la base de datos con la clase por defecto, que en este caso es "OK". De ahí, que el error sea máximo (nótese que coincide con el porcentaje de instancias de la clase "NOK" en la tabla 7.1). El error de C4.5Rules para BD18 es más bajo por dos razones: primero, las reglas cubren más ejemplos y la clase por defecto es "NOK", que tiene menor número de apariciones en la base de datos; y segundo, porque el porcentaje de instancias de la clase "NOK" es menor que en BD23.

Teniendo en cuenta que C4.5 y C4.5Rules son sistemas robustos que tienen un buen comportamiento para una gran cantidad de dominios, los resultados aquí analizados dejan patente la dificultad de encontrar un modelo sencillo y preciso para estas bases de datos.

7.5. Aplicación de HIDER

Como se ha podido comprobar a lo largo de la lectura de esta memoria, HIDER genera un conjunto de reglas de decisión jerárquicas que modelan el comportamiento global de un conjunto de datos frente a los valores de una clase. Sin embargo, es relativamente sencillo forzar a HIDER a buscar objetivos concretos en lugar de un modelo

global. En este sentido, el mayor interés de la empresa se centraba en detectar intervalos de valores de los parámetros para los que el sistema se mantiene estable con cierta seguridad, más que en tener un modelo global que contemplara todas las posibles situaciones.

Según el planteamiento anterior, HIDER fue aplicado en modos distintos para cubrir dos objetivos particulares:

1. Modo jerárquico: es el habitual y obtiene de reglas jerárquicas que describen el comportamiento global del sistema de producción.

2. Modo no jerárquico: generación de reglas no jerárquicas, que descubran regiones seguras (clase "OK") e inseguras (clase "NOK") en el hiperespacio definido por los parámetros.

Para eliminar la jerarquía del conjunto de reglas, simplemente hay que fijar un objetivo, *i.e.* una etiqueta de clase fija, en cada ejecución y hacer que HIDER genere sólo reglas para esa clase. Esto se logra obligando a la función de inicialización a generar la población inicial con individuos de una única clase. Puesto que la clase no es mutada y los cruces generan descendencia con la clase de los padres, todos los individuos mantendrán la misma clase durante el proceso evolutivo.

Con independencia del modo de ejecución de HIDER, la configuración del algoritmo evolutivo para la aplicación fue la dada en la tabla 7.3. Como podemos ver, los valores de los parámetros son similares a los utilizados en las pruebas descritas en el Capítulo 6. El tamaño de la población se ha aumentado a 150 individuos para ampliar el campo de actuación sobre el espacio de búsqueda. Se ha fijado el fpe al 1 % para evitar la generación de reglas que cubran pocos ejemplos al final del proceso evolutivo. Los valores influyentes en la réplica y recombinación de los individuos permanecen iguales, exceptuando mut_ext, que ha sido incrementada para favorecer la eliminación de condiciones en las reglas.

Por otro lado, los expertos de la empresa consideraban que, inicialmente, un error en torno al 15 % es suficientemente bajo para sus objetivos. Por ello, el parámetro cep es fijado a 0.15, ya que establece el límite de error permitido con cierta flexibilidad. Para que este valor tenga efecto y el algoritmo sea poco tolerante con las reglas que superen

Parámetro	Descripción	Valor
P	Tamaño de la Población	150
G	Número of Generaciones	100
fpe	Factor de Poda de Ejemplos	1 %
cep	Coeficiente de Error Permitido	0.15
fp	Factor de Penalización	4
$\%rep$	Porcentaje de Réplicas	20 %
$\%cru$	Porcentaje de Cruces (100- $\%rep$)	80 %
mut_ind	Probabilidad de Mutación Individual	0.5
mut_gen	Probabilidad de Mutación por Gen	$\frac{1}{\|atributos\|}$
mut_ext	Probabilidad de Mutación al extremo	0.15
mut_vdisc	Probabilidad de Mutación de Valores Discretos	$\frac{1}{\|valores\|}$

Tabla 7.3: Configuración de HIDER para el estudio.

el error permitido, es necesario dar un valor relativamente alto de penalización. En este caso fp es establecido a 4, valor que multiplica el número de errores en caso de superar el cep.

7.6. Resultados

Los datos analizados en esta sección se corresponden con los dos objetivos particulares mencionados anteriormente. Así, para cada modo de funcionamiento, jerárquico y no jerárquico, HIDER se ejecutó varias veces para cada base de datos por separado. Al igual que se hiciera en los estudios complementarios, la tasa de error fue calculada, en cada caso, sobre el mismo conjunto de entrenamiento.

7.6.1. Reglas jerárquicas

Las reglas jerárquicas dan una visión global de la conducta del sistema de producción para una configuración preestablecida. Basándonos en los estudios complementarios descritos en la sección 7.4, así como en trabajos anteriores [136, 135], podemos intuir que el conjunto de parámetros ($TCMi$) define un espacio donde las regiones "OK" y "NOK" no están claramente diferenciadas.

Los resultados de precisión y complejidad obtenidos tras la aplicación de HIDER para las dos bases de datos aparecen en la tabla 7.4. Además de la tasa de error (ER) y el número total de reglas (NR), entre paréntesis se muestra el número de reglas de clase "OK" y "NOK", respectivamente.

Base de Datos	ER	NR (OK | NOK)
BD18	14.6	128 (88 | 40)
BD23	13.8	37 (22 | 15)

Tabla 7.4: Resultados de HIDER jerárquico.

El primer aspecto que podemos deducir es que la configuración para la base de datos BD18 provoca mayor riesgo de inestabilidad en el sistema, pues el número de regiones (reglas) es significativamente superior. Como consecuencia directa de esto, el ajuste de los parámetros para equilibrar los niveles de la cámara de mezcla entraña mayor dificultad.

Respecto al error, vemos que HIDER logra mantenerlo por debajo de los niveles esperados (15 %) para ambas bases de datos. Este hecho tiene un efecto contraproducente en la complejidad del modelo, ya que provoca que el número de reglas sea relativamente elevado al no encontrar regiones que cubran muchos ejemplos sin sobrepasar el error permitido.

Observando las reglas obtenidas[1], podemos colegir que el sistema tiende a definir regiones de funcionamiento correcto que incluyen pequeñas zonas inestabilidad con pocos ejemplos, muy esparcidas y difíciles de discriminar. Esto justifica el número de reglas producido por HIDER. Recordemos que C4.5 obtenía 1674 y 905 reglas para BD18 y BD23 respectivamente, debido a que precisa dividir mucho el espacio para reducir el error. Por contra, C4.5Rules tiende a obviar esas pequeñas regiones, por lo que el error resulta casi máximo al clasificar prácticamente todo el espacio con clase "OK".

La base de datos BD23 presenta una excepción a este razonamiento. Observando las dos primeras reglas de la jerarquía (Apéndice A, figuras A.7 y A.8), vemos que se distinguen dos grandes regiones: una de estabilidad, con 13123 casos "OK" y 2192

[1]En el Apéndice A, sección A.3, se muestran los primeros niveles de jerarquía de estas reglas.

casos "NOK"; y otra de funcionamiento incorrecto, con 8855 y 1207 ejemplos "NOK" y "OK", respectivamente. Sin embargo, para el resto del espacio da un comportamiento similar al de BD18.

Finalmente, concluimos que, con los datos disponibles hasta el momento, es difícil encontrar un modelo de comportamiento global sencillo e inteligible para la base de datos BD18 sin que el error supere el 15 %. Por el contrario, para la configuración dada en BD23, el conjunto de reglas obtenido es relativamente reducido y, por tanto, es posible realizar un estudio del mismo para obtener directrices de actuación, sobre todo para las dos grandes regiones anteriormente mencionadas. Un dato interesante que nos indica el buen funcionamiento de la heurística de HIDER es que los resultados se corresponden en gran medida con la experiencia de los técnicos.

7.6.2. Reglas no jerárquicas

Las reglas no jerárquicas han sido generadas para dirigir la búsqueda sobre una clase concreta, con el propósito de localizar regiones en el espacio de parámetros con un comportamiento determinado. En este sentido, el interés de la empresa se centraba principalmente en descubrir regiones seguras (clase "OK"), para llevar el estado del sistema a estas regiones en caso de inestabilidad. Aunque éste puede ser el principal objetivo, nos parece interesante estudiar también las zonas no estables (clase "NOK"), de modo que puedan ser evitadas.

Los resultados descritos en esta sección corresponden a 4 ejecuciones de HIDER independientes, una para cada base de datos y cada clase. Las tablas 7.5 y 7.6 muestran las 5 primeras reglas[2] de cada tipo para BD18 y BD23, respectivamente. Así, para cada regla[3] se tiene la clase, el número de aciertos y errores, la tasa de error y el número total de ejemplos cubiertos.

Respecto a la comparación entre las bases de datos, estas pruebas desvelan los mismos aspectos generales que los experimentos realizados en modo jerárquico. La base de datos BD18 resulta más compleja para generar el modelo que la BD23. El número de ejemplos cubiertos por cada regla en esta última es sensiblemente mayor que en la

[2]En el Apéndice A, sección A.4, se muestran algunas de las reglas generadas.
[3]Notación: RNJ(x)-y = Regla No Jerárquica con clase y para BDx.

Regla	Clase	Aciertos	Errores	ER(%)	Cubiertos
RNJ(18)-1	OK	4417	779	15.0	5196
RNJ(18)-2	OK	4761	823	14.7	5584
RNJ(18)-3	OK	4247	711	14.3	4958
RNJ(18)-4	OK	1391	242	14.8	1633
RNJ(18)-5	OK	1105	190	14.7	1295
RNJ(18)-1	NOK	872	113	11.5	985
RNJ(18)-2	NOK	879	119	11.9	998
RNJ(18)-3	NOK	344	37	9.7	381
RNJ(18)-4	NOK	241	24	9.1	265
RNJ(18)-5	NOK	218	36	14.2	254

Tabla 7.5: Resultados de HIDER no jerárquico para BD18.

