UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Obtención de reglas de clasificación utilizando estrategias adaptativas
Autor: Augusto Villa Monte
Director: Laura Cristina Lanzarini
Carrera: Licenciatura en Sistemas

Resumen

En la actualidad, la mayoría de los procesos cuentan con información histórica lo suficientemente grande como para que sea difícil procesarla en forma manual.
La Minería de Datos, una de las etapas más importantes del proceso de Extracción de Conocimiento, cuenta con un conjunto de técnicas capaces de modelizar y resumir esos datos históricos, facilitando su comprensión y ayudando a la toma de decisiones en situaciones futuras.
Esta tesina presenta una nueva técnica de Minería de Datos, llamada SOM+PSO, capaz de construir, a partir de la información disponible, un conjunto reducido de reglas de clasificación sencillas de cuya lectura se desprenden las relaciones más importantes entre las características registradas.
También se detallan los resultados obtenidos y se los compara contra un método existente en la literatura, el cual fue seleccionado por considerarlo un referente en el tema.

Palabras Claves

Minería de Datos, Reglas de Clasificación, Estrategias Adaptativas, Optimización mediante Cúmulo de Partículas, Mapas Auto-Organizativos.

Conclusiones

- *Esta tesina presenta un nuevo método capaz de generar en forma automática, a partir de la información disponible, un conjunto de reglas de clasificación de fácil lectura e interpretación.*
- *Los resultados obtenidos al aplicarla sobre 19 bases de datos han sido satisfactorios.*
- *Su comparación con otro método existente permite afirmar que el modelo generado es más simple.*

Trabajos Realizados

- *Análisis de los conceptos principales de la Minería de Datos.*
- *Descripción de los problemas de clasificación y de las técnicas utilizadas para obtener reglas.*
- *Estudio de técnicas aplicables a problemas de agrupamiento y de optimización.*
- *Definición, desarrollo e implementación del método SOM+PSO y análisis de los resultados obtenidos.*

Trabajos Futuros

- *Identificar las características comunes en los datos de entrada utilizando una red neuronal SOM dinámica. De esta forma no sería necesario especificar de antemano la cantidad de agrupamientos a formar.*
- *Utilizar una función multi objetivo para medir el desempeño de las reglas durante el proceso de obtención. Esto permitiría contar con un conjunto de soluciones óptimas (según algunos criterios) en lugar de disponer de una única opción.*

Fecha de la presentación: FEBRERO DE 2013

Obtención de reglas de clasificación utilizando estrategias adaptativas

Autor: Augusto Villa Monte

Directora: Prof. Lic. Laura C. Lanzarini

Tesina de Licenciatura en Sistemas

Facultad de Informática

UNIVERSIDAD NACIONAL DE LA PLATA

22 de febrero de 2013

Resumen

En la actualidad, la mayoría de los procesos cuentan con información histórica lo suficientemente grande como para que sea difícil procesarla en forma manual.

La Mineria de Datos, una de las etapas más importantes del proceso de Extracción de Conocimiento, cuenta con un conjunto de técnicas capaces de modelizar y resumir esos datos históricos, facilitando su comprensión y ayudando a la toma de decisiones en situaciones futuras.

Esta tesina presenta una nueva técnica de Minería de Datos capaz de construir, a partir de la información disponible, un conjunto reducido de reglas de clasificación sencillas de cuya lectura se desprenden las relaciones más importantes entre las características registradas.

También se detallan los resultados obtenidos y se los compara contra un método existente en la literatura, el cual fue seleccionado por considerarlo un referente en el tema.

Palabras Claves: Minería de Datos, Reglas de Clasificación, Estrategias Adaptativas, Optimización mediante Cúmulo de Partículas, Mapas Auto-Organizativos.

Publicaciones relacionadas

- **Técnicas de Minería de Datos aplicadas al diseño de un Curso de Estadística**. Publicado en los Proceedings del *V Congreso de Tecnología en Educación y Educación en Tecnología* (TE&ET 2010). ISBN 978-987-1242-42-9, pp. 365-372. Mayo de 2010.
- **E-mail processing using data mining techniques**. Publicado en el Libro *Computer Science & Technology Series: XVI Argentine Congress of Computer Science - Selected Papers* por Edulp. ISBN 978-950-34-0757-8, pp. 109-120. Octubre de 2011.
- **E-Mail Processing with Fuzzy SOMs and Association Rules**. Publicado en el *Journal of Computer Science & Technology* (JCS&T). ISSN 1666-6038, Vol. 11, Nro. 1, pp. 41-46. Abril de 2011.
- **Obtención de reglas de clasificación usando SOM+PSO**. Publicado en los Proceedings del *XIII Workshop de Agentes y Sistemas Inteligentes* - CACIC 2012. ISBN 978-987-1648-34-4, pp. 210-219. Octubre de 2012.

Prefacio

La Minería de Datos es un área de investigación que en los últimos años ha llamado la atención de distintos sectores. Empresarios y académicos, por motivos muy diferentes han contribuido al desarrollo de distintas técnicas capaces de resumir la información disponible con el objetivo de extraer conocimiento nuevo.

Si bien existen distintos modelos, aquellos que posean la capacidad de explicarse a si mismos, serán los elegidos por quienes deben tomar decisiones.

Por tal motivo, las reglas, es decir, afirmaciones del tipo:

```
SI (ocurre esto) ENTONCES (pasa aquello otro)
```

son las preferidas a la hora de caracterizar esa enorme cantidad de datos históricos que fueron guardados automáticamente.

Lamentablemente, la mayoría de los métodos existentes, cubre los ejemplos de una base de datos con un conjunto de reglas tan extenso y complejo que pese a tener la forma `SI-ENTONCES`, se torna prácticamente ilegible.

Por tal motivo, esta tesina propone un método de obtención de reglas de clasificación que posee dos características fundamentales: es de cardinalidad baja y las reglas generadas poseen un antecedente reducido. Los resultados obtenidos al aplicar el método propuesto sobre un conjunto de bases de datos de prueba mostrarán que estas características siempre se verifican aunque en alguna ocasiones las reglas obtenidas pierden ligeramente su precisión. Se ha considerado que dicha pérdida es un error aceptable dada la simplicidad del modelo obtenido.

Esta tesina está organizada de la siguiente forma:

- En el capítulo 1 se desarrollan los conceptos más importantes de la Minería de Datos y los principales modelos que pueden obtenerse. El objetivo es desarrollar un marco teórico mínimo dentro del cual se encuentra inmersa la propuesta de esta tesis.

- Luego, en el capítulo 2 se desarrolla sólo uno de los modelos descriptos en forma general en el capítulo anterior: las reglas de clasificación. Esto tiene que ver con que ellas son el tipo de modelo que se crea con el método propuesto en esta tesina. Por tal motivo, este capítulo resume el estado del arte en el tema e incluye un detalle del método seleccionado para realizar las comparaciones finales.

- Los capítulos 3 y 4 introducen los temas relacionados con Redes Neuronales y Técnicas de optimización. En especial el capítulo 3 describe una arquitectura de Red Neuronal competitiva y el capítulo 4 enfatiza en la optimización por cúmulos de partículas. De la combinación de ambas estrategias surge el método propuesto y por tal motivo fueron desarrollados en esta tesina.

- El capítulo 5 expone el método propuesto. Se trata de un nuevo método adaptativo capaz de extraer un conjunto de reglas reducido con pocas condiciones en sus antecedentes y con una cobertura aceptable para una cota de error preestablecida.

También se detallan los resultados obtenidos y se los compara contra el método detallado en el capítulo 2.

- Finalmente el capítulo 6 contiene las conclusiones finales y algunas líneas de trabajo futuras.

Índice general

1. Extracción de Conocimiento **9**

1.1. Introducción 9
1.2. Minería de Datos y $\mathcal{KDD}$ 10
1.3. Disciplinas relacionadas y aplicaciones 10
1.4. Fases del proceso de $\mathcal{KDD}$ 13
1.4.1. Comprensión del Dominio 13
1.4.2. Recopilación e integración de datos 13
1.4.3. Preparación de los datos 15
1.4.4. Modelado 17
1.4.5. Interpretación y evaluación 18
1.4.6. Difusión y uso de los resultados 19
1.4.7. Implementación de medidas 20
1.4.8. Medición de resultados 20
1.5. Tareas de Minería de Datos 20
1.5.1. Tareas Predictivas 20
1.5.2. Tareas Descriptivas 21
1.6. Conclusiones 22

2. Técnicas de $\mathcal{DM}$ para Clasificación **25**

2.1. Introducción 25
2.2. Arboles de clasificación 26
2.2.1. Criterio de selección de atributos 28
2.2.2. Poda y reestructuración 30
2.2.3. Calidad del modelo 30
2.2.4. Algoritmos principales 31
2.3. Modelos basados en reglas 32
2.3.1. Reglas de asociación 33
2.3.2. Reglas de clasificación 35
2.3.3. Lista de decisión 36

2.3.4. Relación entre reglas y árboles . . . 37
2.3.5. Métricas para reglas . . . 38
2.3.6. Métodos clásicos . . . 40
2.4. Conclusiones . . . 49

3. Clustering con Redes Neuronales **53**
3.1. Introducción . . . 53
3.2. Problema de clustering . . . 53
3.3. Distancias y similitudes . . . 56
3.3.1. Propiedades generales . . . 56
3.3.2. Medidas de distancia . . . 58
3.4. Clasificación a través del agrupamiento . . . 59
3.5. Redes Neuronales y clustering . . . 60
3.6. Mapas auto-organizativos . . . 60
3.6.1. Descripción del modelo . . . 61
3.7. Mapas auto-organizativos dinámicos . . . 63
3.7.1. Métodos existentes . . . 63
3.8. Conclusiones . . . 65

4. Técnicas de Optimización **67**
4.1. Introducción . . . 67
4.2. Clasificación . . . 67
4.2.1. Metaheurísticas Basadas en Trayectoria . . . 70
4.2.2. Metaheurísticas Basadas en Población . . . 72
4.3. PSO - Particle Swarm Optimization . . . 73
4.3.1. Tipos de Algoritmos basados en PSO . . . 77
4.3.2. Topologías de Cúmulos de Partículas . . . 77
4.4. PSO Binario . . . 78
4.5. PSO Binario con control de velocidad . . . 80
4.6. Conclusiones . . . 81

5. SOM+PSO: método propuesto **83**
5.1. Representación de Reglas . . . 83
5.1.1. Enfoque seleccionado . . . 83
5.1.2. Opciones para la representación de una regla . . . 84
5.2. Obtención de reglas de clasificación con PSO . . . 88
5.3. Método de obtención de reglas propuesto . . . 93
5.4. Resultados obtenidos . . . 94

5.5. Conclusiones 96

6. Conclusiones generales y trabajos futuros **99**

A. RapidMiner **101**

Indice de figuras **105**

Indice de tablas **109**

Bibliografía **111**

Capítulo 1

Extracción de Conocimiento

1.1. Introducción

El proceso de Extracción de Conocimiento a partir de Bases de Datos ($\mathcal{KDD}$, por las siglas en inglés de *Knowledge Discovery from Databases*), en [1] se define como el proceso no trivial de identificar patrones válidos, novedosos, potencialmente útiles y comprensibles a partir de los datos.

Se trata de un proceso que, a diferencia de los sistemas tradicionales de explotación de datos basados en la existencia de hipótesis o modelos previos, busca el descubrimiento del conocimiento sin una hipótesis preconcebida.

Es importante remarcar que bajo el enfoque tradicional, generalmente estadístico, es preciso definir las hipótesis que se desean verificar. En otras palabras, habitualmente alguien debe sugerir la respuesta esperada para luego contrastarla con la información de la base de datos. Este no es el enfoque deseado en esta tesina.

Por ejemplo, suponga que dispone de información académica y personal de los alumnos de cierta Universidad y está interesado en identificar las características principales que determinan la deserción durante los primeros años de una carrera. El enfoque convencional opera sobre la verificación de supuestos establecidos de antemano. Por ejemplo, luego de realizar varias consultas SQL sobre la información disponible, utilizando este enfoque podría concluir que un cierto porcentaje de los alumnos que trabajan y que luego de un cierto tiempo no han logrado un significativo avance en su carrera, abandonan sus estudios. La determinación del avance académico del alumno no es algo simple de establecer a priori. Por su parte el proceso de $\mathcal{KDD}$ podría permitirle obtener una expresión de la forma "SI $(TRABAJA = SI)$ Y $(ESTADO\,CIVIL = CASADO)$ Y $(\%\,MATERIAS\,APROBADAS < 0{,}3)$ ENTONCES $(ABANDONA)$". Más allá de la similitud con el análisis convencional, es importante distinguir que la elección de las condiciones a tener en cuenta quedan determinadas por el método y no por criterios preestablecidos que se espera verificar. A medida que se avance sobre la descripción de las técnicas involucradas, las ventajas de la aplicación del proceso de $\mathcal{KDD}$ irán tomando importancia.

Es importante remarcar que contar con herramientas que permitan detectar automática-

mente nuevas asociaciones entre patrones y que ayuden a visualizar la manera en la que se organizan los datos será de suma utilidad en cualquier proceso de toma de decisiones.

1.2. Minería de Datos y $\mathcal{KDD}$

El proceso de $\mathcal{KDD}$ consta de una serie de fases que definen la metodología a utilizar para descubrir conocimiento útil desde las bases de datos. La secuencia de estas fases no es estricta y frecuentemente hay retroalimentación entre ellas, dependiendo del resultado de cada fase. La figura 1.1 muestra las interrelaciones entre las fases y los resultados de las mismas, dejando en claro la naturaleza cíclica del proceso. La metodología del proceso global de $\mathcal{KDD}$ provee una representación completa del ciclo de vida de un proyecto de Minería de Datos.

La Minería de Datos ($\mathcal{DM}$, del inglés *Data Mining*) es la etapa más relevante del $\mathcal{KDD}$ e involucra las técnicas necesarias para la construcción de modelos a partir de la información disponible. Dichas técnicas tienen la probada capacidad de descubrir reglas y/o patrones significativos de información que puedan ayudar tanto en el diagnóstico correcto de un problema como en la formulación de las estrategias para su solución [2].

La importancia que tiene la etapa de generación del modelo dentro de todo el proceso ha llevado a que, en general, se denomine Minería de Datos al proceso completo de Extracción de Conocimiento. Si bien esto no es correcto, es una interpretación que las empresas han impuesto en referencia a este tema.

1.3. Disciplinas relacionadas y aplicaciones

Las áreas en las que se emplean métodos de $\mathcal{DM}$ son cada día más variadas. Se encuentran ejemplos de casos de éxito en aplicaciones bancarias, financieras, científicas, empresariales, económicas, industriales, entre tantas otras [3]. La figura 1.2 muestra las disciplinas más influyentes alrededor de las cuales la Minería de Datos se ha desarrollado.

Desde hace ya varios años, en el Instituto de Investigación en Informática LIDI, se llevan a cabo trabajos, en su mayoría basados en Redes Neuronales y Técnicas de Optimización, los cuales demuestran la vinculación entre la Minería de Datos y distintas disciplinas relacionadas. A continuación se describen algunos de ellos:

- Se desarrolló para la Facultad de Ciencias Médicas de la UNLP un sistema de reconocimiento de movimiento ocular [4], el cual funciona actualmente en consultorios privados. Este sistema de apoyo para la toma de decisiones de especialistas en la disciplina permite detectar alteraciones del equilibrio favoreciendo el diagnóstico satisfactorio de la patología del paciente.

- En el marco del Programa ProSanE impulsado por el Ministerio de Salud de la Nación, se analizó información referida al estado de salud de alumnos de distintas escuelas de la provincia de Buenos Aires. Utilizando técnicas de $\mathcal{DM}$ se detectaron patrones sobre los cuales se elaboró un informe y se hicieron algunas

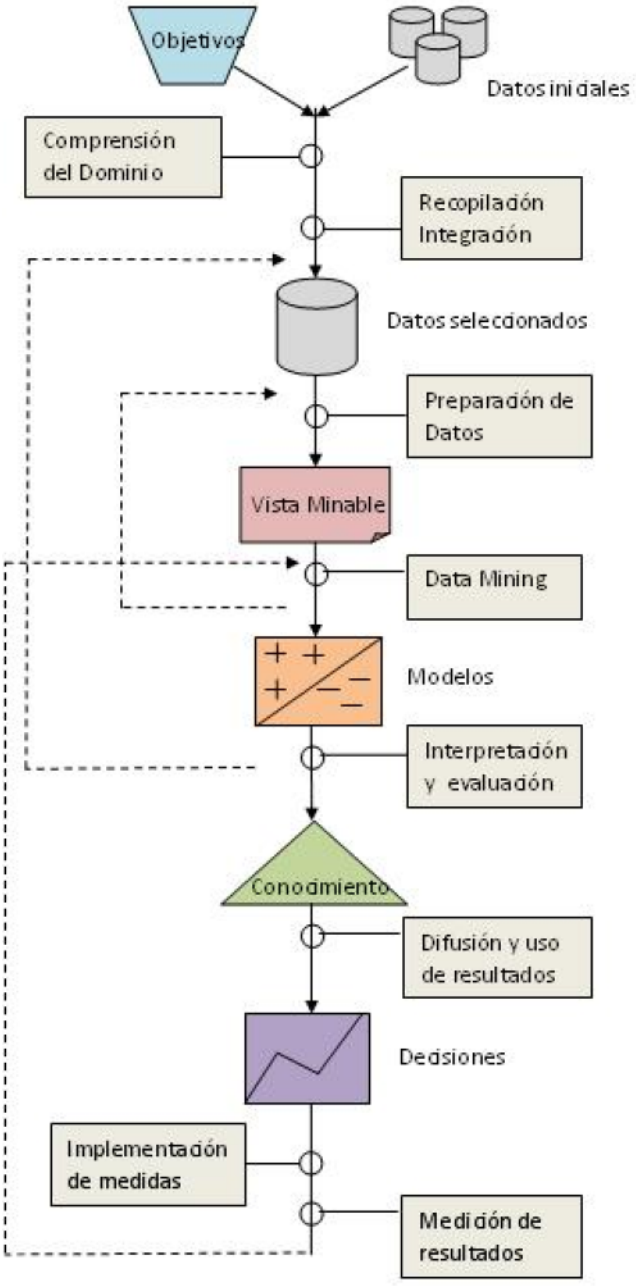


Figura 1.1: Fases que componen el proceso de $\mathcal{KDD}$

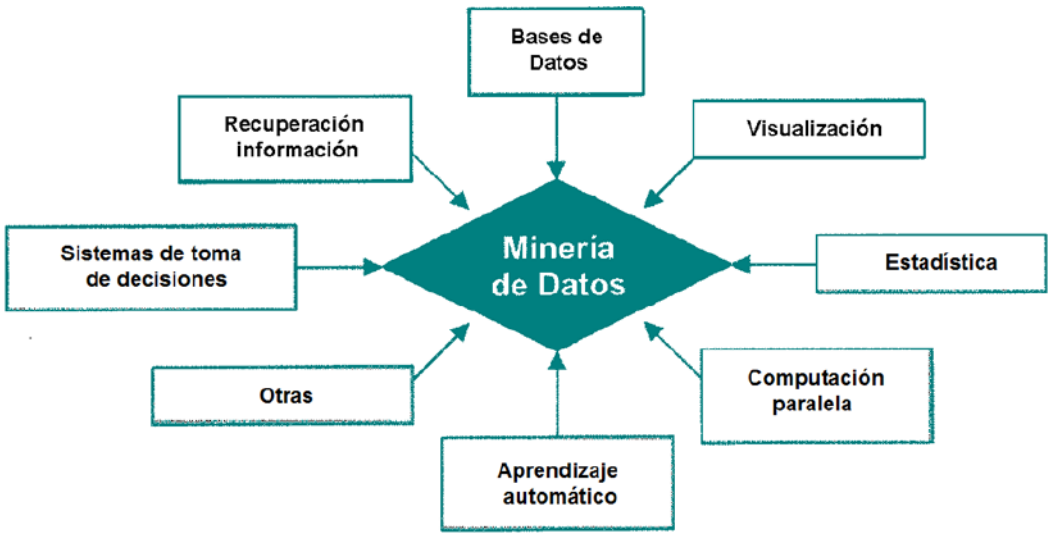


Figura 1.2: Disciplinas que contribuyen a la Minería de Datos

recomendaciones tales como si existe la necesidad de crear una salita de primeros auxilios en determinados centros geográficos.

- Utilizando una técnica de optimización, se desarrolló la asignación automática de móviles para la Unidad Coronaria de Quilmes, con el objetivo de mejorar las prestaciones del operador de cabina al momento de asignar los móviles para las emergencias médicas [5]. De esta manera se disminuyeron los tiempos de espera y se minimizaron los costos, haciendo más eficiente el servicio ofrecido.