BD18. Un aspecto destacable es que para BD23, las dos primeras reglas de cada clase reúnen una gran cantidad de ejemplos, separando las dos regiones que ya conocíamos por los experimentos anteriores.

Por otro lado, el hecho de que las reglas RNJ(18)-i para la clase "NOK" cubran muy pocos ejemplos, aun cuando se centra la búsqueda en este objetivo, denota que estos casos se encuentran agrupados en pequeñas zonas dispersas en el espacio de atributos. Esta situación se repite para la base de datos BD23, excluyendo la primera regla. Esta conclusión se refuerza observando de nuevo las reglas jerárquicas. Para BD18, no se encuentran reglas de inestabilidad hasta la sexta iteración del algoritmo, mientras que, para BD23, la siguiente regla de clase "NOK" tras la segunda es generada en séptimo lugar. Podemos colegir que la dispersión de las reglas de inestabilidad aumenta la dificultad para ajustar los parámetros.

Por otro lado, hay que destacar que a pesar de haber eliminado la jerarquía, la heurística de la búsqueda es la misma. Este aspecto hace que las reglas generadas por HIDER para ambos modos de funcionamiento sean similares, aunque las no jerárquicas cubran algunos ejemplos más manteniendo la precisión. Este pequeño aumento en la cobertura es debido a que el modo no jerárquico explora más reglas de una misma clase que aplicando la jerarquía, afinando más la búsqueda. Como no puede reducir el error sensiblemente, el valor de cobertura presente en la función de evaluación hace que las reglas más grandes tengan mayor bondad.

Regla	Clase	Aciertos	Errores	ER(%)	Cubiertos
RNJ(23)-1	OK	14324	2518	15.0	16842
RNJ(23)-2	OK	2373	290	10.9	2663
RNJ(23)-3	OK	1446	100	6.5	1546
RNJ(23)-4	OK	613	93	13.2	706
RNJ(23)-5	OK	640	51	7.4	691
RNJ(23)-1	NOK	8426	1251	12.9	9677
RNJ(23)-2	NOK	669	116	14.8	785
RNJ(23)-3	NOK	714	110	13.3	824
RNJ(23)-4	NOK	290	18	5.8	308
RNJ(23)-5	NOK	260	41	13.6	301

Tabla 7.6: Resultados de HIDER no jerárquico para BD23.

Respecto al objetivo principal de estas pruebas, es decir, la generación de reglas de estabilidad, hemos obtenido resultados inicialmente satisfactorios. En una primera evaluación, podemos afirmar que las tres primeras reglas generadas para las dos bases de datos definen regiones seguras y suficientemente grandes para facilitar la estabilidad del sistema.

7.7. Conclusiones

En este capítulo hemos presentado una aplicación práctica de HIDER a un problema real, en concreto al control de una planta de producción de ácido sulfúrico. A partir de la información facilitada por ATLANTIC COPPER S.A. y tras la realización de estudios previos, HIDER ha sido aplicado para modelar el comportamiento del sistema de flujo de gases de la planta de ácido. De este estudio han surgido dos modelos de reglas jerárquicas para dos configuraciones distintas del sistema de producción. También han sido generadas reglas no jerárquicas con el fin descubrir regiones de estabilidad donde poder llevar el estado del sistema en caso de mal funcionamiento.

En un primer análisis, podemos calificar los resultados de satisfactorios, ya que coinciden en gran medida con las observaciones de los técnicos. Los modelos han sido enviados a la empresa y se encuentran en fase de evaluación por parte de los expertos.

Desde el punto de vista de la comprobación del rendimiento de nuestra herramienta, los resultados obtenidos desvelan que HIDER es un sistema robusto y preciso. Sin embargo, un problema que presentan las reglas es el elevado número condiciones que contienen. Tras analizar éstas, pudimos comprobar que algunas condiciones establecían límites muy próximos a los extremos del rango de valores del atributo sobre el que actuaban. Esto nos hace pensar que esas condiciones puedan ser eliminadas sin penalizar la precisión.

Capítulo 8

Conclusiones y Trabajos Futuros

Solamente aquel que construye el futuro
tiene derecho a juzgar el pasado.
FRIEDRICH W. NIETZSCHE.

8.1. Conclusiones

Los algoritmos evolutivos han sido extensamente utilizados para resolver problemas de optimización y búsqueda, teniendo especial relevancia en tareas de aprendizaje automático. El éxito de estos algoritmos radica principalmente en su facilidad para adaptarse a diversos dominios de aplicación. Esta versatilidad depende, en gran medida, de la codificación y la evaluación de los individuos de la población. Por un lado, la codificación determina el espacio de búsqueda de soluciones, lo que afecta al tiempo de ejecución (pues una reducción del número de soluciones potenciales acelera la convergencia del algoritmo) y a la exactitud de las reglas (pues establece las posibles fronteras de decisión). Por otro lado, la función de evaluación mide la calidad de las soluciones, al tiempo que el modo en que tal función es aplicada sobre el conjunto de entrenamiento repercute sustancialmente en la eficiencia del algoritmo. Por ende, ambos aspectos intervienen directamente tanto en la precisión del modelo obtenido como en el coste computacional invertido.

El propósito inicial de esta tesis doctoral fue mejorar las técnicas de aprendizaje evolutivo de reglas. La búsqueda de este objetivo general nos ha llevado a desarrollar diferentes propuestas, cuyas conclusiones se resumen en las siguientes secciones.

8.1.1. USD: Discretización Supervisada No Paramétrica

Aunque esta investigación fue inicialmente enfocada hacia las mejoras de los algoritmos evolutivos de aprendizaje, durante el desarrollo de las diferentes ideas fueron surgiendo necesidades no cubiertas por las propuestas de la bibliografía, al menos no satisfactoriamente. Entre estas necesidades, destaca la referente a la discretización de los atributos continuos. Muchos trabajos de investigación han ido dirigidos en este sentido [15, 21, 52, 57, 94, 108], pero ninguno de los métodos propuestos tenía como objetivo la posterior generación de reglas de decisión mediante un algoritmo evolutivo.

La discretización supervisada mediante el algoritmo USD, obtiene un conjunto de intervalos disjuntos maximizando la bondad media de dichos intervalos. Éste es un aspecto muy importante desde el punto de la posterior generación de reglas, ya que la bondad de los intervalos condicionará la exactitud de esas potenciales reglas en la clasificación. Además, este algoritmo no precisa parametrización, lo cual supone una ventaja frente a otras propuestas.

USD ha sido comparado con otros discretizadores, aplicándolo como preprocesado de los datos y probando la calidad de la discretización mediante su uso en diversos algoritmos evolutivos de generación de reglas [3]. Nuestra propuesta presentó resultados muy satisfactorios, produciendo mejor rendimiento de los algoritmos evolutivos que otros métodos de discretización utilizados para el estudio.

Finalmente, USD no sólo es aplicable a la generación de reglas de decisión, sino que ha sido extendido a otros campos del aprendizaje automático. Así, además de ser un discretizador supervisado genérico, se ha extendido su funcionalidad mediante el clasificador simple USD-C y el método de reducción de ejemplos USD-E.

8.1.2. HIDER: Aprendizaje Evolutivo de Reglas de Decisión

Durante el transcurso de esta investigación, se han desarrollado diferentes propuestas, abordando principalmente los dos aspectos de codificación y evaluación comentados, con el objetivo de mejorar la eficiencia y la eficacia en el aprendizaje evolutivo de reglas de decisión. HIDER es el fruto de la integración de esas propuestas en una única

herramienta, descrita en detalle en el Capítulo 5 y siendo las siguientes sus principales aportaciones:

- La codificación natural representa el dominio de los atributos (discretos y continuos) mediante conjuntos finitos de números naturales. Así, se reduce el tamaño del espacio de búsqueda respecto a la codificación híbrida, lo que hace posible que la búsqueda de soluciones se lleve a cabo de forma más eficaz y eficiente. Por una parte, el uso de la codificación natural, unida a los operadores genéticos diseñados para la misma, produce mejoras en la calidad del modelo, aumentando su precisión y disminuyendo el número de reglas generadas. Parte de esta mejora también es debida al aprovechamiento de las propiedades de los intervalos calculados mediante el algoritmo USD. Asimismo, acelera la convergencia del algoritmo evolutivo, ya que disminuye el número de posibles soluciones. Otro aspecto a destacar de esta codificación es que utiliza un único gen por cada atributo, lo que supone una longitud menor de los individuos.

- Los operadores genéticos naturales son expresiones algebraicas simples, que transforman directamente los genes (números naturales) sin necesidad de realizar ningún tipo de conversión a los dominios originales de los atributos. Los operadores naturales no sólo son computacionalmente eficientes en su aplicación, sino que también influyen favorablemente en la búsqueda de las mejores soluciones. En este sentido, un operador de mutación especial permite la generalización de las reglas mediante la eliminación de condiciones innecesarias en las mismas.

- Tras diversas pruebas empíricas con diferentes funciones de evaluación, la mostrada por la ecuación 5.26 fue la que mejores resultados arrojó. Tal función equilibra adecuadamente los aciertos y errores de los individuos, al tiempo que posibilita la expansión de las reglas para cubrir más ejemplos como consecuencia de la aplicación de la cobertura. Además, es posible el ajuste de la misma mediante el coeficiente de error permitido y el factor de penalización, dependiendo si se pretende aumentar la exactitud del modelo o reducir la complejidad del mismo.