- En cuanto a la recuperación de información se trabajó con el Programa PACENI, aplicando técnicas de $\mathcal{DM}$ para procesar las comunicaciones entre alumnos y tutores a través de una plataforma a distancia [6] [7]. Esto permitió detectar los temas que motivaron las consultas más frecuentes con el objetivo de mejorar la atención de los alumnos.

- En aprendizaje automático, se trabajó en biometría reconociendo personas a través de su voz [8] o utilizando imágenes de su rostro [9], en reconocimiento de objetos en video [10], en obtención de árboles y reglas [11] [12], en robótica evolutiva construyendo controladores [13].

Si bien es envidente que la Minería de Datos tiene aplicaciones diversas alrededor de ella, en los últimos años ha evolucionado hacia una disciplina que se encarga de la modelización predictiva, *forecasting* (uso de datos históricos para determinar tendencias a futuro) y optimización de todo tipo de fenómenos y problemas. Se trata de construir modelos a partir de grandes cantidades de datos, análisis inteligente, aprendizaje automático y métodos estadísticos multivariados, que permiten analizar bases de datos con muchas variables (alta dimensionalidad y complejidad de los datos).

Las técnicas de $\mathcal{DM}$ apuntan a transformar datos en información para la toma de decisiones. Dependen en gran medida de los datos disponibles y de la preparación adecuada que se les dé a los mismos, de manera de poder utilizar diferentes algoritmos y metodologías de descubrimiento.

El objetivo de las técnicas de $\mathcal{DM}$ es extraer conocimiento desde los datos y ese conocimiento constituye el modelo de los datos analizados. Los patrones pueden ser utilizados para predecir observaciones futuras o explicar observaciones pasadas, capacidades fundamentales para mejorar el comportamiento en relación a un fenómeno como la deserción universitaria, la producción de infartos cardíacos, etc.

1.4. Fases del proceso de $\mathcal{KDD}$

Tal como se vió en la figura 1.1, el proceso de $\mathcal{KDD}$ se organiza en una secuencia iterativa de etapas. En esta sección se presenta cada una de ellas introduciendo sus principales conceptos.

1.4.1. Comprensión del Dominio

Una fase importante de cualquier proyecto de $\mathcal{DM}$ consiste en entender sus objetivos desde una perspectiva de la organización para poder desarrollar un plan preliminar en pos de los mismos.

Para entender qué datos deben ser analizados y de qué manera hay que hacerlo, es vital poseer un completo entendimiento del problema para el cual se está buscando una solución. Esta fase involucra pasos clave como determinar los objetivos, comprender la situación, determinar el papel de la Minería de Datos en el proyecto y visualizar un plan de trabajo.

Últimamente la importancia de esta fase se ha incrementado debido a la tendencia a acercar las técnicas de $\mathcal{DM}$ a la realidad de los negocios, aprovechando el conocimiento de los expertos sobre el dominio. Esta metodología, conocida como *Domain Driven Data Mining* [14], permite reconocer subjetivamente los modelos de interés para los usuarios.

1.4.2. Recopilación e integración de datos

Esta fase se inicia con la obtención de los datos. Una vez conseguida la información se procede a familiarizarse con ella e identificar su procedencia. En esta etapa se trabaja en recolectar los datos, describirlos, explorarlos y verificar su calidad.

Comenzando con la tarea de recopilación puede decirse que, las bases de datos y las aplicaciones basadas en el procesamiento transaccional de datos en línea (OLTP, del inglés *On Line Transaction Processing*) cubren las necesidades diarias de información de una organización, pero no siempre son suficientes para funciones como el análisis, planificación y predicción. Por ello, en algunos casos es necesario obtener datos de otras áreas de la organización. Incluso puede ocurrir que algunos datos necesarios para el

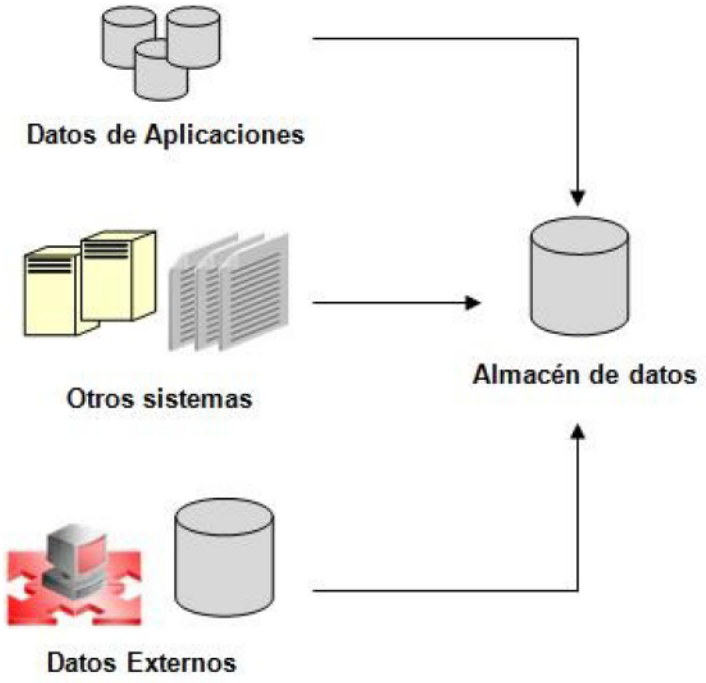


Figura 1.3: Integración en un almacén de datos

análisis no hayan sido recolectados hasta el momento y en estos casos, puede ser necesario obtener datos desde bases de datos públicas (datos censales, datos demográficos, etc.) o privadas (de bancos, compañías de servicios, etc.). Cuando se utilizan fuentes de datos de diferentes orígenes, se enfrentan nuevos problemas entre los que se encuentran, por ejemplo, los diferentes formatos de registro, los distintos grados de agregación en los datos, las claves primarias no coincidentes, etc.

De estos hechos se desprende entonces, la necesidad de integrar todos los datos de los que se dispone, dando lugar a tecnologías de almacenes de datos (*Data Warehouses*). Un almacén de datos (ver figura 1.3) recopila información de diferentes fuentes y las unifica en un único repositorio con un diseño que se adapta perfectamente a las consultas estadísticas y al análisis de datos. Los almacenes de datos se modelan con una estructura de base de datos multidimensional, donde cada dimensión corresponde a un atributo o conjunto de atributos que caracterizan unos hechos o medidas agregadas como puede ser, por ejemplo, la cantidad de productos comprados por una persona en un determinado mes. Esta representación multidimensional es la adecuada para el procesamiento analítico en línea (OLAP, del inglés *On-Line Analytical Processing*).

Los almacenes de datos permiten al usuario obtener informes agregados por diferentes dimensiones en tiempo real, a partir de la información detallada almacenada en los mismos. Las herramientas OLAP pueden utilizarse también para comprobar patrones mediante un proceso deductivo, en cambio la Minería de Datos es un proceso inductivo que permite encontrar dichos patrones. Un almacén de datos es una herramienta muy útil para su uso en las primeras etapas del proceso de $\mathcal{KDD}$, para explorar y comprender

los datos, favoreciendo el proceso de descubrimiento de conocimiento de las etapas posteriores. Se puede decir que un almacén de datos es muy aconsejable para la Minería de Datos, pero no imprescindible en algunos casos, en particular cuando el volumen de datos no es muy grande ya que se puede trabajar directamente con los datos originales.

También es en esta etapa donde el analista debe sopesar el nivel de agregación de los datos y analizar la legalidad del uso de los mismos, para tomar decisiones al respecto.

1.4.3. Preparación de los datos

La calidad del conocimiento descubierto no depende únicamente del algoritmo de $\mathcal{DM}$ utilizado, sino también de la calidad de los datos involucrados [2]. Por eso, después de la recopilación, el próximo paso en el proceso de $\mathcal{KDD}$ es seleccionar y preparar el subconjunto de datos que se van a minar, los cuales constituyen lo que se conoce como vista minable.

La necesidad de construir una vista minable surge principalmente del hecho que la mayoría de los métodos de $\mathcal{DM}$ sólo trabajan con una única tabla. También se debe considerar que dada una base de datos relacional con muchas tablas vinculadas a través de claves foráneas, existen muchas maneras de relacionarlas. La vista minable deja en claro las relaciones que se quieren definir para trabajar sobre ellas.

Esta fase que cubre todas las actividades para construir el conjunto final de datos que será utilizado en las herramientas de modelado, incluye la selección de las tablas (o archivos), registros y atributos, así como la transformación y limpieza de los datos. Las operaciones del lenguaje de consulta estructurado (SQL del inglés *Structured Query Language*) son un estándar que se adapta perfectamente para esta tarea.

En esta etapa se utilizan técnicas de limpieza, transformación y reducción de dimensionalidad que aseguren la calidad de los datos y su adecuación para ser utilizados por las herramientas de modelado.

Algunos de los problemas que se atacan en esta fase son:

- *Presencia de valores que no se ajustan al comportamiento general de los datos (outliers)*
  Pueden deberse a errores o no, siendo valores correctos muy diferentes al resto. Algunos algoritmos de $\mathcal{DM}$ los ignoran, otros los descartan y otros son muy sensibles a ellos cuyo resultado se ve perjudicado por dichos valores. De todas formas, es necesario un análisis de los valores extremos antes de tomar la decisión de eliminarlos, ya que en algunos casos son justamente los valores anómalos los que se quiere detectar. Por ejemplo, puede encontrarse un fraude en las compras realizadas con tarjeta de crédito analizando observaciones de compras por valores extremadamente mayores a la media de la tarjeta utilizada por un cliente.

- *Presencia de datos faltantes o perdidos (missing values)*
  La ausencia de información puede llevar a resultados poco precisos. Antes de tomar una decisión sobre cómo tratar los atributos con valores faltantes es necesario entender su significado. Los motivos para el faltante de datos puede tener orígenes

muy dispares, pudiendo deberse a errores en la aplicación de carga de datos o bien provenir de la integración de diferentes fuentes.

- *Transformación y selección de atributos*
  Es importante que los atributos seleccionados sean relevantes para la tarea de $\mathcal{DM}$. Por ejemplo, si se incluye en los datos el atributo correspondiente al número de paciente, un algoritmo de generación de reglas podría obtener un modelo correcto pero con poca generalización. Queriendo predecir la droga que se le debe suministrar a un paciente, la obtención de la regla

  ```
  SI (nro de paciente es 247) ENTONCES (suministrar la droga Y)
  ```

  en la que se hace referencia a un paciente determinado no es relevante para la tarea que se desea llevar a cabo.

  En la práctica, si bien existe la posibilidad de recurrir al conocimiento del dominio para realizar el proceso de selección en forma manual, suele tratarse de un problema complejo. Por lo tanto es necesario recurrir nuevamente a técnicas de $\mathcal{DM}$ las cuales proveen mecanismos de selección de características relevantes que operan utilizando diferentes criterios.

- *Construcción de atributos*
  Se pueden construir atributos que faciliten el proceso de minería aplicando operaciones o funciones a los atributos originales. En los casos en que se detecte que los atributos originales no poseen un alto poder predictivo por sí solos, es deseable buscar expresiones que calculen nuevos valores con mayor capacidad de predicción o descripción. Por ejemplo, para analizar la deserción universitaria el promedio de notas de un alumno parece ser un atributo más rico que las notas individuales que haya obtenido en cada materia.

- *Modificación de tipos de datos*
  Para facilitar el uso de técnicas que requieren tipos de datos específicos se pueden numerizar datos nominales o discretizar datos numéricos según sea necesario. En el primer caso se asigna un número entero distinto a cada valor nominal. Por ejemplo, el atributo referido al máximo nivel de estudios alcanzado por el padre del alumno puede convertirse en números enteros que representen el orden de los títulos posibles. El segundo caso consiste en tomar valores numéricos continuos, determinar los rangos y luego asignar un valor discreto a cada uno de ellos, como se lo hace en la figura 1.4. El ejemplo corresponde al atributo `Nota`, para el que se determinaron los rangos 0 a 3, 4 a 7 y 8 a 10 cuyos valores discretos asociados son *"desaprobado"*, *"bueno"*, *"muy bueno"* respectivamente.

- *Selección de la muestra de datos*
  La característica fundamental que debe reunir la muestra de datos a utilizar es ser representativa del proceso que se quiere modelizar. Generalmente se dispone de un número de ejemplos mucho más grande que los estrictamente necesarios. Como en el caso de los atributos, podría construirse el modelo a partir de todos los datos

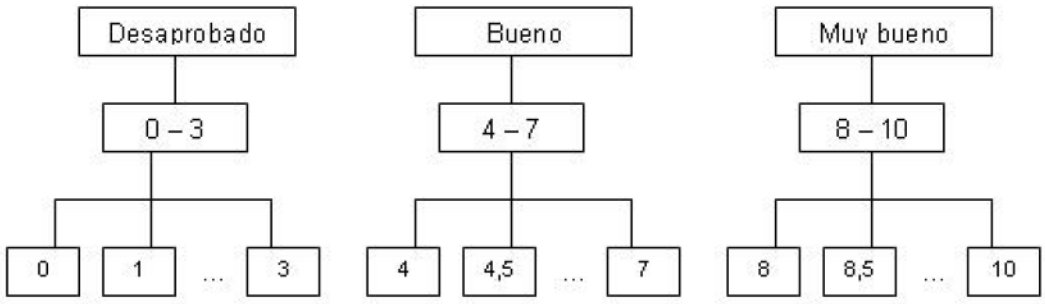


Figura 1.4: Ejemplo de discretización del atributo `Nota`

disponibles o bien puede utilizarse un subconjunto de ellos. Esto último puede realizarse a través de distintas técnicas y permitiendo, en una primera instancia, disminuir los tiempos de procesamiento.

1.4.4. Modelado

En la fase de modelado, también denominada Minería de Datos por ser la más característica del $\mathcal{KDD}$, es donde se produce conocimiento nuevo, construyendo modelos basados en los datos recopilados. El modelo describe los patrones y relaciones presentes en los datos, los cuales pueden utilizarse para predecir comportamiento futuro, entender mejor los datos o explicar situaciones observadas.

En esta fase se seleccionan y aplican diferentes técnicas de modelado configurando sus parámetros para la obtención de resultados. Usualmente existen varias técnicas aplicables a los mismos problemas de $\mathcal{DM}$. Algunas de ellas tienen requerimientos específicos en el formato de los datos, por lo que deben ser preparados los datos previamente.

En esta etapa se deben tomar las siguientes decisiones:

- Elegir la tarea de $\mathcal{DM}$ apropiada para el objetivo del proyecto y para los datos involucrados. Por ejemplo, se puede decidir usar una tarea descriptiva que ayude a conocer con más precisión las características de los alumnos que abandonan sus estudios.
- Elegir el tipo de modelo. Por ejemplo, se podría elegir el agrupamiento para obtener grupos de alumnos con características semejantes que puedan ser descriptos apropiadamente.
- Elegir el algoritmo de $\mathcal{DM}$ que resuelva la tarea y ofrezca un modelo resultante. Por ejemplo, el algoritmo K-medias [15]. Sin embargo, para poder tomar correctamente esta determinación, hay que conocer en detalle los algoritmos posibles de ser seleccionados para el objetivo planteado.

Las técnicas de $\mathcal{DM}$ utilizadas para esta fase son de carácter interdisciplinar. Existen árboles de decisión, redes neuronales, reglas de asociación, algoritmos evolutivos y

muchas más. Cada uno de estos paradigmas incluye a su vez diferentes algoritmos y variaciones de los mismos, con restricciones que hacen que la efectividad del algoritmo elegido dependa del dominio de aplicación. No existe un método de $\mathcal{DM}$ universal aplicable a cualquier tipo de problema.

En la construcción del modelo es donde se ve reflejado, con mayor claridad, el carácter iterativo del proceso de $\mathcal{KDD}$. La búsqueda de la técnica que resulte útil para la resolución del problema que se presente es la que muchas veces hace necesario retroceder a fases anteriores y hacer cambios en los datos o incluso modificar el planteo del problema.

1.4.5. Interpretación y evaluación

Los modelos obtenidos en la fase anterior deben ser interpretados y evaluados. Se revisa su construcción a fin de comprobar que se cumplen los objetivos planteados en las fases preliminares. Es indispensable en esta etapa encontrar una manera de medir la calidad de los modelos obtenidos. De la definición de Minería de Datos se puede ver que los modelos descubiertos deben ser precisos, comprensibles, útiles y novedosos. De estas características deseables, se priorizan unas u otras dependiendo de la aplicación y el uso de los mismos.

Una forma de abordar esta etapa es dividir los datos en dos subconjuntos: el conjunto de entrenamiento, que se utiliza para construir y entrenar un modelo, y el conjunto de prueba, que es usado para validar su efectividad. La separación permite garantizar que la validación de la precisión del modelo es una medida independiente. Existen diferentes técnicas para efectuar la división en subconjuntos, dependiendo de la cantidad total de datos que se disponga.

Una técnica básica de evaluación es la validación simple que reserva, mediante una selección aleatoria, un porcentaje (normalmente entre un 5 y un 50 %) de observaciones de la vista minable como conjunto de prueba. Estos datos no intervienen en la generación del modelo.

En el caso que los datos sean escasos, se puede utilizar el método de validación cruzada que divide los datos aleatoriamente en dos conjuntos equitativos, luego utiliza el primer conjunto para la construcción del modelo y el segundo conjunto para validarlo, y posteriormente realiza la misma tarea con los subconjuntos cambiados de rol. Un método muy utilizado es la validación cruzada de k pliegues (ver figura 1.5), que divide los datos en k grupos, reserva uno de los grupos para la prueba y con los otros $k-1$ grupos restantes (todos juntos) construye el modelo que luego usa para predecir los datos del grupo reservado. Este proceso se repite k veces, dejando cada vez un grupo diferente para la prueba. Ambos métodos de validación cruzada calculan la tasa de error en cada pasada y finalmente construyen un modelo con todos los datos cuya tasa de error se estima promediando las obtenidas en cada pasada.

Existe otra técnica de evaluación conocida como *bootstrapping* que construye numerosos conjuntos de datos por muestreo con reemplazo, es decir que se van seleccionando observaciones del conjunto original pudiendo repetirse. Luego construye un modelo con

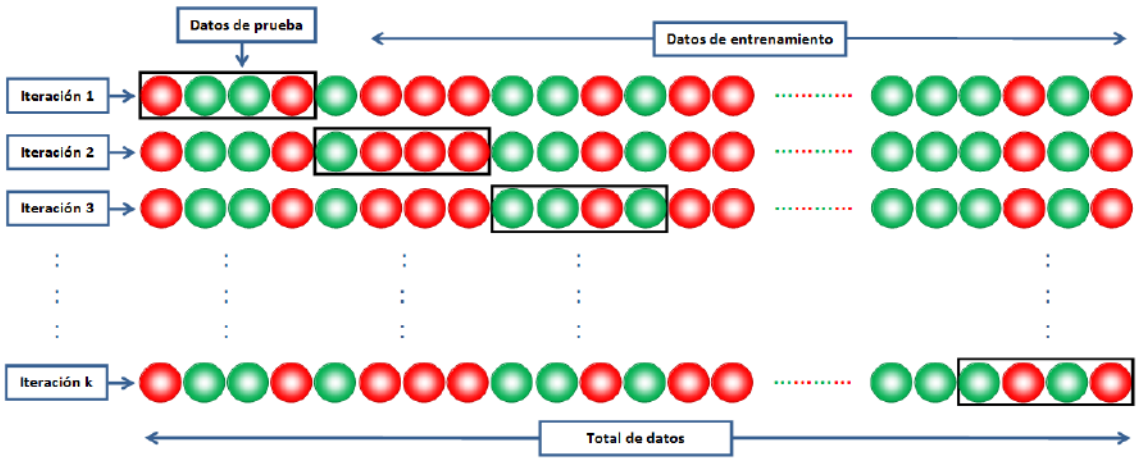


Figura 1.5: Método de validación cruzada de k pliegues

cada conjunto y lo evalúa contra el conjunto de prueba, que son los datos sobrantes de cada muestreo. El error final se calcula promediando los errores para cada muestreo.

Una vez aplicada la técnica de evaluación, existen diferentes medidas para evaluar los modelos, dependiendo de la tarea de $\mathcal{DM}$. Por ejemplo, para algunas tareas de tipo predictivo (ver sección 1.5.1) se puede medir la precisión predictiva, que se calcula como el número de instancias del conjunto de prueba que el modelo predice correctamente dividido por el número de instancias totales del conjunto de prueba. Para tareas descriptivas (ver sección 1.5.2) como el agrupamiento, las medidas de evaluación suelen depender del método utilizado. En general, se utilizan medidas de distancia para la cohesión de cada grupo (distancia media de los miembros de cada grupo a su respectivo centro) y la separación entre grupos (distancia media entre grupos).