- Se han incluido diversos mecanismos de poda para simplificar tanto el número de condiciones dentro de las reglas (poda parcial o generalización) como el

número total de reglas del modelo (poda global). Las pruebas específicas realizadas ponen de manifiesto el buen funcionamiento de estos métodos de poda en la simplificación del modelo.

- La estructura de datos EES organiza la información del conjunto de datos de manera que la evaluación de la población sea eficiente. El proceso usado tradicionalmente para evaluar un individuo, recorre todos los ejemplos del conjunto de entrenamiento, independientemente de que éstos sean o no clasificados por dicho individuo. El uso de esta estructura permite discriminar aquellos ejemplos que no son cubiertos por una determinada regla, de forma que durante la evaluación de un individuo, sólo sean considerados los ejemplos estrictamente necesarios. Los resultados obtenidos tras las pruebas *ad hoc*, muestran una reducción del tiempo de evaluación mayor al 50 % respecto a la evaluación lineal.

Para comprobar empíricamente el rendimiento de HIDER se llevaron a cabo diversas pruebas con bases de datos del UCI Repository, cuyos resultados fueron posteriormente comparados y verificados estadísticamente con los obtenidos por C4.5, C4.5Rules y COGITO. Tras el análisis de las pruebas, concluimos que HIDER tiende a reducir significativamente la complejidad del modelo de conocimiento respecto a C4.5 y C4.5Rules, disminuyendo a la vez la tasa de error. Si tenemos en cuenta la tasa media de acierto por cada regla, podemos afirmar que cada regla generada por HIDER contiene el mismo conocimiento que dos reglas de C4.5Rules y más de seis de C4.5. Respecto a la comparativa con COGITO, nuestra propuesta obtiene modelos de menor complejidad manteniendo la tasa de error, utilizando menos de la tercera parte del tiempo de ejecución y del espacio de almacenamiento precisados por COGITO.

Uno de nuestros objetivos más recientes es conocer cómo se comporta HIDER ante un problema real. En este sentido, estamos llevando a cabo un estudio en colaboración con la empresa ATLANTIC COPPER S.A., en el que investigamos una aplicación práctica de nuestras herramientas a la industria de la química. Concretamente, nuestro trabajo consiste en generar modelos de conocimiento aplicando HIDER a partir de los datos cuantitativos proporcionados por esta empresa, con el propósito final de facilitar y agilizar la toma de decisiones durante el proceso de producción de ácido sulfúrico. Aunque tal estudio se encuentra en sus etapas iniciales, los primeros resultados ya han

sido obtenidos. Estos resultados refuerzan las conclusiones derivadas de las pruebas empíricas realizadas sobre las bases de datos de UCI.

8.2. Trabajos Futuros

El desarrollo de esta tesis doctoral deja abiertas algunas líneas de investigación. A este respecto, actualmente estamos avanzando en las siguientes direcciones:

- Ampliar la definición del método USD, de modo que realice una discretización dinámica donde el tratamiento de cada atributo no fuera independiente, sino que se tuvieran en cuenta los efectos que un determinado corte o intervalo tiene en el resto de características. En una primera aproximación, podríamos establecer que la unión de dos intervalos adyacentes durante la etapa de refinamiento estuviera condicionada no sólo por la bondad media de la misma y las clases mayoritarias, sino también por los cortes del resto de atributos en el subespacio definido por los intervalos participantes en la posible unión. En la figura 8.1 se presenta un ejemplo simple, donde sólo se ha representado el subconjunto de datos y los intervalos necesarios para aclarar esta idea.

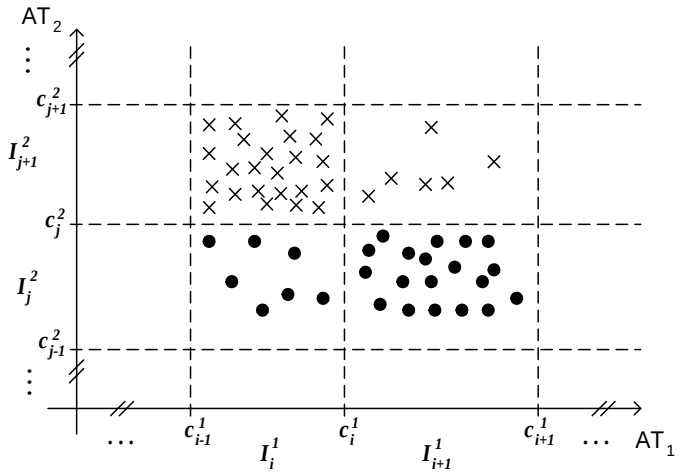


Figura 8.1: USD dinámico.

Calculados los cortes iniciales para los dos atributos del conjunto de datos, nos planteamos si es beneficioso unir los intervalos I_i^1 y I_{i+1}^1 del atributo AT_1. Con el

algoritmo actual, esta unión no sería posible puesto que no comparten la misma clase mayoritaria. Sin embargo, si analizamos los cortes del otro atributo, observamos que en el corte c_j^2 divide claramente el subconjunto de ejemplos que cumplen que $AT_1 \in [c_{i-1}^1, c_{i+1}^1]$ en dos intervalos puros (I_j^2 y I_{j+1}^2). Luego, en este caso, aunque I_i^1 y I_{i+1}^1 no tengan la misma clase mayoritaria, su unión resulta ventajosa, ya que sabemos que para ese nuevo intervalo, hay otro atributo que discrimina correctamente los ejemplos de clases diferentes.

- Análisis durante la ejecución del algoritmo evolutivo de la influencia de los atributos en la evaluación de los individuos. La siguiente definición describe brevemente esta idea.

  Definición 8.1 (Influencia de un atributo). *Sea $\varphi(r_i)$ la función de evaluación global que asigna un valor de bondad al individuo r_i. Sea $\varphi_j(r_i)$ la función de bondad parcial, resultado de aplicar $\varphi(r_i)$ sin tener en cuanta el atributo a_j, es decir, tomando la condición sobre tal atributo como cierta. Definimos la influencia parcial del atributo a_j sobre el individuo r_i como $\mathcal{I}_j(r_i) = \varphi(r_i) - \varphi_j(r_i)$. Así, si P es el tamaño de la población de individuos, la influencia global del atributo a_j viene dada por la expresión*

$$\mathcal{I}_j = \sum_{i=1}^{P} \frac{\mathcal{I}_j(r_i)}{P} \tag{8.1}$$

  El análisis de la influencia parcial y global puede ser aplicado con diferentes objetivos, entre los que destacamos los siguientes:

  - Reajustar la función de evaluación durante la evolución para mejorar la estimación de la bondad de los individuos.
  - Reordenación de los atributos en la estructura EES según su influencia global para aumentar la probabilidad de parada en la evaluación.
  - Aplicar una selección de características basada en la influencia global para reducir el conjunto de atributos durante el proceso evolutivo de extracción de una regla.
  - Utilizar la influencia parcial para la generalización las reglas, de modo que si $\mathcal{I}_j(r_i)$ no supera un determinado umbral, la condición sobre el atributo a_j es suprimida de r_i.

  - Ajuste variable de los valores de probabilidad de aplicación de los diferentes operadores genéticos durante la ejecución del algoritmo. En este sentido, la influencia parcial podría proporcionar estimaciones particulares para cada individuo.

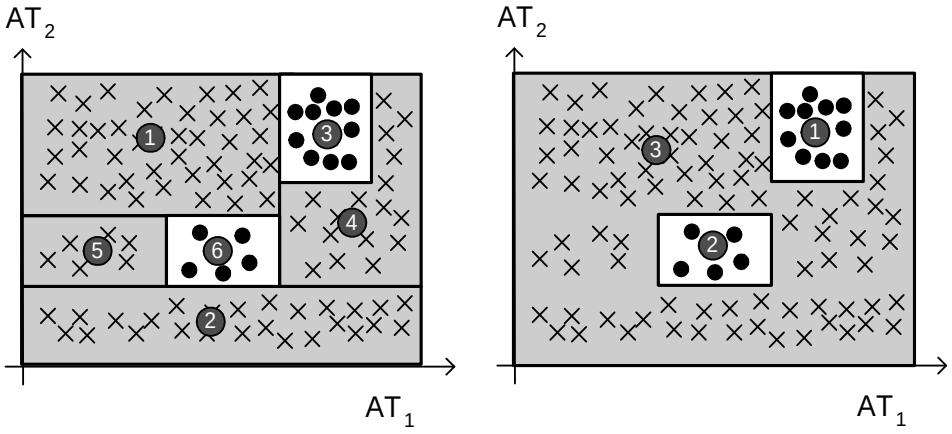
- Aplicación de diferentes técnicas de aprendizaje para construir un modelo de estimación de la función de evaluación. Actualmente, estamos trabajando en el diseño de un modelo neuro-evolutivo (NEM, *Neural-Evolutionary Model*) [10], el cual utiliza una red neuronal para calcular la bondad de los individuos. La red es entrenada durante el proceso evolutivo de forma que, tras un número de generaciones, ésta es capaz de evaluar los individuos a partir de los valores de sus genes sin que sea preciso procesar el conjunto de datos.

- Adaptación de los operadores genéticos durante la ejecución del algoritmo evolutivo, teniendo en cuenta factores como la edad de la población, la capacidad de generalización y especialización, entre otros.

- Diseño de un algoritmo memético [130] que reajuste los intervalos mediante técnicas de búsqueda local durante y después del proceso evolutivo para aumentar la precisión del modelo de conocimiento obtenido.

- Aplicación de técnicas de escalabilidad y aprendizaje incremental para reducir el alto coste computacional del algoritmo evolutivo, por ejemplo, aplicando un muestreo inteligente a los datos de entrenamiento [18, 50, 70].

- Una nueva codificación en la que cada individuo pueda representar un conjunto de reglas y no una única como en HIDER. Ello permitiría la búsqueda simultánea de varias reglas y por tanto la exploración de combinaciones de soluciones que no pueden ser descubiertas con la codificación actual. La figura 8.2 ilustra esta idea mostrando dos posibles soluciones según el tipo de codificación utilizada para un conjunto de datos con dos atributos y dos clases. Cada regla está numerada según su orden en la jerarquía. Con la codificación actual, se podría obtener el modelo representado en la figura 8.2a. HIDER tiende a encontrar (dependiendo el coeficiente de error permitido) primero aquellas reglas que más ejemplos y



(a) Codificación actual (HIDER) (b) Codificación multiregla

Figura 8.2: Codificación con varias reglas por individuo.

más área del espacio de atributos cubren. Así, la primera regla calculada sería la de la parte superior izquierda, ya que es la que más ejemplos cubre de la misma clase sin cometer errores. El crecimiento de esta regla se ve truncado por los ejemplos de la otra clase que posteriormente serán cubiertos por las reglas 3 y 6. En general, si la base de datos presenta pequeños conjuntos de ejemplos de una clase inmersos en grandes áreas de otra clase diferente, HIDER encontrará estas pequeñas reglas al final de la ejecución, o bien incluirá esos ejemplos en una regla más grande aunque de clase distinta, penalizando la precisión. Sin embargo, si tenemos la posibilidad de explorar subconjuntos de reglas, estas pequeñas regiones podrían ser descubiertas a la vez que reglas más grandes y con mayor soporte, como muestra la figura 8.2b. De este modo, esta codificación podría aprovechar mejor las ventajas de la jerarquía y reducir aun más el número de reglas.

Además de las anteriores propuestas, algunas de las cuales ya están siendo abordadas, seguimos trabajando con la empresa ATLANTIC COPPER S.A. analizando conjuntos de datos de más de medio millón de ejemplos.

Todas las líneas de investigación anteriormente mencionadas se enmarcan dentro del proyecto **CICYT TIN-2004-00159**, titulado "MINERVA: TÉCNICAS EMERGENTES DE MINERÍA DE DATOS PARA GRANDES VOLÚMENES DE INFORMACIÓN".

Apéndice A

Datos Experimentales

En este apéndice se muestran algunos datos de interés sobre el estudio experimental de las bases de datos proporcionadas por ATLANTIC COPPER S.A. descrito en el Capítulo 7.

A.1. Parámetros de la planta de ácido

Como se detalló en el estudio, las bases de datos BD18 y BD23 fueron sometidas a un preprocesado simple para eliminar, entre otras deficiencias, aquellos parámetros que a priori no son útiles para el aprendizaje, e incluso pueden llegar a ser perjudiciales. Así, la tabla A.1 muestra los parámetros utilizados durante el estudio, junto a su localización en el sistema de producción y una breve descripción de cada uno. Los parámetros TCM12 y TCM42 aparecen destacados, ya que son los utilizados para el cálculo de la clase. Nótese que los parámetros TCM28, TCM29 y TCM54 sólo están presentes en el estudio de BD18.

Componente	Parámetro	Descripción
HORNO FLASH	TCM4	med. pos. vlv. control tiro
	TCM5	presión h.f.
	TCM7	medida alimentación h.f.
	TCM30	vel. soplante fan253
LINEA 1 Y 2 CONVERTIDORES	TCM2	caudal gas linea 1
	TCM3	caudal gas linea 2
	TCM22	tiro campana CV1
	TCM23	tiro campana CV2
	TCM24	tiro campana CV3
	TCM25	tiro campana CV4
	TCM26	caudal aire soplado CV1
	TCM27	caudal aire soplado CV2
	TCM28	caudal aire soplado CV3 (sólo en BD18)
	TCM29	caudal aire soplado CV4 (sólo en BD18)
CÁMARA DE MEZCLA	**TCM12**	presión .C.M. (Definición de la clase)
	TCM42	sp presión C.M. (Definición de la clase)
PLANTAS RTM1 Y RTM3	TCM10	pos. val. dilución p1 m17
	TCM14	vlv. diluc. 3dk05-1p pi p3
	TCM35	entrada SO_2 an-5701 p.i.p1
	TCM37	vel.sopl.SO_2 1c-400 p.i.p1
	TCM39	sp dilución an-5701 p1 pi
	TCM43	entrada SO_2 q-3q100 p.i.p3
	TCM48	sp dilución q-3q100 p3 pi
	TCM51	SO_2 de cola an-5703 p.i.p1
	TCM44	SO_2 de cola q-3q061 p.i.p3
PLANTA RTM2	TCM11	pos.vlv.fv-201 p.i. p2
	TCM15	caudal entrada p2 f201 pi
	TCM40	entrada SO_2 an-201 p.i.p2
	TCM41	sp dilución an-201 p2 p.i.
	TCM52	SO_2 de cola an-203 p.i.p2
	TCM54	pos.m.marip.eicu-202 pi p2 (sólo en BD18)

Tabla A.1: Parámetros de la Planta de Ácido.

A.2. Resultados: C4.5/C4.5Rules

```
Evaluation on training data (43201 items):

   Before Pruning           After Pruning
  ----------------   --------------------------
  Size      Errors   Size      Errors   Estimate
  4129  582( 1.3%)   3347  790( 1.8%)    ( 6.8%)   <<

    (a)   (b)  <-classified as
   ----  ----
  29764   233  (a): class OK
    557 12647  (b): class NOK
```

Figura A.1: Resultado de C4.5 para BD18.

```
Evaluation on training data (43131 items):
   Before Pruning           After Pruning
  ----------------   --------------------------
  Size      Errors   Size      Errors   Estimate
  2137  257( 0.6%)   1809  338( 0.8%)    ( 3.5%)   <<
    (a)   (b)  <-classified as
   ----  ----
  25213   163  (a): class OK
    175 17580  (b): class NOK
```

Figura A.2: Resultado de C4.5 para BD23.

```
  Final rules from tree 0:
  Rule 1:
        TCM43 <= 5.82
    ->  class NOK  [97.4%]

  Rule 4:
        TCM39 > 6
        TCM39 <= 9.19
        TCM40 <= 9.05
    ->  class NOK  [97.3%]

  Rule 6:
        TCM37 <= 1255.98
        TCM39 > 9.19
        TCM43 > 5.82
    ->  class OK  [74.7%]

  Default class: NOK


  Evaluation on training data (43201 items):

Rule  Size  Error  Used  Wrong            Advantage
----  ----  -----  ----  -----            ---------
   1     1   2.6%    99      1 (1.0%)        0 (0|0)    NOK
   4     3   2.7%    98      1 (1.0%)        0 (0|0)    NOK
   6     3  25.3% 36323   9142 (25.2%)   18039 (27181|9142)     OK

Tested 43201, errors 11958 (27.7%)   <<


      (a)  (b)  <-classified as
     ---- ----
    27181 2816  (a): class OK
     9142 4062  (b): class NOK
```

Figura A.3: Resultado de C4.5Rules para BD18.

```
Final rules from tree 0:

Rule 2:
        TCM35 <= 5.37
        TCM37 <= 1266.3
    ->  class NOK  [98.3%]

Rule 1:
        TCM14 <= 2.35
        TCM37 <= 1266.3
    ->  class OK  [99.7%]

Rule 3:
        TCM10 <= 33.73
        TCM22 <= 0.8
        TCM23 <= -2.65
        TCM30 <= 481.16
        TCM35 > 5.37
        TCM37 <= 1266.3
    ->  class OK  [89.1%]

Default class: OK


Evaluation on training data (43131 items):

Rule  Size  Error  Used  Wrong            Advantage
----  ----  -----  ----  -----            ---------
   2     2   1.7%    80      0 (0.0%)       80 (80|0)   NOK
   1     2   0.3%   537      0 (0.0%)        0 (0|0)    OK
   3     6  10.9%  3111    326 (10.5%)       0 (0|0)    OK

Tested 43131, errors 17675 (41.0%)   <<


      (a)  (b)  <-classified as
     ----  ----
    25376       (a): class OK
    17675   80  (b): class NOK
```

Figura A.4: Resultado de C4.5Rules para BD23.

A.3. Reglas obtenidas por HIDER modo jerárquico

A.3.1. Base de datos BD18

Regla RJ(18)-1:

R1: SI	TCM2 ∈[33556.9, 94089.6]	
	TCM3 ∈[_, 79891.2]	
	TCM4 ∈[43.355, 89.435]	
	TCM5 ∈[-38.355, -2.885]	
	TCM7 ∈[121.635, 159.715]	
	TCM10∈[5.34, 36.925]	
	TCM11∈[9.485, 41.405]	
	TCM14∈[_, 20.955]	
	TCM15∈[40.125, 69.765]	
	TCM22∈[0.635, 2138.54]	
	TCM23∈[-5.03, 0.165]	
	TCM24∈[-3.935, 2.535]	
	TCM25∈[-0.135, _]	
	TCM26∈[_, 33309.4]	
	TCM27∈[1816.09, _]	
	TCM28∈[39.91, 40004.1]	
	TCM29∈[342.825, _]	
	TCM30∈[445.415, 567.68]	
	TCM35∈[7.425, 11.385]	
	TCM37∈[717.29, 1265.17]	
	TCM39∈[9.58, 13.225]	
	TCM40∈[3.11, 12.335]	
	TCM41∈[9.135, 15.915]	
	TCM43∈[5.895, 10.855]	
	TCM44∈[0.007935, _]	
	TCM52∈[0.002125, _]	
	TCM54∈[16.995, _]	ENTONCES OK

Aciertos:	4880
Errores:	854
%Error:	14.9

Figura A.5: Regla R1 obtenida por HIDER en modo jerárquico para BD18.