En esta etapa es crítico determinar si partes importantes de la realidad han sido lo suficientemente consideradas. Además se debe decidir sobre la utilización de los resultados del proceso de $\mathcal{DM}$. Las tareas involucran evaluar los resultados, revisar los procesos y determinar los pasos a seguir basados en el modelo obtenido.

La naturaleza iterativa de la Minería de Datos puede llevar en esta etapa a la revisión de etapas anteriores y pueden surgir nuevas preguntas a responder que hagan que el proyecto retorne a la fase de conocimiento del dominio a fin de poder responderlas.

1.4.6. Difusión y uso de los resultados

La creación del modelo no implica la finalización del proyecto. El conocimiento obtenido debe ser organizado y presentado de manera que pueda ser comprendido y utilizado por el usuario final.

Los modelos construídos pueden utilizarse para decidir acciones basándose en sus resultados. También se pueden utilizar para aplicarlos a nuevos conjuntos de datos e incluso incorporarlos a aplicaciones que utilice la organización.

Esta fase puede ser tan simple como la generación de un informe o tan compleja como

la aplicación del modelo a diferentes juegos de datos, de manera que de lugar a fases complementarias que implementen un proceso iterativo de $\mathcal{DM}$ repetible tantas veces como sea necesario para concretar los objetivos.

También es relevante medir la evolución del modelo, aún cuando éste funcione bien. Continuamente se debe comprobar su efectividad, principalmente debido a que la realidad puede cambiar con el tiempo.

La importancia de esta fase consiste en que el usuario entienda los resultados y pueda utilizar los modelos creados. Es en este punto donde esta tesina hace incapié proporcionando un método capaz de obtener un modelo reducido fácilmente interpretable.

1.4.7. Implementación de medidas

Cuando la fase de uso de resultados genera una clase de conocimiento que habilita al usuario a ejecutar acciones en pos de resolver el problema planteado originalmente, se produce una etapa de implementación de medidas que debe llevar a cabo la organización basadas en el conocimiento obtenido. Estas medidas tendrán como objetivo mejorar o corregir la realidad descubierta a través del modelado, actuando directamente sobre la organización.

Si bien estas tareas pueden no considerarse parte de la metodología interna del $\mathcal{DM}$, siendo la implementación de decisiones generadas por el resultado de los modelos, deben ser tenidas en cuenta para retroalimentar el ciclo completo de resolución del problema.

1.4.8. Medición de resultados

Luego de la implementación de las medidas de la fase anterior, es posible la utilización del $\mathcal{DM}$ para medir los resultados alcanzados por esas acciones.

En esta fase se pueden volver a ejecutar los modelos para compararlos con los obtenidos en la primera iteración y de esa manera conseguir mediciones concretas del éxito o fracaso de las medidas tomadas.

1.5. Tareas de Minería de Datos

Como se dijo con anterioridad en la presentación de las fases del $\mathcal{KDD}$, existen dos tipos de tareas: las predictivas que tienen que ver con la necesidad de predecir un comportamiento y las descriptivas que tienden a mostrar cómo está organizada la información. A continuación se mencionan las más importantes de cada una de ellas.

1.5.1. Tareas Predictivas

Se consideran predictivas a aquellas tareas que requieren de la obtención de un modelo capaz de dar una respuesta, en una etapa posterior, ante la presencia de información nueva.

Según si la respuesta esperada es discreta o continua, se considera que la tarea predictiva es una clasificación o una regresión, respectivamente. Un ejemplo de tarea de clasificación es obtener un modelo que dado un nuevo producto pueda clasificarlo como *"Básico"*, *"Estándar"* o *"De lujo"*. Un ejemplo de tarea de regresión es obtener un modelo que dado un paciente nuevo determine la probabilidad de que tenga cierta enfermedad.

La *clasificación* es una de las tareas más utilizadas. En ella, cada ejemplo o registro de la vista minable, pertenece a una clase la cual se indica mediante el valor de un atributo nominal que se denomina etiqueta. Esta característica permite obtener el modelo a través de una estrategia supervisada que, operando sobre el resto de los atributos de cada instancia, buscará maximizar la tasa de acierto sobre el conjunto de ejemplos de entrada. Al finalizar el proceso, el clasificador obtenido será capaz de determinar la clase para cada nuevo ejemplo sin etiquetar. Entre las tareas de clasificación hay distintas variantes:

- *Clasificación suave*
  Según la técnica utilizada para la construcción del modelo puede incorporarse a la clasificación una función que determine el grado de certeza de la predicción. De esta forma, podría permitirse que un clasificador etiquetara un mismo ejemplo con más de una clase asignando a cada una de ellas un valor de certeza diferente y sería la persona encargada de tomar las decisiones quien debería decidir entre las opciones presentadas.

- *Estimación de la probabilidad de clasificación*
  Es una generalización de la clasificación suave que provee para cada valor de la clase la probabilidad de que un ejemplo sea de dicha clase. A diferencia del clasificador suave, en este caso las opciones serían excluyentes ya que se basan en la teoría de la probabilidad y la cantidad de ejemplos requeridos para su construcción debe ser grande.

- *Categorización*
  A diferencia de la clasificación, se trata de aprender una correspondencia pudiendo asignar más de una categoría a cada ejemplo.

Las tareas de *regresión* también utilizan conjuntos de ejemplos etiquetados y tienen como objetivo aprender una función que represente la correspondencia existente entre los atributos considerados en cada ejemplo y la clase indicada. Su diferencia con respecto a la clasificación es que la salida es numérica mientras que en la clasificación es nominal.

1.5.2. Tareas Descriptivas

Este tipo de tareas busca mostrar nuevas relaciones entre las variables y generalmente son utilizadas para mejorar el modelo. Su objetivo es describir los datos existentes. Entre las tareas descriptivas más frecuentes, pueden mencionarse las siguientes:

- *Agrupamiento (clustering)*
  El objetivo de esta tarea es obtener grupos o conjuntos entre los ejemplos, de manera

que los elementos asignados al mismo grupo sean similares. A priori no se sabe ni cómo son los grupos ni cuantos hay, eso se determina con el proceso de aprendizaje. Una utilidad del agrupamiento reside en que utilizando la función obtenida con nuevos ejemplos se puede determinar a qué grupo pertenece un nuevo elemento y con eso indicar su comportamiento. La tarea de agrupamiento también suele utilizarse con el objetivo de reducir un gran número de ejemplos a sólo algunos grupos que sirvan como resumen de los datos originales.

- *Correlaciones y factorizaciones*
  Se centran en atributos numéricos. Su objetivo es detectar si dos atributos numéricos están correlacionados linealmente o relacionados de algún otro modo. Su utilidad es la detección de atributos redundantes o dependientes, permitiendo analizar la relevancia de atributos y hacer una selección entre ellos.

- *Reglas de asociación*
  Es un estudio similar al de correlaciones pero para atributos nominales, muy frecuentes en las bases de datos. Una regla de asociación se define generalmente de la forma

$$\textbf{SI } (atrib_1 = valor_1) \textbf{ Y } (atrib_2 = valor_2) \textbf{ Y...Y } (atrib_k = valor_k) \textbf{ ENTONCES } (atrib_r = valor_r) \textbf{ Y } (atrib_s = valor_s) \textbf{ Y...Y } (atrib_z = valor_z)$$

  donde todos los atributos son nominales y las igualdades se definen utilizando algún valor de los posibles para el atributo.

1.6. Conclusiones

La Extracción de Conocimiento a partir de la información disponible no es una tarea mágica.

En este capítulo se han resumido las distintas etapas que forman el proceso de $\mathcal{KDD}$ con el objetivo de acercar al lector al esfuerzo que representa la obtención de patrones que la mayoría de las definiciones describe como "... novedosos, útiles y comprensibles".

La primera tarea consiste en decidir cuáles son los datos a procesar. En ocasiones, esto está limitado a los datos históricos previamente recolectados pero en otros casos, es posible elegir y es allí donde surgen las primeras dudas tratando de identificar cuáles son los atributos o características importantes para el problema. En esta etapa hay distintas técnicas que pueden resultar de utilidad. Nótese que la selección realizada en esta etapa tendrá una fuerte incidencia en los resultados obtenidos ya que en Informática, los datos pueden procesarse pero no inventarse. Si un atributo relevante para el problema no hubiera sido registrado, será imposible que el modelo final lo tenga en cuenta en su construcción.

Luego debe realizarse el preprocesamiento de los datos. Este es el proceso que requiere la mayor cantidad de tiempo y tiene por resultado la obtención de la "vista minable".

Estas etapas sólo han sido mencionadas en este capítulo por una cuestión de completitud ya que no han sido tenidas en cuenta en el método propuesto en esta tesina. El lector

notará que todos los resultados han sido medidos sobre bases de datos existentes en repositorios ampliamente conocidos en la literatura.

Siguiendo esta misma línea, el tipo de problema que se busca resolver en esta tesina es la clasificación de la información disponible. Es decir que no todas las técnicas de $\mathcal{DM}$ pueden ser aplicadas. En particular aquí se propone combinar una arquitectura de Red Neuronal con una Técnica de Optimización. Como el lector imaginará, existen técnicas ya definidas para resolver este tipo de problemas y por ese motivo, en el siguiente capítulo se realiza una descripción de las más usadas. En particular, al finalizar el capítulo 2, el lector encontrará una descripción del método contra el cual se realizan las comparaciones de la técnica propuesta.

Las etapas siguientes a la extracción del modelo se relacionan con la interpretación y presentación del conocimiento adquirido. En este caso, eso tiene que ver con la simplicidad del modelo y este es también un aspecto importante en el cual se ha puesto un gran cuidado. Las mediciones efectuadas en el último capítulo permiten afirmar que el modelo obtenido por el método propuesto en esta tesina es más fácil de interpretar debido a su reducido tamaño.

Capítulo 2

Técnicas de $\mathcal{DM}$ para Clasificación

2.1. Introducción

Esta tesina propone una nueva técnica para resolver una de las tareas más importantes que puede realizarse utilizando $\mathcal{DM}$: la clasificación de la información disponible.

Este capítulo tiene como objetivo presentar y ejemplificar brevemente las técnicas existentes que permiten obtener los modelos más populares basados en árboles y reglas de clasificación.

Dichos modelos se construyen a partir del análisis automático de la información disponible la cual está formada por instancias o registros de valores de distintos atributos o características del problema. Estos atributos o características pueden ser: numéricos (ej: cantidad de empleados, edad, sueldo, metros cuadrados, promedio de notas, etc.) o bien categóricos o nominales (ej: estado civil, raza, idioma, etc.).

Por otro lado, frente a un problema de clasificación, generalmente uno de los atributos categóricos presentes en los datos es considerado la etiqueta, es decir, la clase que le corresponde a cada una de las instancias. Sin embargo esta información no siempre está disponible o es conocida. Por tal motivo, existen técnicas de $\mathcal{DM}$ que sólo se aplican cuando se dispone de ejemplos etiquetados y otras que pueden operar aún sin contar con esta información.

Dada una instancia en particular, clasificarla significa determinar entre todas las clases posibles, a cuál pertenece. Por ejemplo, a partir de los síntomas de un paciente decidir la droga a suministrar o en base la información personal y académica de un alumno determinar su probabilidad de continuar con sus estudios universitarios.

Para responder lo pedido, es preciso identificar los atributos relevantes para medir la deserción universitaria o para representar la patología del paciente. Esto lleva a la necesidad de identificar y seleccionar los atributos que participarán del modelo descartando los que sean irrelevantes para la decisión. El problema es, por supuesto, decidir cuáles pueden dejarse de lado sin afectar la decisión final.

La característica más importante de la clasificación es que asume que las clases son disjuntas y por ello, un registro es de una única clase.

Ante un problema que requiere clasificación existen distintos modelos que cumplen dicho

objetivo, entre los cuales se encuentran los árboles de decisión y las reglas de clasificación. Estos modelos, clásicos de la Minería de Datos, son los más usados por quienes toman decisiones por resultar fáciles de utilizar y entender.

Ambos modelos aprenden de los datos de manera supervisada y, además de ser utilizados para clasificar, pueden determinar las características relevantes del problema según los datos aprendidos.

A continuación serán desarrollados ambos tipos de modelos y se proporcionarán ejemplos realizados con RapidMiner (ver apéndice A). Finalizando el capítulo, se profundizará el funcionamiento del método PART [16].

2.2. Arboles de clasificación

Los árboles de decisión son una de las técnicas más utilizada para clasificación. El enfoque "divide y vencerás" a través del cual se construyen este tipo de modelos y el hecho de que los problemas de clasificación asumen clases disjuntas, llevan naturalmente al estilo de representación de un árbol para dichos problemas. Por ejemplo, recorriendo según los síntomas de una persona, la rama del árbol de la figura 2.1 que corresponda, la estructura recomendará suministrar una determinada droga.

Un árbol de decisión es un conjunto de condiciones organizadas en una estructura jerárquica. Cada nodo del árbol involucra un atributo y cada una de sus ramas una condición sobre dicho atributo, las cuales consisten normalmente en la comparación con determinadas constantes si el atributo es nominal. La decisión a tomar a partir de un registro, se determina siguiendo las condiciones que aquel cumple desde la raíz del árbol hasta una de sus hojas asociada a alguna de las clases posibles. Los nodos hoja proporcionan la clasificación que se aplica a todas las instancias que alcanzan cada una de ellas. Esta es precisamente una de las ventajas que poseen los árboles: conducen el ejemplo hasta una única hoja, asignando por tanto, una única clase al ejemplo.

La figura 2.1 muestra un árbol de clasificación que permite diagnosticar la droga que debe ser suministrada a un paciente con síntomas de rinitis alérgica. Como se observa en la figura, existen cinco drogas posibles: A, B, C, X e Y. Si el atributo que se compara en un nodo es nominal, el número de hijos es igual a la cardinalidad del atributo (a menos que en determinado nivel del árbol no haya ejemplos para alguno de sus valores). En este caso el atributo no será comparado nuevamente más abajo en el árbol, dado que se agotaron todos sus posibles valores en esta comparación. Tal es el caso del atributo `Presion` en la figura 2.1 que puede tomar uno de los siguientes valores: *"Alta"*, *"Normal"* y *"Baja"*. En cambio, los valores de un atributo numérico pueden dividirse en subconjuntos, en ese caso el atributo podría ser utilizado en otro nodo con una nueva comparación más específica. Esto ocurre por ejemplo en el nodo raíz del árbol, formado por el atributo `Potasio`, el cual vuelve a utilizarse en el primer nivel de la rama izquierda. Como ya se dijo anteriormente, la aparición de un mismo atributo más de una vez sobre la misma rama del árbol sólo es válido para los atributos numéricos.

Dado que los algoritmos básicos de aprendizaje de árboles de decisión se basan en la disyunción de las clases, las particiones de datos presentes en los árboles también deben

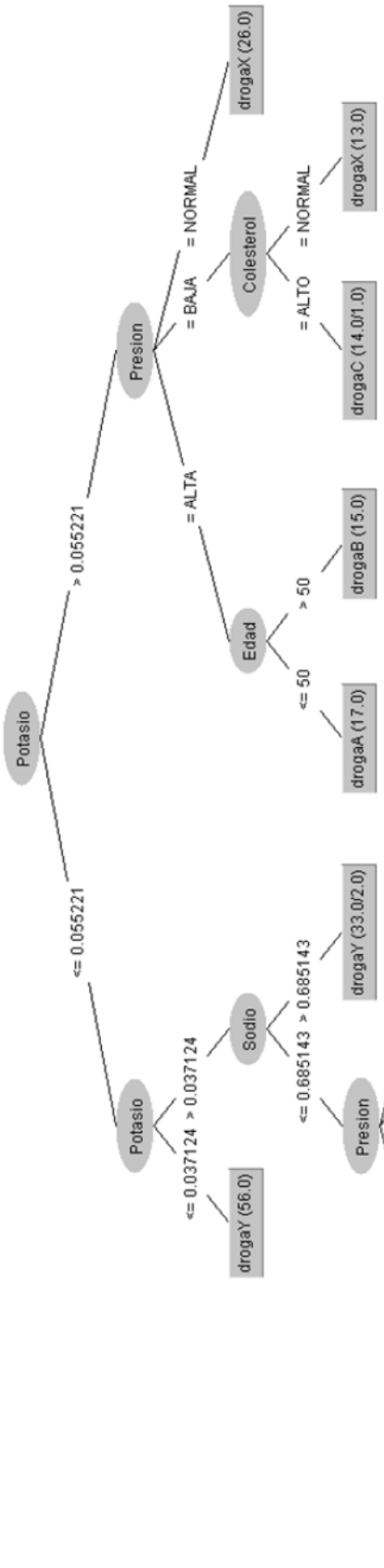


Figura 2.1: Arbol de clasificación para el suministro de drogas a pacientes con rinitis alérgica

ser disjuntas. De esta manera el espacio de instancias se va partiendo de arriba hacia abajo mediante condiciones excluyentes y exhaustivas. Por tal motivo, es importante la selección de "buenas" particiones, seleccionando para cada nodo el atributo que mejor separe los ejemplos entre todos sus hijos. Una mala elección de la partición (especialmente en las partes superiores del árbol) generará un árbol malo.

Los atributos que proporcionan buenas particiones son aquellos relevantes para el problema, de manera que los árboles proveen también una solución a las tareas descriptivas, mostrando jerárquicamente la organización de la información. Es por eso que conociendo la manera en que se organizan los atributos en el árbol de la figura 2.1, por ejemplo, se pueden conocer los síntomas generales que permiten diagnosticar determinada droga.

El algoritmo 1 indica la manera de construir un árbol de clasificación utilizando el método ID3 difinido por Quinlan en [17]. En dicho algoritmo, el árbol se construye en forma recursiva, de arriba hacia abajo. Al comienzo, todos los ejemplos del conjunto de datos están en el nodo raíz y se particionan recursivamente según el atributo seleccionado en cada momento. El particionamiento se detiene cuando todas las muestras para un nodo dado corresponden a la misma clase, cuando no hay más atributos para particionar, o cuando no quedan más ejemplos. Cuando todas las muestras para la hoja correspondan a la misma clase, dicha clase será la utilizada por la hoja para clasificar. Caso contrario, se utilizará la clase con mayor representantes para clasificar la hoja. El algoritmo 1 genera un árbol de decisión para un conjunto de ejemplos E siguiendo este proceso partitivo.

Algoritmo 1: Algoritmo genérico de construcción de un árbol de decisión

```
Function Particion(E: conjunto de ejemplos)
begin
    if (todos los ejemplos de E son de la misma clase c) or (son muy pocos como
    para dividirlos) then
        N ← Generar un nodo hoja
        N.clase ← la clase mayoritaria de E
    else
        Seleccionar el atributo con menor desorden
        N ← Generar un nodo con tantas ramas como valores tenga el atributo
        seleccionado en E
        for i=1 to Cantidad de valores posibles para el atributo seleccionado do
            // E_i son los ejemplos que corresponden a la rama i del nodo N
            N.rama(i) ← Particion(E_i)
    return N
```

2.2.1. Criterio de selección de atributos

Como ya se dijo anteriormente, la forma de seleccionar los atributos es importante para la creación y posterior funcionamiento del árbol. Esta tarea puede ser heurística o mediante

una medida estadística. La idea es disponer de una métrica para decidir cuáles atributos ubican mejor los datos dentro del árbol. Hay distintas variantes de estas medidas y por ello constituyen un parámetro del método a utilizar para la construcción del árbol.

El criterio que habitualmente se utiliza se denomina *Tasa de Ganancia* (en inglés *Gain Ratio*). Es una métrica que tiene en cuenta no sólo la capacidad del atributo para separar los ejemplos sino también la cantidad de valores posibles que puede tomar. El cálculo de la *Tasa de Ganancia* de un atributo requiere de la definición de algunos conceptos previos.

Sean D el conjunto de datos que se desea evaluar y C la cantidad de clases distintas a las que pueden pertenecer los ejemplos de D. Se denominará como $p(D, j)$ a la proporción de ejemplos de D que pertenecen a la j-ésima clase. La incertidumbre referida a la clase a la cual pertenece un ejemplo de D puede expresarse como

$$Info(D) = -\sum_{j=1}^{C} p(D, j) * log_2(p(D, j)) \tag{2.1}$$

Un atributo T con opciones mutuamente excluyentes $T_1, T_2, \ldots, T_k$, al ser aplicado sobre los ejemplos de D dará lugar a los subconjuntos $D_1, D_2, \ldots, D_k$ donde D_i contiene aquellos ejemplos que verifican la condición T_i del atributo.

Por ejemplo para un atributo `Edad` que posea sólo dos opciones *(T_1 : `Edad` ≤ 45)* y *(T_2 : `Edad` > 45)* dividirá al conjunto de ejemplos de entrada D en dos subconjuntos de acuerdo al valor de edad que cada uno de ellos posea. El subconjunto D_1 estará formado por los que tengan una valor del atributo `Edad` por debajo de 45 y el conjunto D_2 contendrá el resto de los ejemplos. Puede verse que la aplicación de las opciones de un atributo T permite obtener una partición de D.

La información ganada por un atributo T que posee k opciones se define de la siguiente forma:

$$Gain(D, T) = Info(D) - \sum_{i=1}^{k} \frac{|D_i|}{D} * Info(D_i) \tag{2.2}$$

La ganancia definida en 2.2 se encuentra fuertemente influenciada por la cantidad de opciones del atributo T y toma su valor máximo cuando cada subconjunto D_i está formado por un único elemento. Esto quiere decir que por ejemplo, si D tuviera información de los alumnos de la Facultad de Informática y para cada uno de ellos se encontrara codificado el número de legajo, ese sería precisamente el atributo que poseería la mayor ganancia ya que no es posible tener dos alumnos con el mismo valor.

Es preciso, por lo tanto, contar con alguna métrica que permita penalizar a aquellos atributos que posean muchas opciones. Para ello se utilizará la entropía relacionada con la cantidad de particiones que un atributo efectúa sobre los datos de entrada. Esta métrica se denomina *Split Information* y se calcula de la siguiente forma:

$$Split(D, T) = -\sum_{i=1}^{k} \frac{|D_i|}{D} * log_2\left(\frac{|D_i|}{|D|}\right) \tag{2.3}$$

Nótese que el incremento del valor de *Split(D,T)* es proporcional a la cantidad de opciones que tiene el atributo para separar los datos.

Finalmente, la *Tasa de Ganancia* de un atributo T para un conjunto de datos D se calcula como

$$GainRatio(D,T) = \frac{Gain(D,T)}{Split(D,T)}) \tag{2.4}$$

Por lo tanto, cuando se trata de seleccionar un atributo de un conjunto de atributos posibles, se evalúa la ecuación 2.4 para cada uno de ellos y se selecciona aquel que posea el mayor valor.

2.2.2. Poda y reestructuración

Los algoritmos de aprendizaje de árboles de decisión obtienen un modelo que cubre todos los ejemplos del conjunto utilizado. Esta situación, que puede parecer ideal, es un ajuste demasiado estricto y suele provocar que el modelo trabaje mal para nuevos ejemplos. La solución a este problema se resuelve con la denominada "poda" del árbol obtenido. La poda elimina nodos inferiores de un árbol que se consideran demasiado específicos.

La poda puede realizarse durante el proceso de construcción (prepoda) o luego de éste (pospoda). En el primer caso se trata de determinar el criterio de parada al momento de seguir especializando una rama, y se basa en el número de ejemplos en un nodo, en el número de excepciones respecto a la clase mayoritaria (error esperado) u otras técnicas más sofisticadas. La pospoda trata de eliminar nodos de abajo hacia arriba hasta un límite, basado en las mismas medidas que la prepoda. La diferencia es que la pospoda se realiza con una visión completa del modelo, pudiendo por eso obtener mejores resultados. Cuando se poda se consiguen nodos impuros (con elementos de diferentes clases), eligiendo normalmente la clase mayoritaria para etiquetar el nodo hoja.

Es una tarea difícil determinar el nivel de poda con exactitud, dado que depende en gran medida de los datos específicos de cada problema. Es por esto que suele utilizarse el conjunto de datos de validación para esclarecer este punto, si se dispone de ellos.

2.2.3. Calidad del modelo

La calidad predictiva de un árbol de clasificación puede expresarse en una tabla de *nxn* siendo *n* la cantidad de clases, como se muestra en la figura 2.2. Esta tabla, llamada matriz de confusión, tiene distribuidos los ejemplos permitiendo ver sobre su diagonal la cantidad de ejemplos que fueron correctamente clasificados y fuera de ella los que no.

Las columnas representan la clase de los ejemplos y las filas las predicciones realizadas por el modelo. Los ejemplos se distribuyen en las celdas según su clase y la clasificación que les proporciona el árbol. En la figura puede observarse que están calculadas para cada clase, la precisión de la predicción (proporción de los ejemplos que fueron clasificados como de una determinada clase, predecidos correctamente) y su alcance (proporción de ejemplos de una determinada clase correctamente clasificados).

accuracy: 97.00%

	true drugY	true drugC	true drugX	true drugA	true drugB	class precision
pred. drugY	88	0	2	0	1	96.70%
pred. drugC	1	16	0	0	0	94.12%
pred. drugX	2	0	52	0	0	96.30%
pred. drugA	0	0	0	23	0	100.00%
pred. drugB	0	0	0	0	15	100.00%
class recall	96.70%	100.00%	96.30%	100.00%	93.75%	

Figura 2.2: Matriz de confusión obtenida a partir del resultado de aplicar el árbol de la figura 2.1 a la base DrugY

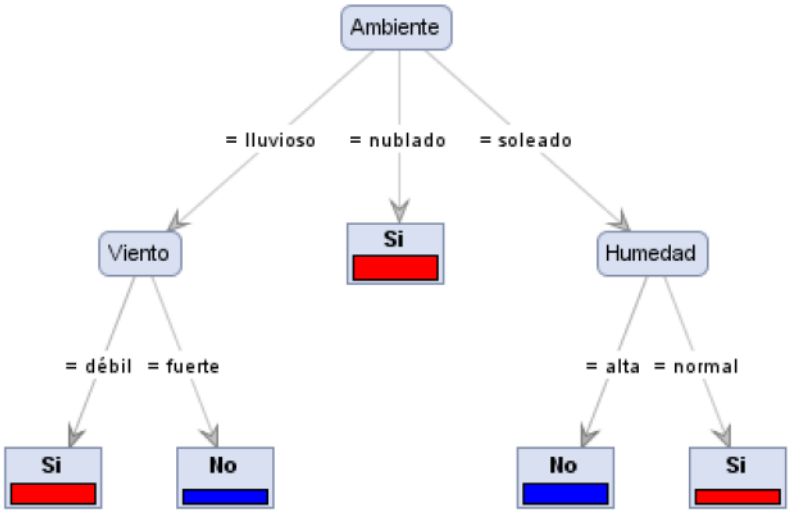


Figura 2.3: Árbol de decisión para determinar si se juega o no al golf

2.2.4. Algoritmos principales

Existen varios algoritmos específicos para la construcción de árboles de decisión. Estos métodos respetan el algoritmo 1 y definen los aspectos desarrollados anteriormente. Los criterios que utilizan cada uno de ellos para el aprendizaje del árbol dan lugar a determinadas restricciones.

Cada método tiene capacidades específicas para el manejo del conjunto de datos. Algunos algoritmos no soportan que existan datos faltantes. Otros necesitan que todos los atributos sean discretos y si existen atributos con valores continuos deben ser discretizados previamente. Tal es el caso del método ID3 definido en 1986 por J. Ross Quinlan [17]. La figura 2.3 muestra un árbol de decisión generado con ID3 en base a los datos de la tabla 2.1.

Años más tarde, el algoritmo ID3 fue mejorado para convertirse en el algoritmo C4.5 incorporando, entre otras cosas, la capacidad de operar con atributos numéricos. Para ello, por cada uno de los atributos numéricos se ordenan de menor a mayor sus valores, se mide

Ambiente	Temperatura	Humedad	Viento	Juega?
soleado	alta	alta	débil	No
soleado	alta	alta	fuerte	No
nublado	alta	alta	débil	Si
lluvioso	media	alta	débil	Si
lluvioso	baja	normal	débil	Si
lluvioso	baja	normal	fuerte	No
nublado	baja	normal	fuerte	Si
Soleado	media	alta	debil	No
Soleado	baja	normal	débil	Si
lluvioso	media	normal	débil	Si
Soleado	media	normal	fuerte	Si
Nublado	media	alta	fuerte	Si
Nublado	alta	normal	débil	Si
lluvioso	media	alta	fuerte	No

Tabla 2.1: Conjunto de datos de condiciones climáticas para jugar al golf formado sólo por atributos categóricos

el valor de desorden de los puntos intermedios entre par de valores y se elige el punto de corte con menor desorden para que compita con el resto de los atributos, nominales y/o numéricos.

Para una descripción detallada del método C4.5 se recomienda leer [18]. También se encuentra disponible para descargar en forma gratuita la implementación del método C4.5 suministrada por el autor en [19].

2.3. Modelos basados en reglas

Los modelos basados en reglas al igual que los árboles de decisión describen los datos de los que aprenden.

Los sistemas de reglas son una generalización de los árboles de decisión en la que no se exige exclusión ni exhaustividad en las condiciones de las reglas (es decir, podría aplicarse más de una regla o ninguna).

Los métodos de construcción de reglas tienen un concepto muy diferente de los que generan árboles: cada regla se generan con el objetivo de cubrir un subconjunto de ejemplos mientras que en el árbol, cada nodo es seleccionado por su capacidad para particionar los ejemplos. Podrían decirse entonces que los árboles son partitivos mientras que las reglas buscan cobertura (figura 2.4). Por este motivo, los árboles de clasificación sólo pueden asignar una clase a un mismo ejemplo mientras que un conjunto de reglas podría asignar más de una (en caso de existir reglas solapadas). Una característica que se observa en la figura 2.4 es que el árbol cubre todo el espacio de entrada mientras que el

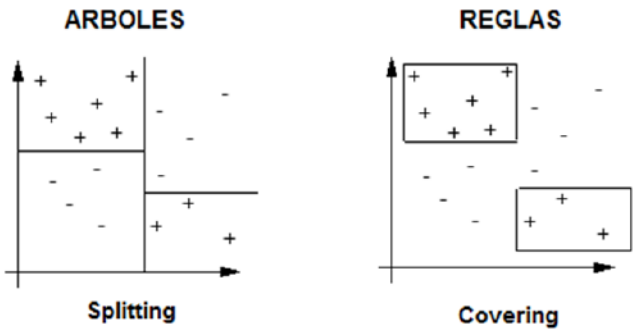


Figura 2.4: Clasificación de ejemplos con dos atributos. A la izquierda se encuentra la partición que cada nodo del árbol realiza sobre los atributos y a la derecha se muestran dos reglas que permiten cubrirlos sin superposición

conjunto de reglas puede presentar "huecos".

2.3.1. Reglas de asociación

Las reglas de asociación, tal como se mencionó en la sección 1.5.2, son expresiones de la forma

SI (*condicion*$_1$) **ENTONCES** (*condicion*$_2$)

donde ambas condiciones, antecedente y consecuente respectivamente, son conjunciones de proposiciones de la forma "`atributo = valor`" y cuya única restricción es que los atributos que intervienen en el antecedente de la regla no formen parte del consecuente.

Una regla de asociación es una proposición sobre la ocurrencia de ciertos estados en una base de datos o asociaciones de valores de atributos. Por ejemplo, en un servidor web permitirían conocer cuáles son los itinerarios más seguidos por los visitantes de un sitio web o en un supermercado la relación en la compra de productos determinando qué productos se compran conjuntamente. Suponiendo una base de datos compuesta por una única tabla como la 2.2, una regla de asociación típica sería

SI (*Bizcochos*) **Y** (*Miel*) **ENTONCES** (*Galletas*)

donde tanto el antecedente como el consecuente de la regla se refieren a productos de un supermercado. La regla dice que cada vez que se compran *Bizcochos* y *Miel* también se compran *Galletas*. Puede apreciarse que la tabla sólo contiene valores binarios indicando "*1*" si el cliente compra determinado producto y "*0*" si no lo hace. Si bien son valores numéricos, representan los valores nominales *"Compra"* o *"No compra"*. Además, no necesariamente deben tener dos valores posibles y ser los mismos valores para todos los atributos. Tal es el caso de la tabla 2.1 que posee (dejando de lado la clase de los datos correspondiente al atributo `Juega?`) cuatro atributos cuyos valores son categorías

Vino	Gaseosa	Leche	Miel	Bizcochos	Galletas	Jugo
1	1	0	0	0	1	0
0	1	1	0	0	0	0
0	0	0	1	1	1	0
1	1	0	1	1	1	1
0	0	0	0	0	1	0
1	0	0	0	0	1	1
0	1	1	1	1	0	0
0	0	0	1	1	1	1
1	1	0	0	1	0	1
0	1	0	0	1	0	0

Tabla 2.2: Tabla cuyas filas se refieren a compras realizadas en un comercio y las columnas a cada uno de los productos en venta en dicho comercio

simbólicas en lugar de números y de los cuales algunos de ellos poseen más de dos valores posibles.

El aprendizaje de este tipo de reglas requiere que se establezcan requisitos mínimos, como por ejemplo un soporte mayor a 0.02 (ver sección 2.3.5). La metodología consiste en extraer el conjunto de ítems que cumplan con las condiciones establecidas (para el ejemplo, el soporte requerido) y generar las reglas a partir de los items conseguidos. Por ejemplo, la tabla 2.1 con el atributo `Ambiente` que puede ser *"soleado"*, *"nublado"* o *"lluvioso"*, el atributo `Temperatura` que puede tomar los valores *"alta"*, *"media"* y *"baja"*, la `Humedad` *"alta"* o *"baja"* y el `Viento` que puede valer *"fuerte"* o *"débil"*, crea 36 combinaciones posibles ($3*3*2*2$) de las cuales 14 están presentes en el conjunto de datos. De estos datos surge, entre otras, la siguiente regla:

$$\textbf{SI}\ \ (Temperatura = baja)\ \ \textbf{ENTONCES}\ \ (Humedad = normal)$$

Si bien el atributo `Juega?`, por ser la clase de los datos, fue dejado de lado para el desarrollo del concepto de regla de asociación, puede formar parte de una regla de asociación. Por ejemplo, la regla

$$\textbf{SI}\ \ (Viento = debil)\ \ \textbf{Y}\ \ (Juega? = No)\ \ \textbf{ENTONCES}\ \ (Ambiente = soleado)\ \ \textbf{Y}\ \ (Humedad = Alta)$$

en la que forma parte del antecedente o la regla

$$\textbf{SI}\ \ (Humedad = normal)\ \ \textbf{Y}\ \ (Viento = debil)\ \ \textbf{ENTONCES}\ \ (Juega? = Si)$$

en la que forma parte del consecuente. Cuando el conjunto de reglas de asociación obtenido presenta en el consecuente el mismo atributo se dice que se trata de un conjunto de reglas de clasificación [20] [2]. Una regla de asociación, a diferencia de una de clasificación, puede "predecir" cualquiera de los atributos e incluso más de uno como se ha visto en los ejemplos suministrados en esta sección.

Ambiente	Temperatura	Humedad	Viento	Juega?
soleado	85	85	no	No
soleado	80	90	si	No
nublado	83	86	no	Si
lluvioso	70	96	no	Si
lluvioso	68	80	no	Si
lluvioso	65	70	si	No
nublado	64	65	si	Si
Soleado	72	95	no	No
Soleado	69	70	no	Si
lluvioso	75	80	no	Si
Soleado	75	70	si	Si
Nublado	72	90	si	Si
Nublado	81	75	no	Si
lluvioso	71	91	si	No

Tabla 2.3: Versión de la tabla 2.1 con algunos atributos numéricos

2.3.2. Reglas de clasificación

Las reglas de clasificación son una alternativa popular a los árboles de decisión. El antecedente o precondición de una regla es una serie de verificaciones como las de los nodos del árbol y el consecuente o conclusión, proporciona la clase que se aplica a los ejemplos cubiertos por la regla. Un ejemplo de regla de clasificación es

$$\texttt{SI}\ (Ambiente = soleado)\ \texttt{Y}\ (Humedad = alta)\ \texttt{ENTONCES}\ (Juega? = No)$$

la cual corresponde a los datos de la tabla 2.1 y predice en términos de jugar o no al golf. La 2.3 es una versión levemente más compleja que la 2.1 ya que dos de los atributos, `Temperatura` y la `Humedad`, tienen valores numéricos. Esto significa que cualquier método de aprendizaje de reglas debe crear desigualdades que involucren estos atributos en lugar de simples verificaciones de igualdad, como en el caso anterior. Para llegar a reglas que implican condiciones numéricas, al igual que sucede con los árboles, se requiere un proceso un poco más complejo. Ahora, la primera regla dada anteriormente puede tomar la siguiente forma:

$$\texttt{SI}\ (Ambiente = soleado)\ \texttt{Y}\ (Humedad > 83)\ \texttt{ENTONCES}\ (Juega? = No)$$

En general, las condiciones del antecedente están todas conectadas a través de un *AND* lógico, y todas las verificaciones deben tener éxito si la regla se dispara. Sin embargo, algunas reglas son formuladas con expresiones lógicas más generales en el antecedente que simplemente conjunciones. Utilizando *OR* lógico para conectar las condiciones de una regla, si una de ellas se verifica, la clase que figura en el consecuente se aplicará a

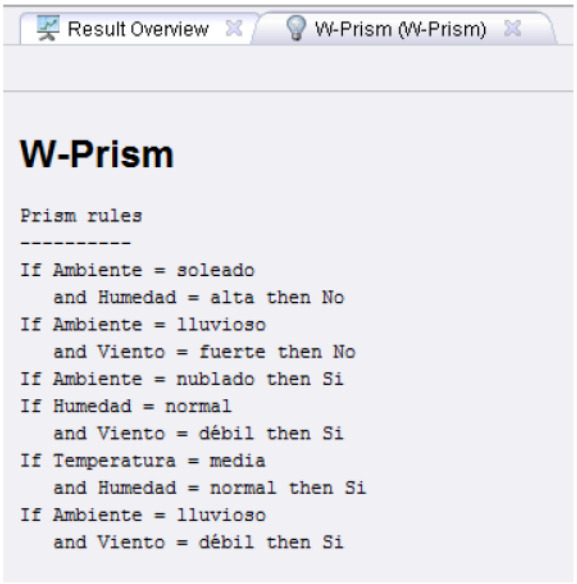


Figura 2.5: Lista de clasificación obtenida a partir de la tabla 2.1

la instancia. Si bien se gana generalización, esto provoca conflicto cuando dada una instancia, varias reglas se pueden aplicar y ellas varían en su consecuente. Esto no ocurre cuando se obtiene una lista de clasificación (ver sección 2.3.3), cuyas reglas forman parte de un grupo y no resultan ser independientes.

2.3.3. Lista de decisión

Una lista de decisión es una serie de reglas de clasificación que deben ser interpretadas en orden y su significado depende de éste por tener un lugar establecido. Para clasificar un ejemplo, las reglas son aplicadas en el orden en que fueron obtenidas y el ejemplo será clasificado con la clase correspondiente al consecuente de la primera regla cuyo antecedente se verifique para el ejemplo en cuestión. Es decir que, en primer lugar se analiza la primera regla, si el ejemplo no cumple con su antecedente a continuación se analiza la segunda, y así sucesivamente hasta que cumpla con el antecedente de alguna de ellas, clasificando el ejemplo con su consecuente. Las reglas de la figura 2.5 constituyen una lista de clasificación para la tabla 2.1.

Que un ejemplo cumpla con el antecedente de una regla significa cumplir con lo que ésta pide y no cumplir con las condiciones de todas las reglas que se encuentran antes que ella. La construcción de una lista de decisión busca caracterizar exactamente a los datos. Para obtenerla se generan las reglas una a una y los ejemplos son retirados del conjunto de datos de entrada a medida que son cubiertos por cada una (figura 2.6). Esto significa que se genera una regla, se eliminan los ejemplos que dicha regla cubre y se continúa generando reglas con los ejemplos restantes hasta que no queden ejemplos por cubrir. Este mecanismo de construcción que sigue el enfoque ´´separa y vencerás" (diferente al utilizado por un árbol), es el que lleva a obtener una lista de decisión, en lugar de un conjunto de reglas de clasificación independientes, dado que el orden de ejecución de las

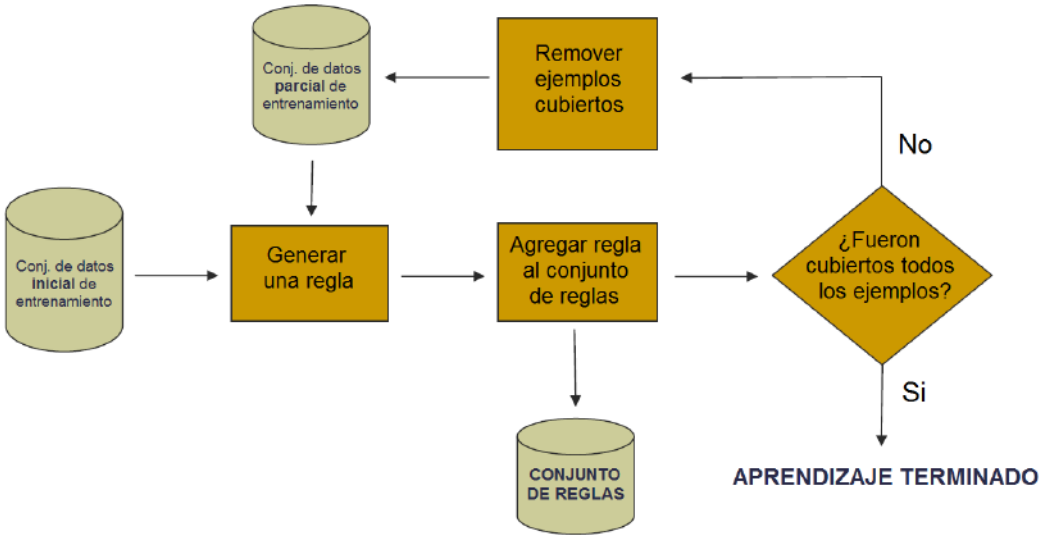


Figura 2.6: Proceso de construcción de una lista de clasificación

reglas queda predeterminado. Indefectiblemente, considerándolas individualmente (fuera de un cierto contexto) tendrán menor desempeño que interpretándolas juntas y en orden.