Regla RJ(18)-2:

SI <R1>…	ENTONCES OK	
SINO, R2: SI	TCM2 ∈[30651.2, 96886]	
	TCM3 ∈[51166.6, 85083]	
	TCM4 ∈[44.485, 73.88]	
	TCM5 ∈[-39.645, -3.705]	
	TCM7 ∈[116.09, 155.615]	
	TCM10∈[10.55, _]	
	TCM11∈[18.595, 42.355]	
	TCM14∈[_, 28.36]	
	TCM15∈[29.615, 69.29]	
	TCM22∈[0.635, 2161.73]	
	TCM23∈[-5.34, -0.00776]	
	TCM24∈[-3.125, 2.99]	
	TCM25∈[-0.115, 0.175]	
	TCM26∈[_, 34071.2]	
	TCM27∈[17588, 41269.9]	
	TCM28∈[8284.98, 40632.1]	
	TCM30∈[472.515, 574.405]	
	TCM35∈[6.575, 11.445]	
	TCM37∈[798.135, 1211.48]	
	TCM39∈[6.495, 13.93]	
	TCM40∈[4.62, 12.635]	
	TCM41∈[_, 12.43]	
	TCM43∈[_, 10.815]	
	TCM44∈[0.007935, _]	
	TCM48∈[9.935, _]	
	TCM51∈[0.001745, _]	
	TCM52∈[0.002185, 0.035]	
	TCM54∈[15.545, _]	ENTONCES OK

Aciertos:	2304
Errores:	403
%Error:	14.9

Figura A.6: Regla R2 obtenida por HIDER en modo jerárquico para BD18.

A.3.2. Base de datos BD23

Regla RJ(23)-1:

R1: SI	TCM2 ∈[_, 95758.7]	
	TCM4 ∈[38.36, 66.05]	
	TCM5 ∈[-28.775, 3.405]	
	TCM7 ∈[_, 151.16]	
	TCM10∈[0.835, 34.87]	
	TCM11∈[_, 39.985]	
	TCM14∈[_, 13.1]	
	TCM15∈[65.2, 71.36]	
	TCM22∈[_, 2.155]	
	TCM23∈[-8.305, 0.705]	
	TCM24∈[0.002495, 0.645]	
	TCM25∈[_, 0.295]	
	TCM26∈[461.765, 41314.1]	
	TCM27∈[32.14, 42343]	
	TCM35∈[5.195, 9.885]	
	TCM37∈[681.63, 1220.81]	
	TCM40∈[5.875, 12.045]	
	TCM41∈[_, 13.5]	
	TCM43∈[5.335, 10.455]	
	TCM44∈[0.007815, _]	
	TCM48∈[9.105, _]	
	TCM51∈[3.165e-06, _]	
	TCM52∈[_, 0.035]	ENTONCES OK

Aciertos:	13123
Errores:	2192
%Error:	14.3

Figura A.7: Regla R1 obtenida por HIDER en modo jerárquico para BD23.

Regla RJ(23)-2:

SI <R1>...	ENTONCES OK	
SINO, R2: SI	TCM2 ∈[29729.5, _]	
	TCM3 ∈[54138, 95197.6]	
	TCM4 ∈[46.125, 90.86]	
	TCM5 ∈[-39.99, 11.655]	
	TCM7 ∈[118.615, 147.595]	
	TCM11∈[_, 39.155]	
	TCM14∈[_, 11.515]	
	TCM15∈[65.255, 70.55]	
	TCM22∈[0.06, 2.025]	
	TCM23∈[-6.395, _]	
	TCM24∈[_, 0.66]	
	TCM25∈[-0.055, _]	
	TCM26∈[13329.2, 44312.5]	
	TCM27∈[13765.3, 46089.1]	
	TCM30∈[421.63, 520]	
	TCM35∈[5.485, 9.955]	
	TCM37∈[817.865, _]	
	TCM39∈[6.165, _]	
	TCM40∈[6.075, 11.765]	
	TCM41∈[_, 14.4]	
	TCM43∈[5.835, 10.005]	
	TCM44∈[0.007885, _]	
	TCM48∈[9.105, _]	
	TCM51∈[3.165e-05, _]	ENTONCES NOK

Aciertos:	8855
Errores:	1207
%Error:	12

Figura A.8: Regla R2 obtenida por HIDER en modo jerárquico para BD23.

A.4. Reglas obtenidas por HIDER en modo no jerárquico

A.4.1. Base de datos BD18

Regla RNJ(18)-1 para la clase OK:

SI	TCM2 ∈[32978, 98685.5]	
	TCM3 ∈[67113.4, 84543.9]	
	TCM4 ∈[38.025, 99.575]	
	TCM5 ∈[-38.405, -0.5]	
	TCM7 ∈[120.205, 169.42]	
	TCM10∈[0.575, 31.44]	
	TCM11∈[14.265, 40.885]	
	TCM14∈[_, 23.935]	
	TCM15∈[41.09, 69.29]	
	TCM22∈[0.585, 2134.75]	
	TCM23∈[_, 0.095]	
	TCM24∈[-3.695, 3.095]	
	TCM25∈[-0.115, 0.175]	
	TCM26∈[_, 36546.4]	
	TCM27∈[23747.4, 40494.2]	
	TCM28∈[5750.82, 38487.4]	
	TCM30∈[460.02, 571]	
	TCM35∈[6.565, 11.435]	
	TCM37∈[779.05, 1271.55]	
	TCM39∈[6.495, 13.93]	
	TCM40∈[4.645, 12.345]	
	TCM43∈[5.895, 10.475]	
	TCM44∈[0.007935, 0.025]	
	TCM48∈[9.76, _]	
	TCM51∈[0.001895, _]	
	TCM52∈[0.002185, _]	
	TCM54∈[16.995, _]	ENTONCES OK

Aciertos:	4417
Errores:	779
%Error:	15

Figura A.9: Regla R1(OK) obtenida por HIDER en modo no jerárquico para BD18.

Regla RNJ(18)-1 para la clase NOK:

SI	TCM2 ∈[26833.7, 95456.9]	
	TCM3 ∈[18285.3, 80627.8]	
	TCM4 ∈[37.005, 87]	
	TCM5 ∈[-35.69, 2.655]	
	TCM7 ∈[70.275, 166.115]	
	TCM10∈[4.405, 36.895]	
	TCM11∈[3.66, 96.595]	
	TCM14∈[_, 22.125]	
	TCM15∈[23.2, _]	
	TCM22∈[0.795, 2158.13]	
	TCM23∈[-5.365, 0.065]	
	TCM24∈[-3.935, 3.84]	
	TCM25∈[-0.115, 0.195]	
	TCM26∈[330.525, 36126.9]	
	TCM27∈[101.975, 41561.9]	
	TCM28∈[1834.83, 43532.4]	
	TCM29∈[_, 366.895]	
	TCM30∈[394.05, 579.625]	
	TCM35∈[6.565, 11.515]	
	TCM37∈[756.3, 1277.78]	
	TCM39∈[_, 14.29]	
	TCM40∈[3.87, 12.375]	
	TCM41∈[_, 15.915]	
	TCM43∈[_, 10.675]	
	TCM44∈[0.007755, _]	
	TCM48∈[9.76, _]	
	TCM51∈[0.002275, _]	
	TCM52∈[0.002105, _]	
	TCM54∈[14.09, 67.345]	ENTONCES NOK

Aciertos:	872
Errores:	113
%Error:	11.5

Figura A.10: Regla R1(NOK) obtenida por HIDER en modo no jerárquico para BD18.

A.4.2. Base de datos BD23

Regla RNJ(23)-1 para la clase OK:

SI	TCM2 ∈[26582.6, 96142.5]	
	TCM3 ∈[_, 88412.7]	
	TCM4 ∈[_, 63.355]	
	TCM5 ∈[-31.43, 18.195]	
	TCM7 ∈[94.335, 141.175]	
	TCM10∈[0.835, 34.225]	
	TCM11∈[0.835, 40.505]	
	TCM14∈[_, 14.395]	
	TCM15∈[64.93, 71.455]	
	TCM22∈[_, 1.305]	
	TCM23∈[-8.305, 0.9]	
	TCM24∈[_, 0.585]	
	TCM25∈[-0.055, 0.315]	
	TCM26∈[486.595, 39095.7]	
	TCM27∈[_, 41646.9]	
	TCM35∈[5.225, 9.795]	
	TCM37∈[669.23, 1269.06]	
	TCM40∈[5.705, 12.045]	
	TCM41∈[_, 13.5]	
	TCM43∈[5.255, 9.875]	
	TCM44∈[0.007915, _]	
	TCM51∈[5.675e-05, _]	ENTONCES OK

Aciertos:	14324
Errores:	2518
%Error:	15

Figura A.11: Regla R1(OK) obtenida por HIDER en modo no jerárquico para BD23.