Una lista de decisión tiene la importante ventaja de que no genera ambigüedades cuando se interpretan sus reglas como sí puede suceder desafortunadamente con reglas de clasificación independientes.

2.3.4. Relación entre reglas y árboles

Los árboles de decisión pueden considerarse una forma de aprendizaje de reglas ya que su estructura puede leerse como tales sólo recorriendo sus ramas. Cada rama del árbol puede interpretarse como una regla, donde los nodos internos en el camino desde la raíz a la hoja definen los terminos de la conjunción que constituye el antecedente y el consecuente la clase asignada a la hoja. Este proceso produce reglas no ambiguas en las cuales el orden en el que se ejecutan no es relevante. Sin embargo, en general, las reglas que derivan de un árbol de decisión son mucho más complejas de lo necesario, y por ello, suelen ser podadas. Por ejemplo, el siguiente conjunto de reglas corresponde al árbol de la figura 2.3:

SI $(Ambiente = lluvioso)$ **Y** $(Viento = debil)$ **ENTONCES** $(Juega? = Si)$
SI $(Ambiente = nublado)$ **ENTONCES** $(Juega? = Si)$
SI $(Ambiente = soleado)$ **Y** $(Humedad = normal)$ **ENTONCES** $(Juega? = Si)$
EN OTRO CASO $(Juega? = No)$

Si bien las reglas operan sobre atributos categóricos por derivar de un árbol ID3, se puede utilizar C4.5 para construir un árbol y obtener reglas con atributos numéricos. Notar también que se han agrupado en la regla por defecto "`EN OTRO CASO`" todas las ramas del árbol cuya hoja asigna la clase *"No"* y el conjunto de reglas constituye una lista de

decisión en la que el orden importa. Las listas de decisión son similares a los árboles ya que cubren todos los ejemplos, pretenden ser perfectos y dado un ejemplo hay una única regla o rama, según corresponda, que éste cumple.

En general las reglas son más compactas que los árboles, especialmente si puede usarse una regla por defecto como se acaba de ver. Cuando se trata de reglas independientes cada una de ellas puede representar un concepto distinto, lo que permite agregar o quitar reglas fácilmente, algo que no es fácil de hacer en un árbol o una lista de decisión. Pero por otro lado, distintas reglas pueden conducir a conclusiones diferentes para la misma instancia. Esta situación no sucede para reglas que se leen directamente de un árbol de decisión o que pertenecen a una lista de clasificación ya que que su estructura evita cualquier ambigüedad en la interpretación. Por otra parte, debido a que los árboles de decisión no pueden expresar la separación implícita entre las diferentes reglas en un conjunto de reglas independientes, la transformación de un conjunto de reglas en un árbol no es tan sencilla y genera redundancia. Esto se ve en el árbol de la figura reffig:ArbolRedundante en el que se debe replicar un determinado subárbol para poder clasificar de manera equivalente a como lo hacen las siguientes dos reglas:

SI (a) **Y** (b) **ENTONCES** $(clase = x)$
SI (c) **Y** (d) **ENTONCES** $(clase = x)$

2.3.5. Métricas para reglas

Existen distintas métricas para determinar la importancia o calidad de una regla ya sea de asociación o clasificación. Las más habituales son:

- Soporte o cobertura: proporción de ejemplos que cumplen con la regla. Por ejemplo, del total de 14 registros de la tabla 2.1 en 4 de ellos se verifica que la humedad es normal, el viento es débil y sí se juega, por lo que el soporte de la regla "**SI** $(Humedad = normal)$ **Y** $(Viento = debil)$ **ENTONCES** $(Juega? = Si)$" será 4/14.
- Confianza o precisión: es el cociente entre la cantidad de ejemplos que cumplen con la regla y la cantidad de los que cumplen únicamente con el antecedente. Por ejemplo, la regla "**SI** $(Temperatura = baja)$ **Y** $(Humedad = normal)$ **ENTONCES** $(Juega? = Si)$" verificada por 3 de los 14 registros disponibles y habiendo 4 que tienen la temperatura baja y la humedad normal, tiene confianza 3/4.

Si la regla es de clasificación, su rendimiento predictivo puede resumirse mediante la siguiente matriz:

		Clase real		
		C	no C	
Clase predecida	C	***TP***	***FP***	cumplen A
	no C	***FN***	***TN***	no cumplen A

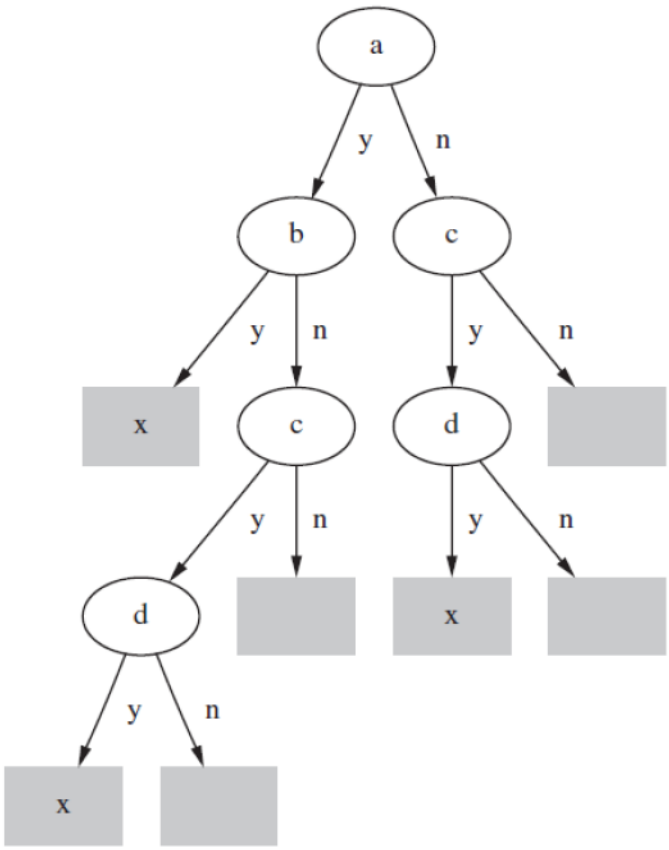


Figura 2.7: Árbol de decisión redundante para una disyunción simple

En ella son distribuidos todos los ejemplos, según la clase que la regla predice (filas de la tabla) y la clase real de los ejemplos (columnas de la tabla). Siendo A el antecedente de la regla y C denotando la clase de su consecuente el significado de su contenido es el siguiente:

- *TP* (del inglés True Positives) es el número de ejemplos que satisfacen A y C.
- *FP* (del inglés False Positives) es el número de ejemplos que satisfacen A pero no
- *FN* (del inglés False Negatives) es el número de ejemplos que no satisfacen A pero si C.
- *TN* (del inglés True Negatives) es el número de ejemplos que no satisfacen A ni C.

A partir de estos conceptos el soporte y la confianza de una regla pueden expresarse utilizando las ecuaciones 2.5 y 2.6 respectivamente. Evidentemente, para que una regla sea buena, ésta debería tener valores altos de *TP* Y *TN*, y bajos de *FP* y *FN*.

$$Soporte(regla_i) = \frac{TP}{TP + FP + FN + TN} \tag{2.5}$$

$$Confianza(regla_i) = \frac{TP}{TP + FP} \tag{2.6}$$

Hasta ahora se ha supuesto que sólo hay dos clases (C y no C), produciendo una matriz de confusión de $2x2$. En caso de haber n clases se tendría una matriz de confusión de nxn. Sin embargo, para simplificar la evaluación de las reglas, cuando existen más de dos clases aún así se utilizan matrices de confusión de $2x2$, siempre y cuando se evalúe con la matriz una única regla a la vez. En este caso, la clase C predecida por la regla es considerada la clase positiva, y todas las otras clases son consideradas clases negativas.

Cuando se trata de una lista de clasificación, se deben evaluar todas las reglas a la vez y por ello su desempeño general se representa, al igual que se lo hace con un árbol, a través de una tabla como la 2.2.

2.3.6. Métodos clásicos

Existen diversos métodos de construcción de reglas. Cuando se trata de obtener reglas de asociación el método que más se utiliza es el Apriori [21] o alguna de sus variantes generalmente orientadas a reducir el tiempo de cálculo. En cualquiera de los casos, se trata de identificar los conjuntos de atributos más frecuentes para luego combinarlos de todas las maneras posibles y formar las reglas. Para ello, este método tiene como parámetros la métrica a considerar (soporte, confianza, etc.) y el umbral a utilizar (1, 0.8, etc.). Si las reglas, por ejemplo, se miden por confianza con un umbral 0.75 el método retornará todas las reglas con confianza superior a dicho valor.

Para realizar clasificación, el método más simple es ZeroR que clasifica todos los ejemplos como pertenecientes a la clase mayoritaria. Por ejemplo, para los datos de la tabla 2.1, según la clase mayoritaria la respuesta siempre es $(Juega? = Si)$ con una tasa de acierto del 64.3 %. Este método si bien no es muy preciso en algunas ocasiones puede dar buenos resultados.

El método 1R [22], a diferencia de ZeroR, clasifica en base a un único atributo. Para ello, por cada atributo y cada uno de sus valores crea una regla de la forma "`SI` $(atributo = valor)$ `ENTONCES` $(clasemasfrecuente)$" a la que se le calcula su error. Finalmente, selecciona las reglas del atributo con el error total más bajo. Este método es fácil de aplicar pero tiende a obtener muchas reglas que no necesariamente dan una clasificación exacta. Puede trabajar con datos faltantes y atributos numéricos.

Otra opción para obtener reglas de clasificación es la utilización del método Apriori aunque como pudo verse es un método más general. Para ello, deberían filtrarse las reglas que posean un determinado consecuente o utilizar su versión predictiva [23].

Si en vez de obtener reglas independientes, se quiere obtener una lista de clasificación, el algoritmo PRISM [24] puede ser usado siempre y cuando se trate de atributos nominales. Con este método fueron obtenidas las reglas de la figura 2.5.

En muchas ocasiones es necesario reorganizar y refinar las reglas obtenidas para mejorar el modelo. Hay métodos como RIPPER [25] que optimizan el conjunto de reglas construído con el fin de reducir su tamaño y mejorar su ajuste, descartando reglas que juntas funcionan mejor que por separado.

Hay otros métodos de construcción de listas que se basan en árboles y además optimizan las reglas. El método PART [16], definido por Frank y Witten en 1998, es uno de ellos

que obtiene reglas a partir de la mejor hoja de un árbol C4.5 parcial. Por derivar de dicho árbol, a diferencia de PRISM, trabaja con atributos numéricos.

A continuación se describe el método PART por ser el método contra el cual se comparan los resultados obtenidos por el método propuesto en esta tesina.

PART

El método PART, definido por Witten en [16], permite obtener una lista de reglas de clasificación; es decir, un conjunto de reglas que a la hora de aplicarse a un ejemplo, deben ejecutarse de manera ordenada hasta encontrar la primera regla que el ejemplo en cuestión verifique. Dicha regla tendrá en su consecuente la clase a asignar.

Es importante notar la diferencia entre un conjunto de reglas independientes y una lista de clasificación. Si las reglas son independientes, podrían superponerse en el espacio de entrada permitiendo que un mismo ejemplo pertenezca a dos clases distintas. Por el contrario, en una lista de clasificación, las reglas se analizan de una por vez comenzando por la que se generó primero hasta hallar una que cumpla con el ejemplo. La lista incluye una clase por defecto la cual será asignada a los ejemplos que no cumplan con ninguna regla. Es decir que al utilizar una lista de clasificación, un ejemplo dado sólo puede ser asignado a una clase.

Por otro lado, las reglas que componen la lista de clasificación que se genera con el método PART no son perfectas. Su construcción requiere de la definición previa de un árbol de clasificación parcial. Se dijo previamente que las ramas de un árbol de clasificación pueden verse como reglas si se recorren todas sus ramas desde la raíz hasta las hojas; sin embargo el costo de construir el árbol completo suele ser computacionalmente alto. Por tal motivo PART utiliza un árbol construído parcialmente. Esto último implica que en cierto punto del proceso, el árbol deja de construirse y a partir de las ramas que terminen en hojas se decide la regla que se va a generar. La rama elegida será la que presente mayor cobertura.

El algoritmo de construcción del árbol inicia seleccionando el atributo que mejor particiona los ejemplos. Para ello se utiliza la *Tasa de Ganancia* definida en 2.4. Luego, para cada rama del atributo seleccionado, se calcula su entropía utilizando la ecuación 2.1. Como resultado se tendrá un valor para cada subconjunto de ejemplos y la expansión del árbol se hará en orden comenzando por el subconjunto de menor valor, es decir, por la rama más prometedora. Esto se repite recursivamente hasta que todos los subconjuntos expandidos sean hojas.

El algoritmo 2 describe el proceso de expansión mencionado.

En este punto se inicia un proceso de poda del subárbol generado comenzando por las hojas y en dirección a la raíz. La poda sólo se llevará a cabo si el error en el subárbol es superior al error en el nodo. De ser así, la expansión realizada se anula y el nodo se convierte en hoja asignando los ejemplos a la clase mayoritaria.

Este proceso se repite recursivamente hasta que todos los subconjuntos expandidos se conviertan en hoja. Es decir que si en algún momento el proceso de poda ésta no se lleva a cabo, la construcción del árbol termina.

Algoritmo 2: Expansión de conjunto de ejemplos para formar un árbol parcial

```
Function ExpandirSubconj(S)
begin
    Seleccionar un atributo utilizando la Tasa de Ganancia
    Utilizar el atributo seleccionado para dividir los ejemplos
    while ( haya un subconjunto X sin expandir) and (todos los subconjuntos
    expandido sean hojas do
        ExpandirSubconj(X)
    if (todos los subconjuntos expandidos son hojas) and (el error estimado del
    subárbol es mayor al estimado en el nodo) then
        Deshacer la expansión en subconjuntos y convertir el nodo en hoja
```

Una vez finalizada la construcción del árbol parcial, se selecciona la rama con mayor cobertura que finaliza en una hoja.

Luego de haber obtenido la regla, los ejemplos que ella cubre se eliminan del conjunto de datos de entrada y el árbol se descarta. Este proceso se repite hasta alcanzar la cobertura deseada.

Con respecto a la métrica utilizada para estimar el error cometido en un nodo al considerar la clase mayoritaria, se utiliza un cálculo basado en un razonamiento estadístico. Se trata de una heurística ya que varios aspectos matemáticos no han sido tenidos en cuenta al momento de su aplicación.

Considérese que al clasificar un conjunto de ejemplos de tamaño N utilizando su clase mayoritaria se producen E errores. Si la clasificación de los ejemplos se lleva a cabo en forma independiente, es decir si la probabilidad p de que se cometa un error es siempre la misma, esta situación puede verse como un experimento binomial donde el tamaño de muestra es N y la cantidad de éxitos (en este caso son los ejemplos mal clasificados) dentro de la muestra puede verse como una variable aleatoria X con distribución binomial con valor esperado $\mu_X = Np$ y desviación $\sigma_X = \sqrt{N\ p(1-p)}$.

Dado que se está trabajando sobre los ejemplos de entrenamiento, se utilizará una estimación pesimista de la tasa de error utilizando el límite superior de un intervalo de confianza.

Para un valor c indicado a priori, los límites del intervalo de confianza para la verdadera proporción p se obtienen calculando la siguiente probabilidad

$$P\left(-z < \frac{\hat{p} - p}{\sqrt{p(1-p)/N}} < z\right) = 1 - c \tag{2.7}$$

siendo $\hat{p}$ la proporción observada en la muestra. En este caso, $\hat{p} = E/N$. La ecuación 2.7 surge al aproximar la distribución binomial de X por una distribución normal. Esto sólo puede hacerse bajo algunas condiciones de la muestra que en este caso no están siendo consideradas.

Bajo estas suposiciones, el límite superior del intervalo de confianza que se utilizará para estimar el peor error e que se puede cometer en el nodo es

$$e = \frac{\hat{p} + \frac{z^2}{2N} + z\sqrt{\frac{\hat{p}}{N} - \frac{\hat{p}^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}} \tag{2.8}$$

En base a la distribución de X, a partir de la ecuación 2.7 se obtiene el valor de z a utilizar. Por ejemplo, en el método C4.5 y el PART se utiliza $c - 0{,}25$ y $z = 0{,}69$.

A continuación se muestra un ejemplo concreto para obtener la primera regla aplicando el método PART a los ejemplos de la tabla 2.4. El proceso comienza con la elección del atributo que se ubicará en la raíz del árbol parcial. Para ello es preciso calcular la tasa de ganancia para cada uno de los atributos posibles y tomar el de mayor valor. Esta es la información que se muestra en la tabla 2.5. El atributo seleccionado es "*% materias aprobadas*" y el árbol parcial hasta el momento es el que se muestra en la figura 2.8(a).

Luego de elegir el nodo raíz, se han formado 4 ramas donde sólo una de ellas es pura. El análisis continua por la primera rama no pura con la menor entropía. En este caso es la rama "*(% materias aprobadas = [0, 0.25])*".

Para los 9 ejemplos que la forman, se debe volver a seleccionar el atributo con mayor ganancia. La tabla 2.6 muestra los valores obtenidos y el atributo seleccionado. Al agregar al árbol el atributo `Trabaja` se separan los ejemplos como se observa en la figura 2.8(b). Se continua hacia abajo en el árbol seleccionando la rama con menor entropía. El proceso sigue por la rama "*(% materias aprobadas = [0, 0.25]) AND (Trabaja = SI)*" y el nuevo atributo seleccionado es el "*Promedio*" (ver figura 2.8(c)). Esta es la mayor profundidad que alcanzará esta rama porque aunque una de la hojas no es pura, el método PART al igual que el método C4.5 aplica un criterio de prepoda exigiendo que cada hoja tenga un número mínimo de ejemplos (en este caso 2).

Es el momento de decidir si se produce un retroceso o no. Para ello debe calcularse el error en el nodo antes y después de generar el subárbol. La figura 2.9 muestra ambas opciones con su correspondiente valor de error. Puede observarse que es menor el error antes de haber elegido el atributo "*Promedio*" y por lo tanto, el subárbol es descartado considerando al subconjunto de ejemplos como si fuera una hoja. Los ejemplos que pertenecen a ella serán etiquetados con la clase mayoritaria.

La figura 2.10(a) ilustra el árbol parcial construido hasta el momento. Resta analizar los 4 ejemplos de la rama "*(% materias aprobadas = [0, 0.25]) AND (Trabaja = NO)*". El cálculo de entropía para los dos atributos que quedan se muestra en la tabla 2.7 y nuevamente el atributo "*Promedio*" es seleccionado dando lugar al árbol parcial de la figura 2.10(b).

Este último atributo clasifica correctamente los 4 ejemplos involucrados y porl o tanto el proceso de expansión terminar.

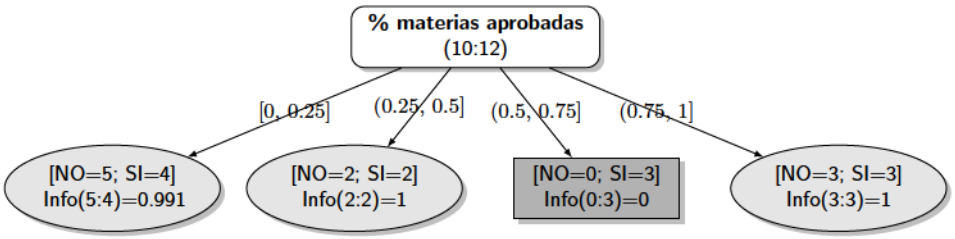
Nuevamente en este punto corresponde analizar si la postpoda debe llevarse a cabo. Para ello, se calculan los errores cometidos antes y después de agregar el último subárbol. Dichos valores son 0.4203 y 0.1834 respectivamente y se encuentran indicados en la figura 2.11. Por lo tanto, como el subárbol tiene menos error que el nodo original, la postpoda no se realiza y el proceso de construcción del ábrol parcial finaliza en este punto.

Trabaja	Estado Civil	% materias aprobadas	Promedio	Alumno regular
NO	Casado	(0.25, 0.5]	[0, 4)	NO
NO	Casado	(0.25, 0.5]	[0, 4)	NO
SI	Casado	(0.25, 0.5]	[0, 4)	SI
SI	Casado	(0.25, 0.5]	[0, 4)	SI
NO	Soltero	(0.5, 0.75]	[4, 6)	SI
SI	Casado	(0.5, 0.75]	[4, 6)	SI
NO	Casado	(0.5, 0.75]	[6, 10]	SI
SI	Casado	(0.75, 1)	[0, 4)	SI
SI	Soltero	(0.75, 1)	[4, 6)	NO
NO	Soltero	(0.75, 1)	[6, 10]	NO
SI	Soltero	(0.75, 1)	[6, 10]	SI
NO	Soltero	[0, 0.25]	[0, 4)	SI
NO	Casado	[0, 0.25]	[0, 4)	SI
NO	Soltero	[0, 0.25]	[0, 4)	SI
SI	Casado	[0, 0.25]	[0, 4)	NO
SI	Soltero	[0, 0.25]	[0, 4)	NO
NO	Casado	[0, 0.25]	[4, 6)	NO
SI	Casado	[0, 0.25]	[4, 6)	NO
SI	Soltero	[0, 0.25]	[4, 6)	SI
SI	Soltero	[0, 0.25]	[6, 10]	NO
NO	Casado	(0.75, 1)	[6, 10]	NO
SI	Soltero	(0.75, 1)	[4, 6)	SI

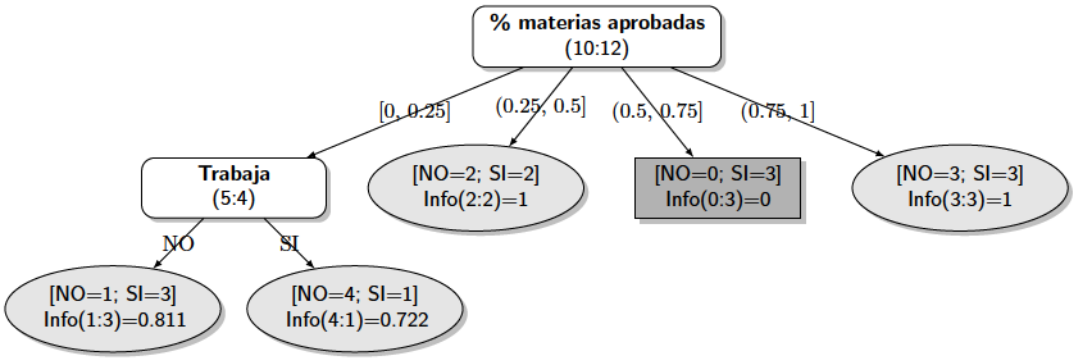
Tabla 2.4: Información referida a 24 alumnos que espera ser utilizada para determinar las características principales que determinan su condición de regularidad

Atributo	Tasa de ganancia	
Trabaja	0.005041	
Estado Civil	0.007278	
% materias aprobadas	**0.071381**	(ATRIB. SELECC)
Promedio	0.012127	

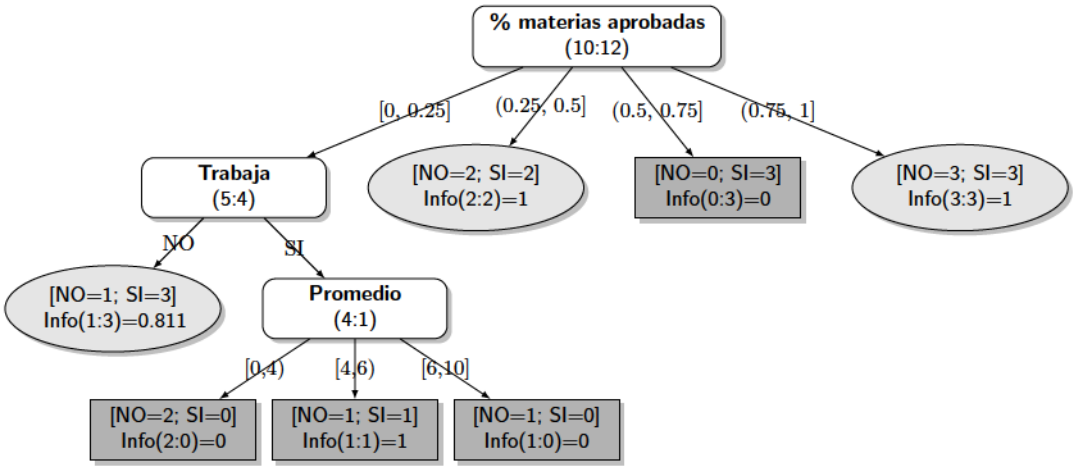
Tabla 2.5: Tasa de ganancia calculada para cada atributo de la tabla 2.4 con el objetivo de determinar la raíz del árbol parcial



(a) Nodo raíz



(b) Arbol parcial de nivel 1

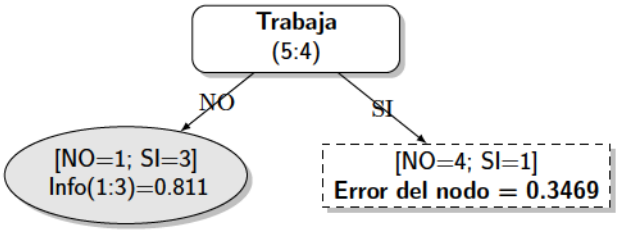


(c) Arbol parcial de nivel 2

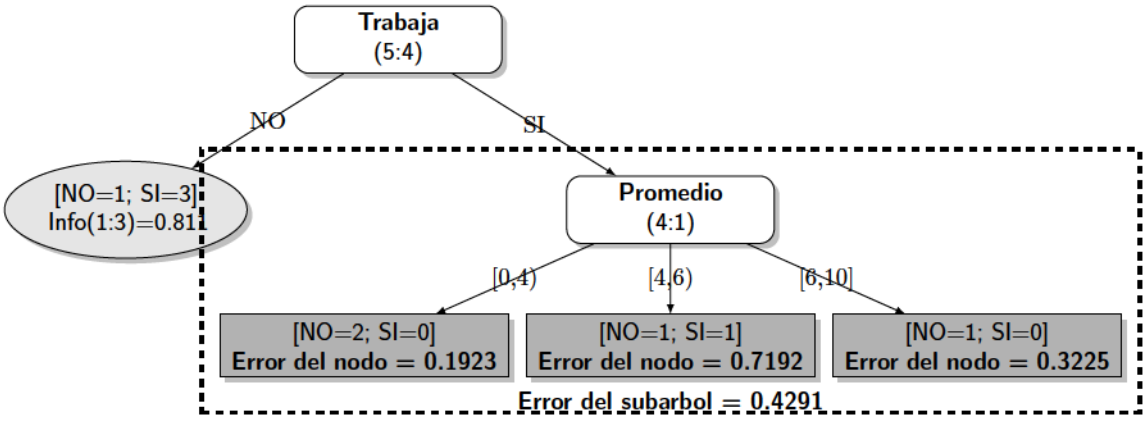
Figura 2.8: Construcción de la regla 1

Atributo	Tasa de ganancia	
Trabaja	**0.231503**	(ATRIB. SELECC)
Estado Civil	0.091911	
Promedio	0.107691	

Tabla 2.6: Selección del atributo que mejor separa los ejemplos correspondientes a la rama "*(% materias aprobadas = [0, 0.25]) AND (Trabaja = NO)*". Este atributo se ubicará en el nivel 2 del árbol parcial

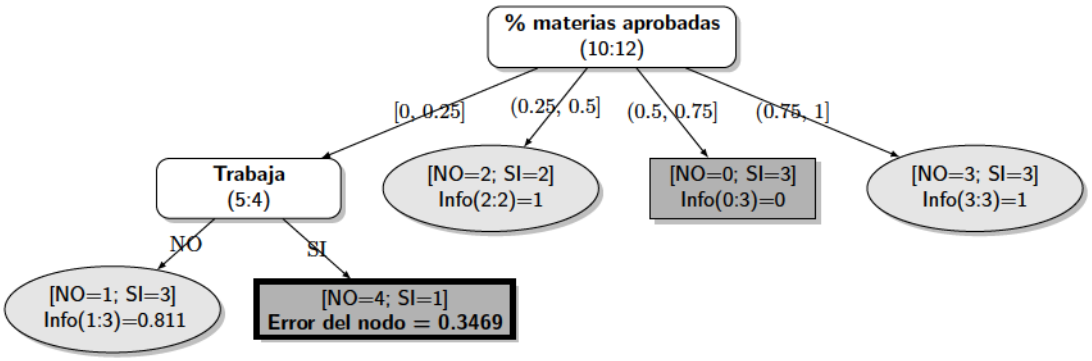


(a)

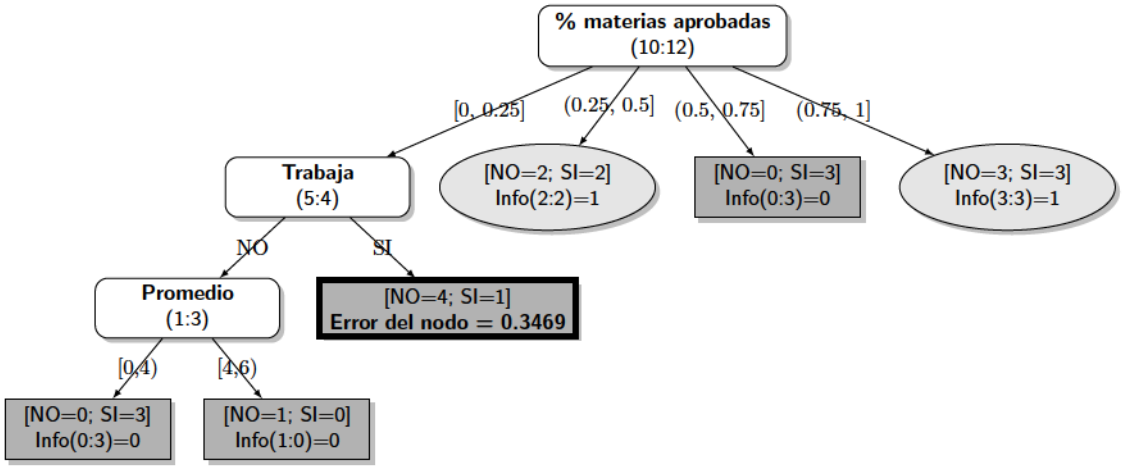


(b)

Figura 2.9: El error correspondiente a los datos de la rama "*(% materias aprobadas = [0, 0.25]) AND (Trabaja = SI)*" es 0.3469 mientras que el correspondiente al subárbol generado a partir del atributo '*Promedio*" es 0.4291. Por ser este último mayor, el subárbol se descarta



(a) Arbol parcial de nivel 2 con subárbol podado

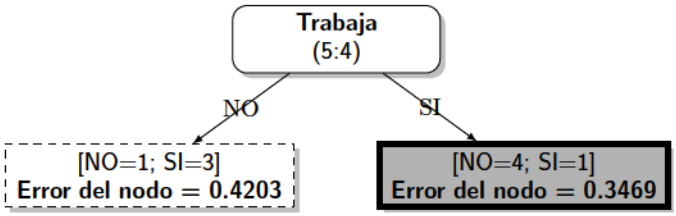


(b) Arbol parcial de nivel 3 con subárbol podado

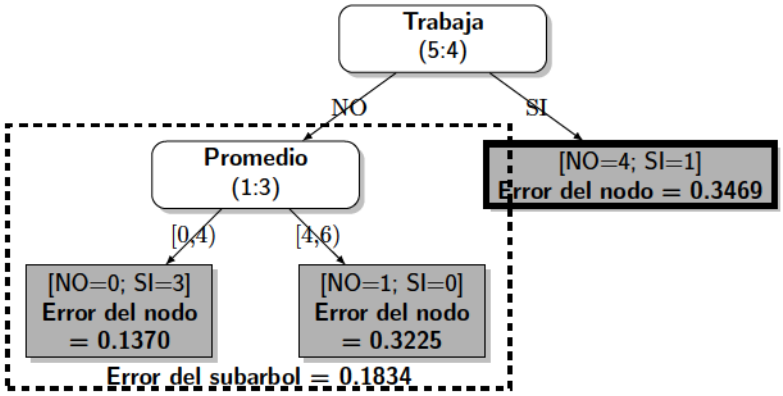
Figura 2.10: Construcción de la regla 1

Atributo	Tasa de ganancia	
Estado Civil	0.311278	
Promedio	**1.000000**	(ATRIB. SELECC)

Tabla 2.7: Selección del atributo que mejor separa los ejemplos correspondientes a la rama "*(% materias aprobadas = [0, 0.25]) AND (Trabaja = NO)*". Este atributo se ubicará en el nivel 2 del árbol parcial



(a)



(b)

Figura 2.11: El error correspondiente a los datos de la rama "*(% materias aprobadas = [0, 0.25]) AND (Trabaja = NO)*" es 0.4203 mientras que el correspondiente al subárbol generado a partir del atributo "*Promedio*" es 0.1834. Por ser este último menor, el subárbol no se descarta

Han quedado entonces 4 ramas que terminan en hoja y por lo tanto, pueden ser seleccionadas como la primera regla. Las opciones son:

1. **SI** $(\%\ materias\ aprobadas = (0{,}5, 0{,}75])$
   ENTONCES $(Alumno\ regular = SI)$

2. **SI** $(\%\ materias\ aprobadas = [0, 0{,}25])$
   AND $(Trabaja = SI)$
   ENTONCES $(Alumno\ regular = NO)$

3. **SI** $(\%\ materias\ aprobadas = [0, 0{,}25])$
   AND $(Trabaja = NO)$
   AND $(Promedio = [0, 4))$
   ENTONCES $(Alumno\ regular = SI)$

4. **SI** $(\%\ materias\ aprobadas = [0, 0{,}25])$
   AND $(Trabaja = NO)$
   AND $(Promedio = [4, 6))$
   ENTONCES $(Alumno\ regular = NO)$

En la figura 2.12 puede verse que la cobertura de las opciones 1 y 3 es de 3 ejemplos cada una, la opción 4 sólo cubre 1 ejemplo mientras que la opción 2 cubre 4 ejemplos y por lo tanto es la rama seleccionada como primera regla.

A continuación se quitan del conjunto de ejemplos originales (tabla 2.4) los 5 casos cubiertos por la regla recientemente creada (opción 2) y se repite el proceso nuevamente. La próxima regla que se obtendrá de la construcción del siguiente árbol parcial es **SI** $(Trabaja = SI)$ **ENTONCES** $(Alumno\ regular = SI)$.

2.4. Conclusiones

Dado que el método propuesto en esta tesina se refiere a la modelización de la información disponible utilizando un conjunto de reglas de clasificación, en este capítulo se han descripto brevemente los métodos más utilizados en la literatura.

Se comenzó presentando una de las alternativas más comunes que consiste en la transformación de un árbol de clasificación en un conjunto de reglas a través del recorrido de cada una de sus ramas.

Se analizaron básicamente dos métodos de construcción de árboles, ambos propuestos por Quinlan: ID3 y C4.5. La elección de estos métodos radica no sólo en su popularidad sino que además, constituyen la base del método PART definido por Witten contra el cual se compara la propuesta de esta tesina.

El método PART propone una alternativa interesante para la construcción de reglas y es el uso de árboles parcialmente construidos. Esto tiene como finalidad principal lograr reglas más generales en un tiempo menor.

El ejemplo desarrollado en la sección anterior ilustra claramente los pasos a seguir para obtener el modelo buscado. Aunque sólo se detalló la primera parte del proceso, puede

verse que se trata de un método totalmente determinístico que si bien no requiere construir el árbol de clasificación completo insume un costo de análisis considerable.

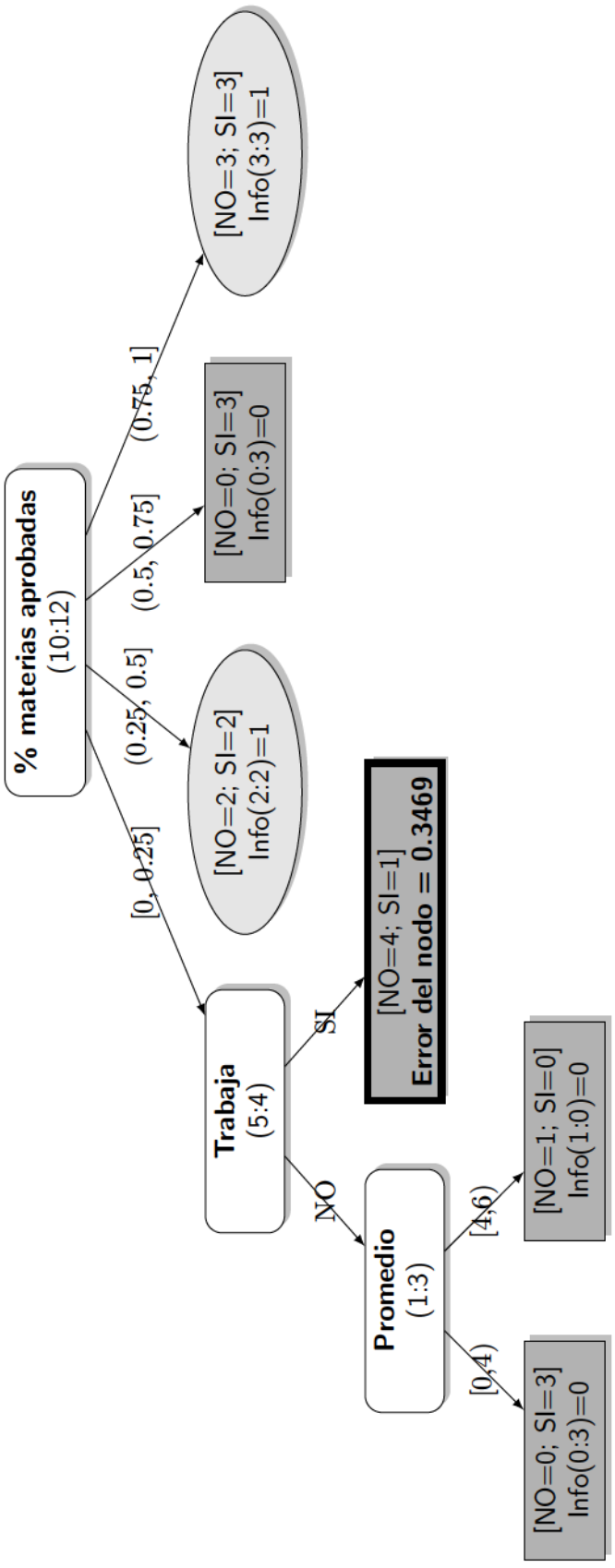


Figura 2.12: Arbol parcial del que se extrae la regla 1

Capítulo 3

Clustering con Redes Neuronales

3.1. Introducción

La técnica de $\mathcal{DM}$ propuesta en esta tesina combina una Red Neuronal con una Técnica de optimización para obtener un modelo de clasificación.

En particular, la Red Neuronal utilizada permite agrupar la información disponible según una medida de similitud dada.

Las técnicas de agrupamiento o técnicas de clustering buscan que los elementos de un mismo grupo sean lo más homogéneos posible y a la vez muy distintos de los elementos contenidos en los clusters restantes.

Se trata de una tarea relevante para la Minería de Datos ya que permite encontrar características generales a partir de las cuales puede generarse un modelo descriptivo de los datos. Por ejemplo, utilizando agrupamientos pueden encontrarse las características comunes de los alumnos que aprueban un curso o las características principales que permiten identificar una transacción como fraudulenta o no.

Además, se espera que los clusters sean hallados sin información previa y sugeridos únicamente por la propia esencia o naturaleza de los datos. Es por eso que la obtención de estos grupos se basa en la proximidad o lejanía de los datos y por lo tanto, resulta importante el uso adecuado del concepto de distancia.

En este capítulo se introduce el análisis de clusters, se detallan algunas medidas de distancia y se presentan formas de clasificar las técnicas de clustering. De entre todos los algoritmos existentes, al final de este capítulo será desarrollado, en especial, SOM (del inglés Self-Organizing Maps) [26] por presentar notorias ventajas frente a otros métodos, razón por la cual es utilizado en esta tesina para el desarrollo del método propuesto.