Regla RNJ(23)-1 para la clase NOK:

SI	TCM2 ∈[74858.5, 104761]	
	TCM3 ∈[57592, 89337]	
	TCM4 ∈[46.125, 74.465]	
	TCM5 ∈[-29.605, 11.545]	
	TCM7 ∈[117.72, 147.075]	
	TCM10∈[10.795, _]	
	TCM11∈[0.835, 38.775]	
	TCM14∈[_, 12.525]	
	TCM15∈[65.43, 71.36]	
	TCM22∈[0.385, 1.875]	
	TCM23∈[-6.395, 0.615]	
	TCM24∈[-0.007745, 0.585]	
	TCM25∈[_, 0.315]	
	TCM26∈[13832, _]	
	TCM27∈[10453.3, 45011.8]	
	TCM30∈[432.635, 530.315]	
	TCM35∈[5.455, 9.805]	
	TCM37∈[838.33, _]	
	TCM40∈[5.965, 11.725]	
	TCM41∈[9.1, 14.4]	
	TCM43∈[5.885, 10.175]	
	TCM48∈[9.145, _]	
	TCM51∈[7.285e-05, _]	
	TCM52∈[_, 0.035]	ENTONCES NOK
(8426 | 1251)		

Aciertos:	8426
Errores:	1251
%Error:	12.9

Figura A.12: Regla R1(NOK) obtenida por HIDER en modo no jerárquico para BD23.

Bibliografía

[1] J. S. Aguilar. *Generación de Reglas Jerárquicas de Decisión con Algoritmos Evolutivos en Aprendizaje Supervisado*. PhD thesis, Universidad de Sevilla, 2001.

[2] J. S. Aguilar, J. C. Riquelme, and M. Toro. Cogito: Un sistema de autoaprendizaje basado en algoritmos genéticos. In *III Jornadas de Informática*, 1997.

[3] J. S. Aguilar-Ruiz, J. Bacardit, and F. Divina. Experimental evaluation of discretization schemes for rule induction. In *Lecture Notes in Computer Science 3102. Springer-Verlag. Genetic and Evolutionary Computation Conference (GECCO'04)*, pages 493–494, Seattle, Washington, EE.UU., June 2004.

[4] J. S. Aguilar-Ruiz, R. Giráldez, and J. C. Riquelme. Natural coding: A more efficient representation for evolutionary learning. In *Lecture Notes in Artificial Intelligence 1415. Springer-Verlag. Genetic and Evolutionary Computation Conference (GECCO'03)*, pages 979–990, Chicago, USA, July 2003.

[5] J. S. Aguilar-Ruiz, J. Riquelme, and M. Toro. Three geometric approaches for representing decision rules in a supervised learning system. In *Genetic and Evolutionary Computation Conference (GECCO '99)*, page 771, Orlando, Florida, EE.UU., 1999.

[6] J. S. Aguilar-Ruiz, J. Riquelme, and M. Toro. Evolutionary learning of hierarchical decision rules. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 33(2):324–331, April 2003.

[7] J. S. Aguilar-Ruiz, J. C. Riquelme, and M. Toro. Decision queue classifier for supervised learning using rotated hyperboxes. In *Progress in Artificial Intelligence*

IBERAMIA'98. Lecture Notes in Artificial Intelligence 1484. Springer-Verlag, pages 326–336, 1998.

[8] J. S. Aguilar-Ruiz, J. C. Riquelme, and M. Toro. A tool to obtain a hierarchical qualitative set of rules from quantitative data. In *Lecture Notes in Artificial Intelligence 1415. Springer-Verlag*, pages 336–346, 1998.

[9] J. S. Aguilar-Ruiz, J. C. Riquelme, and M. Toro. Data set editing by ordered projection. In *Proceedings of the* 14^{th} *European Conference on Artificial Intelligence (ECAI'00)*, pages 251–255, Berlin, Germany, August 2000.

[10] J. S. Aguilar-Ruiz and D. S. Rodríguez. Evolutionary neuroestimation of fitness functions. In *Lecture Notes in Computer Science 2902. Springer-Verlag. Progress in Artificial Intelligence, 11th Protuguese Conference on Artificial Intelligence, (EPIA'03)*, pages 74–83, Beja, Portugal, December 2003.

[11] J. S. Aguilar-Ruiz, R. Ruiz, J. Riquelme, and R. Giráldez. Snn: A supervised clustering algorithm. In *Lecture Notes in Artificial Intelligence 2070, Springer-Verlag, Engineering of Intelligent Systems (IEA-AIE)*, pages 207–216, Budapest, Hungary, June 2001.

[12] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[13] H. Almuallim and T. Dietterich. Learning boolean concepts in presence of many irrelevant features. *Artificial Intelligence*, 69(1-2):279–305, 1994.

[14] J. Antonisse. A new interpretation of schema notation that overturns the binary encoding constraint. In *Third International Conference on Genetic Algorithms*, pages 86–97. Morgan Kaufmann, 1989.

[15] C. Apte and S. Hong. *Predicting equity returns from securities data. Chapter 22*. Fayyad et al., 1996.

[16] J. Bacardit and J. M. Garrell. Evolution of multi-adaptive discretization intervals for a rule-based genetic learning systems. In *Progress in Artificial Intelligence*

IBERAMIA'02. Lecture Notes in Artificial Intelligence 2527. Springer-Verlag, pages 350–360, 2002.

[17] J. Bacardit and J. M. Garrell. Métodos de generalización para sistemas clasificadores de pittsburgh. In *Primer Congreso Español de Algoritmos Evolutivos y Bioinspirados (AEB'02)*, pages 486–493, 2002.

[18] J. Bacardit and J. M. Garrell. Incremental learning for pittsburgh approach classifier systems. In *Segundo Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*, pages 303–311, 2003.

[19] T. Bayes. An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions*, 53:370–418, 1763.

[20] J. L. Bentley and J. H. Friedman. Data structures for range searching. *ACM Computing Surveys*, 11(4):397–409, 1979.

[21] P. Berka and I. Bruha. Empirical comparison of various discretization procedures. Technical report, Laboratory of Intelligent Systems, Prage, 1995.

[22] J. Bezdek and S. E. Pal. *Fuzzy models for pattern recognition*. IEEE Press, 1992.

[23] S. Bhattacharyya and G. Koehler. An analysis of non–binary genetic algorithms with cardinality 2^v. *Complex Systems*, 8:227–256, 1994.

[24] C. Blake and E. K. Merz. UCI repository of machine learning databases, 1998.

[25] P. B. Brazdil, C. Soares, and J. P. da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003.

[26] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Wadsworth International Group, Belmont, CA, 1984.

[27] C. E. Brodley and P. E. Utgoff. Multivariate versus univariate decision trees. Technical Report UM-CS-1992-008, 1992.

[28] C. E. Brodley and P. E. Utgoff. Multivariate decision trees. *Machine Learning*, 19:45–77, 1995.

[29] E. Cantu-Paz and C. Kamath. On the use of evolutionary algorithms in data mining. Book chapter in *Data Mining: A Heuristic Approach*, H. Abbass, R. Sarker, and C. Newton (Eds.), pp. 48-71, 2001.

[30] J. Catlett. *Megainduction: machine learning on very large databases*. PhD thesis, University of Sydney, 1991.

[31] J. Catlett. On changing continuous attributtes into ordered discrete attributes. In *Proceedings of European Working Session on Learning*, pages 164–178, Berlin, 1991. Springer-Verlag.

[32] G. Cervone, L. A. Panait, and R. S. Michalski. The development of the aq20 learning system and initial experiments. In *Proceedings of the International Conference on Intelligent Systems (IIS 2000)*, Poland, July 2001.

[33] C.-L. Chang. Finding prototypes for nearest neighbor classifiers. *IEEE Transactions on Computers*, 23(11):1179–1184, November 1974.

[34] D. K. Y. Chiu, B. Cheung, and A. K. C. Wong. Information synthesis based on hierarchical entropy discretization. *Experimental and Theoretical Artificial Intelligence*, 2:117–129, 1990.

[35] P. Clark and R. Boswell. Rule induction with cn2: Some recents improvements. In *Machine Learning: Proceedings of the Fifth European Conference (EWSL-91)*, pages 151–163, 1991.

[36] P. Clark and T.Ñiblett. The cn2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.

[37] P. Compton and B. Jansen. Knowledge in context: A strategy for expert system maintenance. *J.Siekmann (Ed): Lecture Notes in Artificial Intelligence, Subseries in Computer Sciences*, 406, 1988.

[38] A. L. Comrey. *Manual de Análisis Factorial*. Ediciones Cátedra. Colección Teorema, 1985.

[39] O. Cordón, M. del Jesus, and F. Herrera. Evolutionary approaches to the learning of fuzzy rule-based classification systems.

[40] S. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10:57–78, 1993.

[41] T. M. Cover. Estimation by nearest neighbor rule. *IEEE Transactions on Information Theory*, IT-14:50–55, 1968.

[42] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, IT-13(1):21–27, 1967.

[43] B. V. Dasarathy. *Nearest Neighbor(NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, 1991.

[44] K. A. DeJong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.

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

[46] Devijver and Kittler. On edited nearest neighbor rule. In *k-NN Norm*, 1980.

[47] P. Devijver and J. Kittler. *Statistical Pattern Recognition*. Prentice Hall, 1982.

[48] Y. Dimopoulos and A. Kakas. Learning non-monotonic logic programs: Learning exceptions. In *In N. Lavra and S. Wrobel, editors, European Coonference on Machine Learning, Lecture Notes in Artificial Intelligence 912, Springer Verlag*, pages 122–137, 1995.

[49] F. Divina and E. Marchiori. Evolutionary concept learning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '02)*, pages 343–350, New York, 2002.

[50] C. Domingo, R. Gavaldà, and O. Watanabe. Adaptive sampling methods for scaling up knowledge discovery algorithms. *Data Mining and Knowledge Discovery*, 6:131–152, 2002.

[51] P. Domingos. Rule induction and instance-based learning: A unified approach. In *Proceedings of International Joint Conference on Artificial Intelligence*, 1995.

[52] D. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretisation of continuous features. In *Machine Learning: Proceedings of the Twelfth International Conference*, 1995.

[53] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.

[54] S. Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-6, 4:325–327, 1975.

[55] B. Efron. Estimating the error rate of a prediction rule: improvement on cross-validation. *Journal of the American Statistical Association*, 78:316–330, 1983.

[56] L. J. Eshelman and J. D. Schaffer. Real-coded genetic algorithms and interval-schemata. *Foundations of Genetic Algorithms-2*, pages 187–202, 1993.

[57] U. M. Fayyad and K. B. Irani. Multi-interval discretisation of continuous valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1993.

[58] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Knowledge discovery and data mining: Towards a unifying framework. In *Knowledge Discovery and Data Mining*, pages 82–88, 1996.

[59] F. J. Ferrer, J. S. Aguilar, and J. Riquelme. Non-parametric nearest neighbour with local adaptation. In *10th Portuguese Conference on Artificial Intelligence (EPIA '01), Lecture Notes in Artificial Intelligence, Vol. 2258, Springer-Verlag*, pages 22 – 29, Oporto, December 2001.

[60] R. A. Fisher. The use of multiple measurement in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[61] E. Fix and J. L. Hodges. Discriminatory analysis, nonparametric discrimation consistency properties. Technical Report 4, US Air Force, School of Aviation Medicine, Randolph Field, TX, 1951.

[62] E. Fix and J. L. Hodges. Discriminatory analysis, nonparametric discrimation: small sample performance. Technical Report 11, US Air Force, School of Aviation Medicine, Randolph Field, TX, 1952.

[63] D. B. Fogel. *Evolutionary Computation: Towards a New Philosophy of Machine Learning*. IEEE Press, New York, 1995.

[64] D. H. Foley. Consideration of sample and feature size. *IEEE Trans. Information Theory*, 18:618–626, 1972.

[65] M. Freeston. A general solution of the n-dimensional b-tree problem. In M. J. Carey and D. A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 80–91. ACM Press, 1995.

[66] A. Freitas. A survey of evolutionary algorithms for data mining and knowledge discovery, 2001.

[67] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[68] B. R. Gaines and P. Compton. Induction of ripple down rules. In *Proceedings 5th Australian Joint Conf. on Artiificial Intell., Hobart, Australia, World Scientific*, pages 349–354, 1992.

[69] G. W. Gates. The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, 18:431–433, May 1972.

[70] R. Gavaldà and O. Watanabe. Sequential sampling algorithms: Unified analysis and lower bounds. In *1st Intl. Symposium on Stochastic Algorithms: Foundations and*

Applications (SAGA'01). Lecture Notes in Computer Science 2264. Springer-Verlag, pages 173–187, 2001.

[71] S. Geisser. The predictive sample reuse method with applcations. *Journal of the American Statistical Association*, 70(350):320–328, 1975.

[72] R. Giráldez, J. Aguilar-Ruiz, and J. Riquelme. Discretización supervisada no paramétrica orientada a la obtención de reglas de decisión. In *Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA'01)*, 2001.

[73] R. Giráldez, J. Aguilar-Ruiz, and J. Riquelme. Discretization by maximal global goodness. In *International Conference on Fuzzy Systems and Knowledge Discovery (FSKD'02)*, pages 742–746, Singapore, 2002.

[74] R. Giráldez, J. Aguilar-Ruiz, and J. Riquelme. Discretization oriented to decision rule generation. In *International Conference on Knowledge-Based Intelligent Information & Engineering Systems (KES'02), IOS Press*, pages 275–279, Crema, Italy, 2002.

[75] R. Giráldez, J. Aguilar-Ruiz, and J. Riquelme. Indexación de datos para evaluación rápida de reglas de decisión. In *VII Jornadas de Ingeniería del Software y Bases de Datos (JISBD'02)*, pages 35–44, El Escorial, Madrid, 2002.

[76] R. Giráldez, J. Aguilar-Ruiz, J. Riquelme, and D. Mateos. An efficient data structure for decision rules discovery. In *18th ACM Symposium on Applied Computing, Data Mining Track (SAC'03)*, pages 475–479, Melbourne, Florida, USA, March 2003.

[77] R. Giráldez, J. Aguilar-Ruiz, J. Riquelme, and D. Mateos. Cogito*: Aprendizaje evolutivo de reglas de decisión con codificación netural. In *Segundo Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'03)*, pages 538–547, Gijón, Febrero 2003.

[78] R. Giráldez, J. S. Aguilar-Ruiz, and J. C. Riquelme. Knowledge-based fast evaluation for evolutionary learning. *IEEE Transactions on Systems, Man & Cybernetics – Part C*, (in press), 2004.

[79] R. Giráldez, J. S. Aguilar-Ruiz, and J. C. Riquelme. Una propuesta para reducir el coste de evaluación en aprendizaje evolutivo. In *Tercer Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'04)*, pages 134–140, Córdoba, Febrero 2004.

[80] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[81] D. Gordon. *Active bias adjustment for incremental, supervised concept learning*. PhD thesis, Computer Science Department, University of Maryland, College Park, MD, 1990.

[82] C. Goutte. Note on free lunches and cross validation. *Neural Computation*, 9:1211–1215, 1997.

[83] J. J. Grefenstette, C. L. Ramsey, and A. C. Schultz. Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5(4):355–381, 1990.

[84] A. Guttman. R–Trees : A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD Intl. Conf. on Management of Data*, pages pp. 47–57, 1984.

[85] J. Han and M. Kamber. *Data Mining – Concepts and Techniques*. Morgan Kaufmann, 2001.

[86] P. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(3):515–516, May 1968.

[87] A. Henrich, H.-W. Six, and P. Widmayer. The LSD tree: Spatial access to multidimensional point and nonpoint objects. In P. M. G. Apers and G. Wiederhold, editors, *Proceedings of the Fifteenth International Conference on Very Large Data Bases, August 22-25, 1989, Amsterdam, The Netherlands*, pages 45–53. Morgan Kaufmann, 1989.

[88] F. Herrera, M. Lozano, and J. Verdegay. Generating fuzzy rules from examples using genetic algorithms. *Fuzzy Logic and Soft Computing (B. Bouchon–Meunier, R.R. Yager, L.A. Zadeh Eds.)*, pages 11–20, World Scienific, 1995.

[89] C. A. R. Hoare. Quicksort. *Computer Journal*, 5(1):10–15, 1962.

[90] J. H. Holland. Concerning efficient adaptive systems. *M. C. Yovits, G. T. Jacobi and G. D. Goldstein, editors, Self-Organizing Systems–1962*, pages 215–230, 1962.

[91] J. H. Holland. Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery*, 9:297–314, 1962.

[92] J. H. Holland. *Adaptation in natural and artificial systems*. PhD thesis, University of Michigan, 1975.

[93] J. H. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. *Machine learning: an artificial intelligence approach (vol. 2)*, 1986.

[94] R. C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11:63–91, 1993.

[95] R. C. Holte, L. Acker, and B. W. Poter. Concept learning and the problem of small disjuncts. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 233–240. Morgan Kaufmann, 1989.

[96] E. B. Hunt, J. Marin, and P. J. Stone. *Experiments in induction*. Academic Press, New York, 1966.

[97] H. V. Jagadish. Spatial search with polyhedra. In *Proceedings of the Sixth International Conference on Data Engineering, February 5-9, 1990, Los Angeles, California, USA*, pages 311–319. IEEE Computer Society, 1990.

[98] C. Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 1(13):169–228, 1993.

[99] K. A. Kaufman and R. S. Michalski. Learning from inconsistent and noisy data: The aq18 approach. In *Proceedings of the Eleventh International Symposium on Methodologies for Intelligent Systems*, pages 411–419, 1999.

[100] R. Kerber. Chimerge: Discretization of numeric attributes. In *Proceedings of the* 10^{th} *National Conference on Artificial Intelligence*, pages 123–128. MIT Press, 1992.

[101] D. Kibler and D. W. Aha. Learning representative exemplars of concepts: An initial case study. In *Proceedings of Fourth International Workshop on Machine Learning*, pages 24–30, Irvine, CA, 1987. Morgan Kaufmann.

[102] K. Kira and L. Rendell. A practical approach to feature selection. In *Proceedings of International Conference on Machine Learning*, pages 249–256, Irvine, CA, 1992. Morgan Kaufmann.

[103] S. Kirkpatrick, J. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[104] J. Kivinen, H. Mannila, and E. Ukkonen. Learning hierarchical rule sets. In *Computational Learing Theory*, pages 37–44, 1992.

[105] J. Kivinen, H. Mannila, and E. Ukkonen. Learning rules with local exceptions. Technical report, University of Helsinki, 1993.

[106] J. Kivinen, H. Mannila, E. Ukkonen, and J. Vilo. An ALgorithm for learning hierarchical classifiers. In *European Conference on Machine Learning*, pages 375–378, 1994.

[107] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97:273–324, 1997.

[108] R. Kohavi and M. Sahami. Error-based and entropy-based discretization of continuous features. In *Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 114–119. AAAI Press, 1996.

[109] I. Kononenko. Estimating attributes: analysis and extensions of relief. In *Proceedings of European Conference on Machine Learning*. Springer-Verlag, 1994.

[110] P. A. Lachenbruch. An almost unbiased method of obtaining confidence intervals for the probability of misclassification in discriminant analysis. *Biometrics*, pages 639–645, 1967.

[111] P. A. Lachenbruch and M. R. Mickey. Estimation of error rates in discriminant analysis. *Technometrics*, (1):1–11, 1968.

[112] T. Lim, W. Loh, and Y. Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms, 2000.

[113] H. Liu, F. Hussain, C. Tan, and M. Dash. Discretization: An enabling technique. *Journal of Data Mining and Knowledge Discovery*, 6(4):393–423, 2002.