3.2. Problema de clustering

Clasificar casos u objetos en diferentes grupos es una necesidad cotidiana donde hace falta poner orden y agrupar, por ejemplo, desde pacientes, libros, alumnos hasta animales, organismos y la información que tantas veces buscamos en internet.

Nro. de Ejemplo	Cant. de Materias	Antiguedad (años)	Promedio
1	20	4	7.56
2	7	8.5	7.2
3	8	9.5	8.25
4	2	1	6.4
5	0	0.5	6
6	2	1.5	5.5
7	3	1.5	7
8	14	5.5	5.25
9	12	6	6
10	3	9	9
11	18	3.5	8.2
12	9	8.5	7.6
13	19	4	9
14	15	4	5.6
15	10	5	7
16	10	10	8

Tabla 3.1: Información no etiquetada de 16 alumnos de una unidad académica

La Minería de Datos presenta una variedad de métodos capaces de agrupar la información disponible de manera que aquellos casos que presenten características muy similares formen un mismo grupo. Cuanto más compactos sean los agrupamientos y más separados estén entre ellos, mejor será la técnica de clustering utilizada.

Por ejemplo, suponga que dispone de información referida al desempeño académico de los alumnos de cierta unidad académica y que las características relevadas son: cantidad de materias aprobadas, promedio general y antiguedad del alumno en la carrera (cantidad de años transcurridos desde que comenzó). La tabla 3.1 contiene un conjunto reducido de ejemplos con estas características.

Nótese que no se ha asociado a priori ninguna etiqueta o clase a cada alumno sino que se espera que sea la técnica de $\mathcal{KD}$ la que los relaciones en base a una medida de similitud o distancia.

Para llevar a cabo esta tareas hay distintos enfoques según la forma en que se representa la cercanía o parecido entre ejemplos y se dividen en dos grandes categorías:

- Jerárquicas: Resultan apropiadas cuando se desconoce la cantidad de grupos que se desean formar. Su funcionamiento se basa en relacionar los ejemplos utilizando una medida de distancia. Pueden llevar a cabo esta tarea considerando inicialmente a cada ejemplo como un agrupamiento aislado, luego los agrupamientos se van uniendo de a pares a medida que ascienden en la jerarquía. El otro enfoque opera en sentido inverso, considerando que todas las observaciones pertenecen a un único grupo el cual se va dividiendo recursivamente a medida que baja en la jerarquía. El primer enfoque se conoce como agrupamiento jerárquico aglomerativo y el segundo

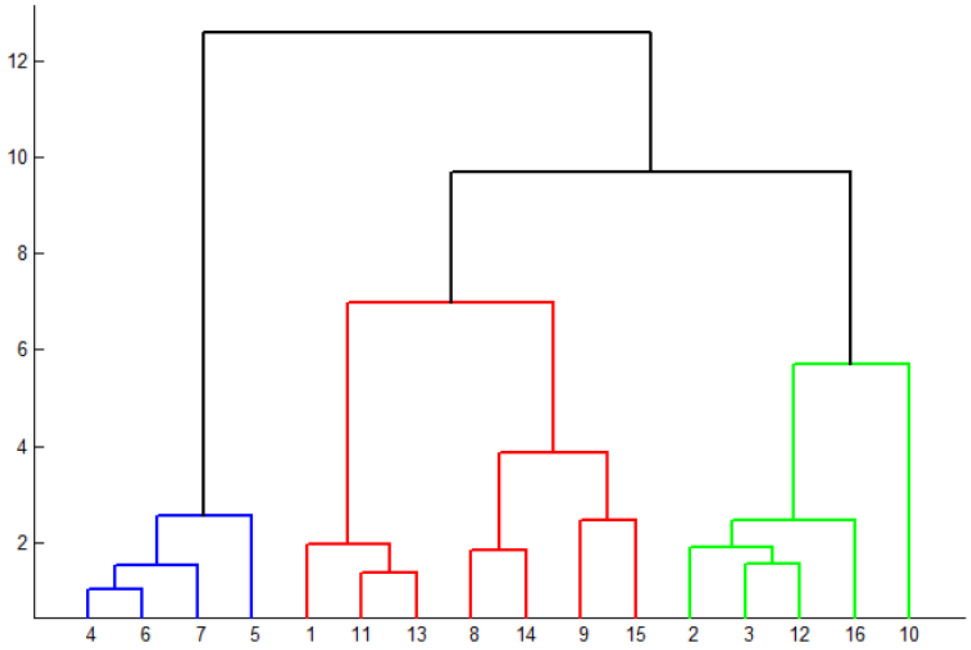


Figura 3.1: Agrupamiento jerárquico aglomerativo realizado sobre los datos de la tabla 3.1. Los números sobre el eje horizontal representan el número de ejemplo

agrupamiento jerárquico divisorio.

El resultado de este tipo de agrupamiento se representa generalmente en forma de dendrograma (*"dendrogram"*). La figura 3.1 ilustra un agrupamiento jerárquico aglomerativo realizado sobre los datos de la tabla 3.1.

- Partitivo: en los que el número de grupos se determina de antemano y las observaciones se van asignando a los grupos en función de su cercanía. A esta categorías pertenecen técnicas como K-medias [15] y las redes neuronales SOM. Cada grupo tiene un representante o centroide que permite resumir la información del grupo. La figura 3.2 ilustra este tipo de agrupamiento.

Nótese que en ambos casos, el modelo obtenido se resume a través de relaciones entre los ejemplos a partir de una medida de parecido especificada a priori. Luego de haber formado los agrupamientos, es el experto quien puede asignarle una etiqueta reconociendo en ellos una situación específica. Por ejemplo, el dendrograma de la figura 3.1 indica que los ejemplos 2, 3, 12, 16 y 10 son similares. Si se observa la tabla 3.1 puede verse que se trata de alumnos con buen promedio que han rendido pocas materias y que hace mucho que están inscriptos en la carrera. También se puede ver la relación entre los ejemplos 1, 11 y 13 correspondiente a alumnos que llevan la carrera al día con buenas calificaciones.

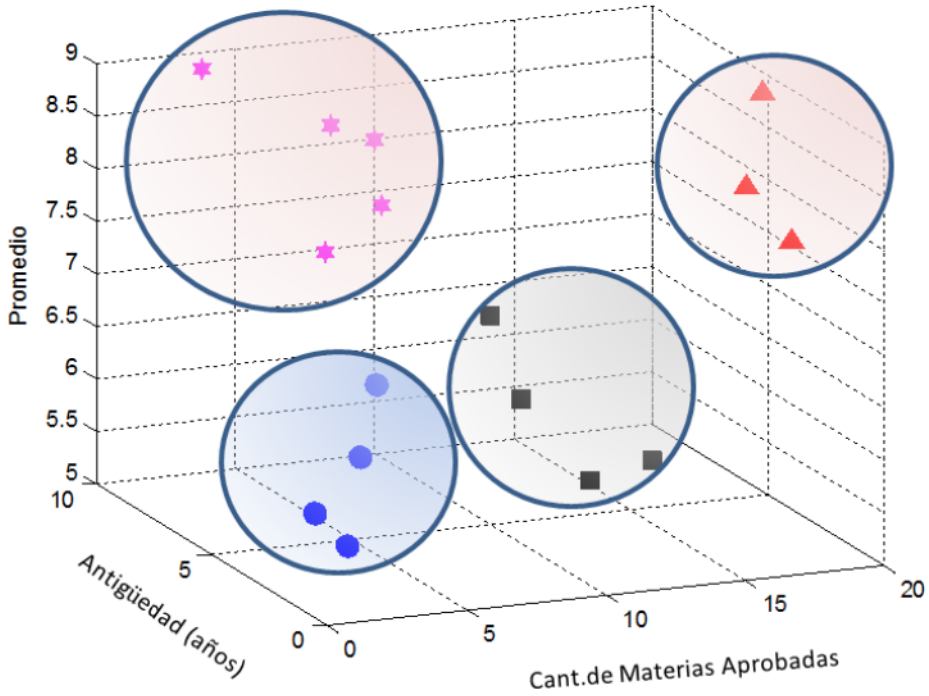


Figura 3.2: Agrupamiento partitivo realizado sobre los datos de la tabla 3.1. Cada grupo posee un centroide que resume las características de los elementos que lo forman

3.3. Distancias y similitudes

Las técnicas de agrupamiento, a diferencia de las de clasificación, modelizan la información sin necesidad de contar a priori con ejemplos etiquetados. Es decir que el modelo se obtiene de manera no supervisada ya que para un ejemplo suministrado se desconoce a priori la clase a la que pertenece. Esta carencia de etiquetado inicial se reemplaza por una medida de similitud o distancia, es decir que, es necesario incorporar un criterio que determine cuándo dos ejemplos son parecidos y cuándo no lo son.

Este es precisamente uno de los aspectos más interesante de las técnicas de agrupamiento: el mismo método puede funcionar de manera diferente, variando la métrica de distancia. Dicho de otra manera, la función de distancia a utilizar puede considerarse un parámetro de la técnica a aplicar (al igual que ocurre con el criterio con el que se eligen los atributos que componen un árbol de decisión según lo visto en la sección 2.2.1).

A continuación, se verán los conceptos y propiedades referidos a una medida de distancia.

3.3.1. Propiedades generales

Teniendo en cuenta que el objetivo principal es hallar clusters que contengan casos similares, va a ser necesario medir las similitudes o bien las distancias que hay entre los casos.

Para poder operar matemáticamente con los ejemplos es necesario contar con una representación para cada uno de ellos.

Sea X el conjunto de ejemplos sobre el cual se realizará el agrupamiento. X estará formado por m ejemplos de n atributos cada uno. Es decir que cada ejemplo será representado como un vector número n-dimensional.

Existen varias formas de considerar la distancia que separa dos vectores del espacio de datos n-dimensional. Matemáticamente hablando, se consideran funciones de distancia sólo las funciones definidas adecuadamente $d : X \times X \to \mathbb{R}$ y que cumplen las siguientes propiedades:

- $d(i, j) \geq 0$
- $d(i, i) = 0$
- $d(i, j) = d(j, i)$

La primera de las propiedades dice que la distancia entre dos elementos i y j cualesquiera de los $1, 2, \cdots, m$ disponibles en X debe ser positiva. La segunda que cada caso no puede distar de sí mismo. Finalmente la última propiedad establece simetría lo que equivale a decir que la distancia de i a j es la misma que la de j a i.

Cuando además de las tres propiedades anteriores se cumple la desigualdad triangular, $d(i, j) \leq d(i, k) + d(k, j)$, se dice que la distancia es métrica y que (X,d) forma un espacio métrico. Esta propiedades significa que la distancia entre dos ejemplos cualesquiera, i y j, es menor o igual que la suma de la distancia de i a un tercer ejemplo k, más la distancia de k a j.

Como el número de casos m es finito, es posible ordenar las interdistancias en una matriz simétrica mxm, denominada matriz de distancias D sobre X donde $d(i, j) = d_{ij}$:

$$D = \begin{bmatrix} d_{11} & d_{12} & d_{13} & \cdots & d_{1m} \\ d_{21} & d_{22} & d_{23} & \cdots & d_{2m} \\ d_{31} & d_{32} & d_{33} & \cdots & d_{3m} \\ \vdots & & & & \vdots \\ d_{m1} & d_{m2} & d_{m3} & \cdots & d_{mm} \end{bmatrix}$$

El concepto dual a la distancia es la similaridad. Al igual que una función de distancia, una función de similaridad s sobre un conjunto X es una función $s : X \times X \to \mathbb{R}$ que verifica ciertas propiedades:

- $0 \leq s(i, j) \leq 1$
- $1 = s(i, i) \geq s(i, j)$
- $s(i, j) = s(j, i)$

La primera propiedad dice que la similaridad debe ser no negativa y establece una escala. La segunda, que cada caso se parece a sí mismo más que a cualquier otro caso y la última establece simetría. En cuanto a la interpretación se puede decir que cuanto mayor sea la similaridad $s(i, j)$, más parecidos entre sí serán los casos i y j. De la misma manera que se construyó la matriz de distancias se puede construir la matriz de similaridades sobre X:

$$S = \begin{bmatrix} s_{11} & s_{12} & s_{13} & \cdots & s_{1m} \\ s_{21} & s_{22} & s_{23} & \cdots & s_{2m} \\ s_{31} & s_{32} & s_{33} & \cdots & s_{3m} \\ \vdots & & & & \vdots \\ s_{m1} & s_{m2} & s_{m3} & \cdots & s_{mm} \end{bmatrix}$$

Una vez construida una matriz de similaridades S hay varias formas de pasar a una matriz de distancias sobre X y viceversa. Dados dos casos i y j, la transformación de Gower [27] asegura que se cumple la siguiente relación:

$$d_{ij}^2 = s_{ii} + sjj - 2s_{ij} \tag{3.1}$$

Nótese que como $s_{ii} = s_{jj} = 1$, es equivalente a $d_{ij} = \sqrt{1 - s_{ij}}$. Existen otras tansformaciones como la distancia complemento $d_{ij} = 1 - s_{ij}$ o la raíz del complemento al cuadrado $d_{ij} = \sqrt{1 - s_{ij}^2}$.

Las propiedades geométricas de las distancias pueden ser, en algunos casos, de utilidad para su manejo e interpretación.

Una pregunta que surge a partir de tantos conceptos desarrollados es cuántas funciones de distancia y similitud hay.

Sobre cada caso de X se han medido n variables y por tanto, cada caso puede ser representado como un vector $x = (x_1, \cdots, x_n)$ de $\mathbb{R}^n$, de manera que cada x_i es el valor que toma la i-ésima variable medida sobre el caso. Dependiendo de la naturaleza de las variables que se hayan considerado (variables continuas, binarias o mixtas), se deben utilizar diferentes tipos de distancias o similaridades. Además, el uso de una medida u otra también depende de la naturaleza de los datos, es decir, si provienen de un estudio genético, ecológico, industrial, etc. Así pues hay una enorme variedad de diferentes funciones de distancias y similaridad. A continuación se describen las medidas de distancia más habituales.

3.3.2. Medidas de distancia

Como se verá a continuación, además de la distancia euclídea o clásica, existen otras funciones que también cumplen los requisitos de una función de distancia (llamada entonces métrica) y que pueden funcionar mejor en ciertos contextos. Más aún, dependiendo de la aplicación, se pueden definir funciones *ad-hoc* que actúen como métrica de distancia.

Las medidas de distancia más tradicionales son aquellas que se aplican sobre dos instancias o ejemplos, tales que todos los atributos son numéricos. Por ejemplo, se pueden

definir las distancias entre dos vectores, puntos o instancias x e y de dimensión n de muy distintas formas. A continuación se presentan algunas de ellas:

- **Distancia Euclídea**. Es la distancia clásica, como la longitud de la recta que une dos puntos en el espacio euclídeo:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

- **Distancia de Manhattan**. También llamada distancia por cuadras (city-block). Tal como su nombre indica, hace referencia a recorrer un camino no en diagonal (por el camino más corto), sino zigzagueando, como se haría en Manhattan:

$$d(x,y) = \sum_{i=1}^{n}|x_i - y_i|$$

- **Distancia de Minkowsky** ($q \geq 1$). Cuando $q = 2$ ésta se reduce a la distancia Euclídea y cuando $q = 1$, se obtiene la distancia de Manhattan:

$$d(x,y) = \left(\sum_{i=1}^{n}|x_i - y_i|^q\right)^{\frac{1}{q}}$$

- **Distancia de Chebychev**. Simplemente calcula la discrepancia más grande en alguna de las dimensiones:

$$d(x,y) = max_{i=1..n}|x_i - y_i|$$

- **Distancia del coseno**. Considerando que cada ejemplo es un vector, la distancia sería el coseno del ángulo que forman:

$$d(x,y) = \arccos\left(\frac{x^T y}{||x|| \,.\, ||y||}\right) \tag{3.2}$$

- **Distancia de Mahalanobis**. Todas las distancias anteriores asumen, en cierto modo, que los atributos son independientes (es decir, consideran cada atributo una dimensión ortogonal a las demás). Esta distancia es más robusta y utiliza la matriz de covarianzas S:

$$d(x,y) = \sqrt{(x-y)^T S^{-1}(x-y)} \tag{3.3}$$

3.4. Clasificación a través del agrupamiento

Como ya se ha visto, y en especial en el capítulo 2, cuando se aprende de ejemplos, casos o datos conocidos, generalmente lo que se busca es poder tomar una decisión sobre nuevos casos.

Parece de sentido común pensar que ante una nueva situación se deberá actuar como se hizo en situaciones anteriores parecidas o similares, si en éstas se tuvo éxito. En consecuencia, parece una regla bastante clara y natural que puede ser aplicada a multitud de tareas de minería de datos.

Tanto la clasificación como el agrupamiento, son tareas que basan su funcionamiento en la identificación de características comunes en los datos de entrada.

Su mayor diferencia radica en la posibilidad de contar con una etiqueta para el grupo. Esta etiqueta es requerida sólo en las técnicas de clasificación y puede ser obtenida en el momento de la recolección de los datos o a posteriori a través un proceso manual llevado a cabo por un experto.

Ante la llegada de un nuevo ejemplo, éste será asignado al grupo más parecido y si dicho grupo posee una etiqueta asignada, podrá ser clasificado a través de ella.

Nótese que el modelo obtenido a través del agrupamiento cumple un rol netamente descriptivo ya que resume de alguna forma la relación entre los ejemplos que le dieron origen.

3.5. Redes Neuronales y clustering

Las Redes Neuronales Artificiales o simplemente Redes Neuronales, buscan emular el comportamiento del cerebro humano en lo que se refiere al procesamiento de información imprecisa y a la capacidad de aprender sin instrucciones explícitas creando representaciones internas adecuadas. Su estructura está formada básicamente por neuronas artificiales y conexiones entre ellas, donde una neurona artificial al ser "activada" envía señales a las neuronas artificiales a las cuales está conectada, llamadas neuronas vecinas. Desde el punto de vista de la implementación la estructura de la red es un grafo dirigido cuyos arcos almacenan el conocimiento adquirido.

Existen distintos tipos de Redes Neuronales según la clase de problema que se busque resolver.

Las Redes Neuronales competitivas son las adecuadas para resolver un problema de agrupamiento. La siguiente sección define la arquitectura más popular en esta temática.

3.6. Mapas auto-organizativos

Los mapas auto-organizativos o redes SOM (del inglés Self-Organizing Maps) fueron definidos por Kohonen en 1982 [26]. Su aplicación principal es el agrupamiento de la información disponible y se caracteriza por su capacidad para preservar la topología de los datos de entrada.

Es una técnica de clustering partitiva ya que asocia cada ejemplo a un vector promedio o centroide. Además posee la capacidad de relacionar los centroides de manera de indentificar similitudes entre ellos. Esto último es una característica propia de esta red que no está disponible en la mayoría de las técnicas de clustering basadas en centroides.

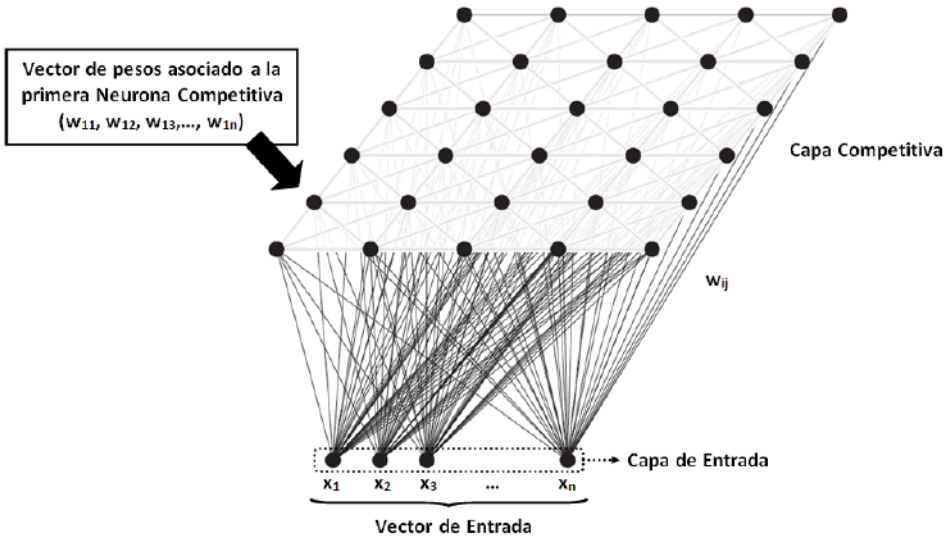


Figura 3.3: Estructura clásica de una red SOM

Por este motivo suelen utilizarse como herramienta de visualización y para reducir la dimensionalidad del espacio de entrada.