[114] H. Liu and R. Setiono. Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of the Seventh IEEE International Conference on Tools with Artificial Intelligence*, 1995.

[115] H. Liu and R. Setiono. Feature selection and classification: a probabilistic wrapper approach. In *Proceedings of the IEA-AIE*, 1996.

[116] D. G. Lowe. Similarity metric learning for a variable-kernel classifier. *Neural Computation*, 7(1):72–85, 1995.

[117] W. Maass. Efficient agnostic pac-learning with simple hypotheses. In *Proceedings of the 7th Annual ACM Conference on Computational Learning Theory*, pages 67–75, 1994.

[118] S. W. Mahfoud. *Niching Methods for Genetic Algorithms*. PhD thesis, University of Illinois at Urbana–Champaign, Urbana, May 1995.

[119] M. M. Mano. *Digital design*. Prentice-Hall, 1991.

[120] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, pages 1087–1092, 1953.

[121] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1994.

[122] R. Michalski, J. Carbonell, and T. Mitchell. *A theory and methodology of inductive learning*. Palo Alto: Tioga, 1983.

[123] R. S. Michalski. Discovering classification rules using variable-valued logic system. In *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pages 162–172, Stanford, CA, 1973.

[124] R. S. Michalski and J. B. Larson. Incremental generation of vl1 hypotheses: The underlying methodology and the description of the program aq11. Technical Report UIUCDCS-F-83-905, Computer Science Department, Univ. of Illinois at Urbana-Champaign, 1983.

[125] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The aq15 inductive learning system: An overview and experiments. In *Proceedings of the American Association for Artificial intelligence Conference (AAAI)*, 1986.

[126] H. Minkowsky. *Geometrie der Zahlen*. Teubner, Leizpig, 1896.

[127] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[128] A. W. Moore and M. S. Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1998.

[129] D. F. Morrison. *Multivariate Statistical Methods*. McGraw Hill, 1976.

[130] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P - 826, Caltech Concurrent Computation Program, 1992.

[131] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 1994.

[132] H. S. Nguyen and S. H. Nguyen. Discretization methods with back-tracking, 1997.

[133] J.Ñievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable symmetric multikey file structure. *ACM Transactions on Database Systems, ACM CR 8411-0931*, 9(1), 1984.

[134] B. C. Ooi. Spatial KD–Tree: A data structure for geographic database. In *BWT*, pages 247–258, 1987.

[135] V. Pachón. *Aplicación de Técnicas de Minería de Datos al Proceso de Producción de Ácido*. Memoria de Investigación presentada en la Universidad de Sevilla para la obtención del Diploma de Estudios Avanzados, 2004.

[136] V. Pachón., J. Mata, F. Roche, J. C. Riquelme, and J. M. Tejera. Practical application of kdd techniques to an industrial process. In *6th International Conference on Enterprise Information Systems (ICEIS'04)*, pages 309–314, 2004.

[137] B. Pfahringer. Compression-based discretization of continuous attributes. In *Proceedings of the 20th International Conference on Machine Learning*. Morgan Kaufmann, 1995.

[138] R. Pérez. *Aprendizaje de reglas difusas usando algoritmos genéticos*. PhD thesis, Universidad de Granada, 1997.

[139] J. R. Quinlan. Discovering rules by induction from collections of examples. In *Expert Systems in the Micro-Electronic Age*, pages 168–201, Edinburgh, 1979. Edinburgh University Press.

[140] J. R. Quinlan. Learning efficient classification procedures and their application to chess end games. In *Machine Learning: An Artificial Intelligence Approach*, Palo Alto, Tioga, 1983.

[141] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[142] J. R. Quinlan. Generating production rules from decision trees. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 304–307, Milan, Italy, 1987.

[143] J. R. Quinlan. Rule induction with statistical data - a comparison with multiple regression. *Journal of the Operational Research Society*, 38:347–352, 1987.

[144] J. R. Quinlan. An empirical comparison of genetic and decision trees classifiers. In *Proceedings of the 5th International Joint Conference on Machine Learning*, pages 135–141, 1988.

[145] J. R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo, California, 1993.

[146] J. R. Quinlan. The minimum description length principle and categorical theories. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 233–241, 1994.

[147] J. R. Quinlan. Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[148] R. Quinlan. See5.0 (http://www.rulequest.com), 1998-2001.

[149] I. Rechenberg. Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution. 1973.

[150] M. Richeldi and M. Rossotto. Class–driven statistical discretization of continuous attributes. In *European Conference on Machine Learning (ECML'95). Lecture Notes in Artificial Intelligence 914. Springer-Verlag*, pages 335–338, 1995.

[151] R. L. Riolo. *Empirical studies of default hierarchies and sequences of rules in learning classifier systems*. PhD thesis, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, EE.UU., 1988.

[152] J. C. Riquelme and J. S. Aguilar. Codificación indexada de atributos continuos para algoritmos evolutivos en aprendizaje supervisado. In *Congreso Español de Algoritmos Evolutivos y Bioinspirados*, pages 161–167, Mérida, 2002.

[153] J. C. Riquelme, J. S. Aguilar, and M. Toro. Cogito 2.0: Una herramienta para obtener un clasificador jerárquico en aprendizaje supervisado. In *VII Conferencia de la Asociación Española para la Inteligencia Artificial*, pages 489–498, 1997.

[154] J. C. Riquelme, J. S. Aguilar, and M. Toro. A ga-based tool to obtain a hierarchical classifier for supervised learning (in spanish). *Revista Iberoamericana de Inteligencia Artificial*, 1(5), 1998.

[155] J. C. Riquelme, J. S. Aguilar, and M. Toro. A decision queue based on genetic algorithms: axis-paralle classifier versus rotated hyperboxes. In *Computational Intelligence and Applications*, pages 38–43, Atenas, 1999.

[156] J. C. Riquelme, J. S. Aguilar, and M. Toro. Discovering hierarchical decision rules with evolutive algorithms in supervised learning. *International Journal of Computers, Systems and Signals*, 1(1):73–84, 2000.

[157] J. C. Riquelme, F. J. Ferrer, and J. S. Aguilar. Búsqueda de un patrón para el valor de k en k-nn. In *IX Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA '01), Libro de Actas, Volumen I, A. Bahamonde, R. Otero (eds)*, pages 63–72, Gijón, Noviembre 2001.

[158] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[159] G. Ritter, H. Woodruff, S. Lowry, and T. Isenhour. An algorithm for a selective nearest neighbor decision rule. *IEEE Transactions on Information Theory*, 21(6):665–669, 1975.

[160] R. L. Rivest. Learning decision lists. *Machine Learning*, 1(2):229–246, 1987.

[161] J. T. Robinson. The K–D–B–Tree: A search structure for large multidimensional dynamic indexes. In Y. E. Lien, editor, *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, Ann Arbor, Michigan, April 29 - May 1, 1981*, pages 10–18. ACM Press, 1981.

[162] S. Ruggieri. Efficient C4.5. Technical Report TR-00-01, 2, 2000.

[163] S. L. Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, 6:277–309, 1991.

[164] T. Scheffer. Algebraic foundations and improved methods of induction or ripple-down rules. In *Proceedings of the 2 nd Pacific Rim Knowledge Acquisition Workshop*, 1996.

[165] H.-P. Schwefel. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, Technical University of Berlin, Department of Process Engineering, Berlin, Germany, 1975.

[166] R. Setiono and H. Liu. A probabilistic approach to feature selection - a filter solution. In *Proceedings of International Conference on Machine Learning*, pages 319–327, 1996.

[167] K. Shim. *SIGKDD Explorations*, volume 2(2). ACM Press, December 2000.

[168] S. F. Smith. *A learning system based on genetic adaptive algorithms*. PhD thesis, Department of Computer Science, University of Pittsburgh, 1980.

[169] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society B*, 36:111–147, 1974.

[170] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9, 1989.

[171] K. M. Ting. Discretization of continuous-valued attributes and instance-based learning. Technical Report 491, University of Sydney, 1994.

[172] G. Venturini. SIA: a supervised inductive algorithm with genetic search for learning attributes based concepts. In *Proceedings of European Conference on Machine Learning*, pages 281–296, 1993.

[173] S. Vere. Multilevel counterfactuals for generalizations of relational concepts and productions. *Artificial Intelligence*, 14:139–164, 1980.

[174] M. Vose and A. Wright. The simple genetic algorithm and the walsh transform: Part i, theory. *Evolutionary Computation*, 6(3):253–273, 1998.

[175] M. Vose and A. Wright. The simple genetic algorithm and the walsh transform: Part ii, the inverse. *Evolutionary Computation*, 6(3):275–289, 1998.

[176] M. S. Weiss and C. A. Kulikowski. *Computer Systems that Learn*. Morgan Kaufmann Publishers, Inc., 1991.

[177] D. Wettschereck and T. G. Dietterich. An experimental comparison of nearest neighbor and nearest hyperrectangle algorithms. *Machine Learning*, 19(1):5–28, 1995.

[178] D. Wilson. Asymtotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man and Cybernetics*, 2(3):408–421, July 1972.

[179] D. R. Wilson and T. R. Martinez. Reduction techniques for instance–based learning algorithms. *Machine Learning*, 38(3):257–286, 2000.

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

[181] J. Wnek, J. Sarma, A. Wahab, and R. S. Michalski. *Comparing learning paradigms via diagramatic visualization*. Amsterdam, North Holland, 1990.

[182] A. K. C. Wong and D. K. Y. Chiu. Synthesizing statistical knowledge from incomplete mixed-mode data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(6):205–218, 1987.

[183] L. A. Zadeh. Fuzzy set. *Information and Control*, 8:338–353, 1965.