3.6.1. Descripción del modelo

Puede ser representada como una estructura de dos capas: la capa de entrada cuya función es sólo permitir el ingreso de la información a la red y la capa competitiva que es la encargada de realizar el agrupamiento. Las neuronas que forman esta segunda capa se encuentran conectadas y poseen la capacidad de identificar la cantidad de ”saltos´´ o conexiones que la separan de cada una de las restantes dentro de este nivel. Cada neurona competitiva lleva asociado un vector de pesos o centroide representado por los valores de los arcos que llegan a ella desde la capa de entrada. De esta forma, la red SOM maneja dos estructuras de información: una referida a los centroides asociados a las neuronas competitivas y otra encargada de determinar la proximidad entre neuronas. Esto, a diferencia de un método del estilo ”winner-take-all´´ como el método K-medias, brinda información adicional con respecto a los agrupamientos ya que neuronas cercanas dentro de la arquitectura representarán agrupamientos similares en el espacio de los datos de entrada.

La figura 3.3 muestra la estructura de una red SOM donde la capa de entrada está formada por un vector de n dimensiones y la capa competitiva posee 5x6=30 neuronas. Cada neurona de esta segunda capa posee 8 vecinos directos (conexiones inmediatas). Esta forma de conexión puede cambiar según el problema a resolver.

Inicialmente los pesos de la red, cuyos valores se encuentran representados en la figura mediante la matriz W, son aleatorios y se adaptan con las sucesivas presentaciones de los vectores de entrada. Se utilizará w_{ij} para denotar el peso del arco que va desde la jésima neurona de entrada hasta la iésima neurona competitiva

Por tratarse una estructura competitiva, cada vector de entrada se considera representado por (o asociado con) la neurona competitiva que posea el vector de pesos más parecido según una medida de similitud dada. El valor final de W se obtiene mediante un proceso iterativo que se repite hasta que los vectores de pesos no presenten modificaciones significativas o lo que es lo mismo, hasta que cada vector de entrada sea representado por la misma neurona competitiva que en la iteración anterior.

Durante el entrenamiento de la red SOM, en cada iteración, para cada vector de entrada $X = (x_1, x_2, ..., x_n)$, se determina la neurona que lo representa, es decir, la neurona más parecida hasta el momento. A esta neurona se le llama neurona ganadora ya que gana la competencia por la representación del vector por ser la más cercana utilizando una medida de distancia. Es decir que siendo $(w_{i1}, w_{i2}, ..., w_{in})$ el vector de pesos de la i-ésima neurona competitiva , SOM identifica a la neurona ganadora como aquella que cumple con la ecuación 3.4

$$\left\|W_{ganadora} - X\right\| = min(\|W_i - X\|) \qquad i = 1..N \tag{3.4}$$

donde *ganadora* es la neurona ganadora, $\|.\|$ es una medida de distancia, generalmente distancia euclídea (ecuación 3.3.2) y N es la cantidad total de neuronas competitivas. Luego, SOM sólo actualiza el vector de peso de dicha neurona y de su vecindad según la ecuación 3.5

$$W_{ij} = W_{ij} + \alpha * (X_i - W_{ij}) \qquad j = 1..n \tag{3.5}$$

siendo n la dimensión del espacio de entrada, i la neurona competitiva cuyo vector se desea actualizar y α un valor entre 0 y 1 que representa un factor de aprendizaje. Los vectores de pesos de las restantes neuronas competitivas permanecen sin cambios. La ecuación 3.5 tiene variantes que pueden consultarse en [28].

El concepto de vecindad es utilizado para permitir que la red se adapte adecuadamente. Esto implica que neuronas competitivas vecinas representan patrones de entrada similares. Por tal motivo, durante el proceso de entrenamiento (obtención de los valores de W) se comienza con una vecindad amplia para luego ir reduciéndola a lo largo de las iteraciones.

El algoritmo 3 describe el pseudocódigo correspondiente al proceso básico de adaptación de la red SOM.

Algoritmo 3: Pseudocódigo básico de entrenamiento de la red SOM

```
W ← valores iniciales aleatorios;
Vecindad ← fijar el tamaño de vecindad inicial;
NroIteReduccion ← fijar la cantidad de iteraciones que deben transcurrir para
reducir la vecindad;
while no se alcance el criterio de terminación do
    for cada vector de entrada do
        Ingresar el vector a la red y calcular la neurona ganadora;
        Actualizar la neurona ganadora y su vecindad;
    Reducir Vecindad si corresponde según NroIteReduccion;
```

3.7. Mapas auto-organizativos dinámicos

La gran desventaja del SOM radica en la definición estática de su arquitectura ya que la cantidad y modo de conexión de las neuronas que la forman deben ser definidos a priori, antes de comenzar con el entrenamiento, condicionando de esta manera la eficiencia y eficacia de la red.

Como forma de resolver esto se han propuesto soluciones alternativas denominadas Mapas Auto-organizativos Dinámicos los cuales permiten incorporar y/o eliminar neuronas durante el entrenamiento. En estos casos, el crecimiento de la estructura se realiza en función de la cantidad de veces que gana cada neurona. Aquellas que lo hacen un mayor número de veces son reforzadas con nuevas vecinas con el objetivo de distribuir de manera más adecuada los vectores. Un mapa auto-organizativo dinámico (o creciente) es una versión extendida del clásico SOM de Kohonen con la capacidad de expandir su estructura de manera controlable.

Para trabajar con este tipo de arquitecturas las neuronas de la red deben tener posibilidad de incorporar o eliminar vecinos dinámicamente. Esto genera la necesidad de identificar el lugar donde deben insertarse nuevos elementos en la arquitectura. Este aspecto, sumado al hecho de que la sobrecarga de las neuronas es medida en forma aislada puede llevar a la pérdida de preservación de la topología. Esto se manifiesta en la ubicación de representantes de agrupaciones similares (neuronas competitivas) en posiciones muy distantes dentro de la red neuronal.

En general, la mayoría de las estrategias existentes para la definición de mapas autoorganizativos dinámicos, poseen las características principales del SOM definido por Kohonen y utilizan una medida de error dependiente del problema para determinar cuándo deben insertarse nuevas neuronas [29].

Para ello, en cada paso de adaptación la información del error local es acumulada en la neurona ganadora, evitando que un mismo elemento de la red acumule la representación de la mayoría de los patrones de entrada y siendo utilizada para determinar dónde deben insertarse nuevas unidades en la red. Cuando se realiza una inserción, la información del error es redistribuida localmente evitando nuevas inserciones en el mismo lugar.

3.7.1. Métodos existentes

Se han propuesto una cantidad importante de métodos basados en la filosofía de los mapas auto-organizativos dinámicos. A continuación se describirán brevemente algunos de los métodos existentes:

Growing cell structure (GCS)

En 1993, Brend Fritzke desarrolló una arquitectura de red neuronal auto-organizativa dinámica que, a diferencia de la mayoría de los métodos existentes, posee dos variantes de aprendizaje: no-supervisado y supervisado. Esta última es una combinación de la variante de aprendizaje no-supervisado con una función base radial [30].

Este modelo utiliza la figura de hipertetraedro de una dimensionalidad k definida a priori. Un hipertetraedro de dimensión k es el poliedro más simple que puede formarse con $k+1$ vértices. Ejemplo de hipertetraedros para $k \in 1, 2, 3$ son líneas, triángulos y tetraedros.

En GCS, la arquitectura comienza con un único hipertetraedro. Al igual que el SOM propuesto por Kohonen, para cada vector de entrada se elige la neurona más cercana. La neurona ganadora y sus vecinas forman parte del aprendizaje. La diferencia con SOM es que solo las vecinas directas forman parte del aprendizaje. Además, el peso de adaptación es constante en todo el entrenamiento.

Luego de un número λ de pasos de adaptación, se determina la unidad q que posea la máxima frecuencia de señal relativa y se inserta una nueva neurona dividiendo el arco que separa a q de su vecino directo más lejano. Luego se agregan las conexiones necesarias para preservar la forma de la arquitectura de manera que siempre esté integrada por hipertetraedros de dimensión k. Este proceso de reconexión es muy sencillo. Si la nueva neurona fue insertada entre q y f, sus vecinos directos serán q, f y todos los vecinos comunes de q y f.

Dado que GCS posee una dimensionalidad definida a priori puede emplearse como herramienta de visualización utilizando k con valor 2 o 3.

Una descripción detallada de esta arquitectura y su método de entrenamiento se pueden encontrar en [31].

Growing neural gas (GNG)

El método GNG no impone ninguna restricción sobre la forma de conexión entre las neuronas competitivas. Su funcionamiento combina el mecanismo de crecimiento de GCS y la generación de la topología a través de CHL (Competitive Hebbian Learning), propuesto por Martinetz [32] [33].

Este método de aprendizaje no cambia los valores de los vectores de referencia; sólo genera conexiones entre neuronas vecinas de manera de construir un grafo que preserva óptimamente la topología. En particular, las conexiones finales del grafo se corresponden con la triangulación de Delaunay del conjunto de vectores de referencia dados.

Para combinar ambas tareas, GNG propone aplicar la generación de conexiones según CHL incorporando el concepto de edad a cada conexión. En cada paso de adaptación, además de corregir el vector prototipo asociado a la neurona ganadora y de incrementar su valor de error como se describió previamente, se incrementan las edades de los arcos que la conectan con sus vecinos directos. Cuando la edad de una conexión alcanza un determinado umbral la conexión se elimina. Las neuronas que pierden todas sus conexiones también son eliminadas.

El proceso de crecimiento de la estructura está regulado por un parámetro λ que representa el umbral de pasos de adaptación que debe esperarse para insertar una nueva neurona a la estructura. Alcanzado este umbral se determina la neurona q que posea el mayor error acumulado. Luego, se determina entre los vecinos de q la neurona f con mayor valor de error acumulado. Se crea entonces una nueva neurona r entre q y f. Se conecta r con q y f y se elimina la conexión directa entre q y f. Finalmente se reducen los errores acumulados

de q y f en una fracción indicada a priori y se calcula el error de r como el promedio de los errores de q y f.

El proceso completo se repite hasta que se haya alcanzado el número máximo de neuronas permitidas dentro de la red o hasta que cumpla algún criterio de finalización establecido previamente.

Se recomienda la lectura de [34] para contar con un conocimiento detallado de cómo funciona este método.

Growing selforganizing map

Este método organiza las neuronas en una grilla bidimensional permitiendo que cada elemento de la red pueda tener a lo sumo cuatro vecina directas [35].

La estructura inicial está formada por neuronas conectadas de manera que cada una de ellas sólo posea dos vecinas. Antes de comenzar con el entrenamiento, se determina el umbral GT que debe alcanzar el error acumulado de una neurona para considerar la necesidad de agregarle un vecino inmediato.

El proceso de entrenamiento consta de dos partes: la primera genera todas las neuronas de la estructura y la segunda acomoda los vectores de pesos correspondientes a cada unidad de la red.

La etapa de generación respeta, en líneas generales, el proceso descrito inicialmente ya que en cada paso de adaptación se recalculan los vectores de pesos correspondientes y se incrementa el error de la neurona ganadora. Cuando dicho error supera el umbral indicado por GT, se agrega una nueva vecina de la neurona ganadora. Esto ocurre siempre que tenga menos de cuatro vecinos; sino es así, sólo se distribuyen los pesos. Esta etapa finaliza cuando se logra un crecimiento mínimo o cuando ya no se producen nuevas neuronas.

Finalmente, una vez obtenida la arquitectura de la red, se realiza una etapa de suavizado reduciendo la velocidad de aprendizaje y el tamaño de la vecindad.

Un concepto importante que Alahakoon presenta en su método es el uso de un parámetro llamado factor de dispersión (spread factor) que permite indicarle a la red en que medida debe dispersarse o crecer. De esa manera, la red puede crecer poco para tener una visión amplia de cuales son los principales clusters que se forman con los datos de entrada, o dispersarse mucho más obteniendo así un mayor detalle de cada agrupamiento.

La organización bidimensional de las neuronas permite una fácil visualización de los agrupamientos obtenidos luego del entrenamiento independientemente de la dimensionalidad de los patrones de entrada.

3.8. Conclusiones

Este capítulo dedicado a las técnicas de clustering resume brevemente el concepto general de agrupamiento y su capacidad descriptiva del conjunto de datos.

El modelo obtenido en este caso no se resume en un conjunto de reglas sino en una manera de relacionar la información a través de una medida de similitud dada de antemano.

En el capítulo anterior, donde se presentaron alternativas para la obtención de reglas de clasificación se trabajó con información etiquetada. Por el contrario, en este capítulo se mostraron técnicas alternativas para construir modelos sin contar con esta información.

En el contexto de esta tesina, se utilizó una red SOM para determinar, a través de los centroides de cada neurona, las características más representativas de los datos de entrada. Luego, utilizando medidas de tendencia central sobre cada uno de los componentes de los centroides fue posible estimar la importancia que cada uno de los atributos posee a la hora de definir el antecedente de una regla. Esta información resulta de suma utilidad para la técnica de optimización encargada de conducir la búsqueda hacia el conjunto de reglas adecuado. Gracias a la red SOM, este proceso comienza en zonas prometedoras, cercanas al óptimo, mejorando la respuesta obtenida y reduciendo el tiempo de cálculo.

La arquitectura utilizada en esta tesina para la red SOM surguió luego de diferentes pruebas realizadas sobre los conjuntos de datos disponibles. En el futuro, esto podría automatizarse utilizando una red competitiva dinámica la cual determinaría la cantidad de neuronas óptima en cada situación. Por este motivo se incluyó en la parte final del capítulo algunas soluciones existentes en la literatura sobre esta temática aunque no se han realizado aún las implementaciones y pruebas correspondientes.

Capítulo 4

Técnicas de Optimización

4.1. Introducción

En general, cuando se busca resolver un problema, se desea hacerlo de la mejor manera posible. En algunas ocasiones puede alcanzarse la mejor solución existente y en otras, sólo una aproximación de ella.

En Informática se han desarrollado diversos métodos, denominados técnicas de optimización, que han demostrado ser capaces de encontrar soluciones cercanas al óptimo.

Este capítulo introduce este tema haciendo hincapié en la optimización mediante cúmulos de partículas ya que es de gran importancia para el método propuesto en esta tesina. Antes de llegar a ella, se hará una clasificación de las técnicas existentes para dar un contexto donde pueda apreciarse la diversidad de opciones disponibles en esta temática.

Es importante considerar que las técnicas aquí expuestas pueden aplicarse a una amplia variedad de situaciones. Para ello, sólo es necesario comprender los aspectos principales del problema a resolver y encontrar la manera de describirlos en términos de la técnicas de optimización a aplicar. Cada técnica se diferencia en la manera de encontrar el óptimo. Si el planteo del problema en los términos de la técnica es el adecuado y la técnica seleccionada es la correcta, la obtención de una solución aproximadamente optima es posible.

4.2. Clasificación

Primeramente, las técnicas de optimización pueden clasificarse en exactas (o enumerativas, exhaustivas, etc.) y técnicas aproximadas. Las técnicas exactas garantizan encontrar la solución óptima para cualquier instancia de cualquier problema en un tiempo acotado. El inconveniente de estos métodos es que el tiempo necesario para llevarlos a cabo, aunque acotado, crece exponencialmente con el tamaño del problema, ya que la mayoría de éstos son NP-Completos. Esto provoca en muchos casos que el tiempo necesario para la resolución del problema sea inaceptable. Debido a estas limitaciones surgen los algoritmos aproximados o heurísticas. Estos métodos sacrifican la garantía de encontrar el

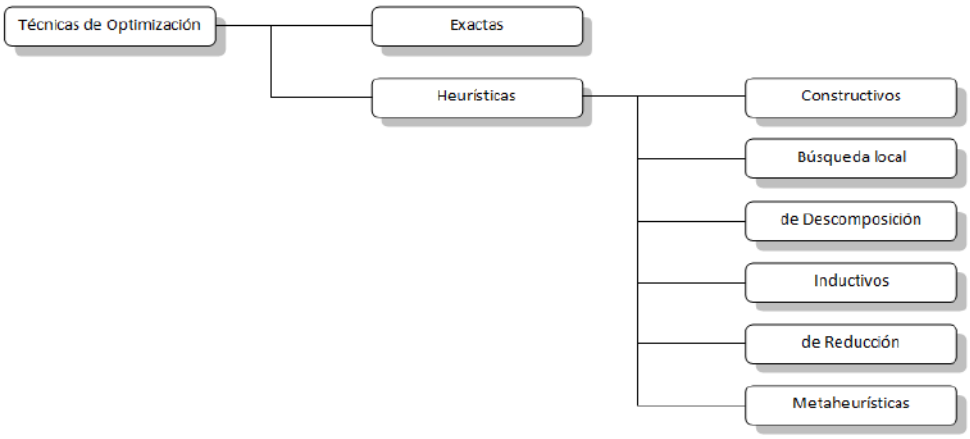


Figura 4.1: Clasificación de las técnicas de optimización

óptimo a cambio de encontrar una buena solución en un tiempo razonable. En algunos casos, ni siquiera pueden determinar qué tan cerca del óptimo se encuentra una solución factible en particular.

Dentro de los algoritmos no exactos existen muchos métodos heurísticos de naturaleza muy diferente, por lo que es complicado dar una clasificación completa de los mismos. Además, muchos de ellos han sido diseñados para un problema específico sin posibilidad de generalización o aplicación a otros problemas similares. De acuerdo a lo dicho anteriormente, como se observa en la figura 4.1, el siguiente esquema realiza una categorización general para ubicar a los métodos heurísticos más conocidos según la forma de buscar y construir la solución [36]:

- *Métodos constructivos o también llamados voraces*
  Los heurísticos constructivos suelen ser los métodos más rápidos. Consisten en construir literalmente paso a paso una solución del problema. Generan una solución partiendo de una vacía a la que se les va añadiendo componentes hasta tener una solución completa, que es el resultado del algoritmo. Usualmente son métodos deterministas y suelen estar basados en la mejor elección en cada iteración.

- *Métodos de búsqueda local o también llamados de seguimiento del gradiente*
  Parten de una solución ya completa junto con el uso del concepto de vecindario, recorren parte del espacio de búsqueda hasta encontrar un óptimo local. En esa definición han surgido diferentes conceptos, como el de vecindario y óptimo local. El vecindario de una solución s, denominado $N(s)$, es el conjunto de soluciones que se pueden construir a partir de s aplicando un operador específico de modificación (generalmente denominado movimiento). Un óptimo local es una solución mejor o igual que cualquier otra solución de su vecindario. Estos métodos, partiendo de una solución inicial, examinan su vecindario y eligen el mejor vecino continuando el proceso hasta que encuentran un óptimo local. El procedimiento realiza en cada paso un movimiento de una solución a otra con mejor valor. En muchos casos, la

exploración completa del vecindario es inabordable y se siguen diversas estrategias, dando lugar a diferentes variaciones del esquema genérico. Según el operador de movimiento elegido, el vecindario cambia y el modo de explorar el espacio de búsqueda también, pudiendo simplificarse o complicarse el proceso de búsqueda. El método finaliza cuando, para una solución, no existe ninguna solución accesible que la mejore.

- *Métodos de descomposición*
  El problema original se descompone en subproblemas más sencillos de resolver, siendo la salida de uno la entrada del siguiente.

- *Métodos inductivos*
  La idea de estos métodos es generalizar de versiones pequeñas o más sencillas al caso completo. Propiedades o técnicas identificadas en estos casos más fáciles de analizar pueden ser aplicadas al problema completo.

- *Métodos de reducción*
  Consiste en identificar propiedades que se cumplen mayoritariamente por las buenas soluciones e introducirlas como restricciones del problema. El objeto es restringir el espacio de soluciones simplificando el problema. El riesgo obvio es dejar fuera las soluciones óptimas del problema original.

- *Metaheurísticas*
  En los años setenta surgió una nueva clase de algoritmos no exactos, cuya idea básica era combinar diferentes métodos heurísticos a un nivel más alto para conseguir una exploración del espacio de búsqueda de forma eficiente y efectiva. Estas técnicas se han denominado metaheurísticas, término utilizado por primera vez por Glover en [37]. Antes de que este término fuese aceptado completamente por la comunidad científica estos métodos eran denominados como heurísticos modernos [38].

El método propuesto en esta tesina hace uso de una metaheurística y por tal motivo, se analizará esta categoría con un poco más de detalle.

Podemos describir las propiedades principales que caracterizan las metaheurísticas como [39] [40] [41]:

- Estrategias o plantillas generales que guían el proceso de búsqueda.
- Exploración del espacio de búsqueda eficiente con el objetivo de encontrar soluciones (casi) óptimas.
- Son algoritmos no exactos y generalmente son no deterministas.
- Pueden incorporar mecanismos para evitar las áreas del espacio de búsqueda no óptimas.
- El esquema básico de cualquier metaheurística es general y no depende del problema a resolver.

- Hacen uso de conocimiento del problema que se trata de resolver en forma de heurísticos específicos que son controlados de manera estructurada por una estrategia de más alto nivel.
- Utilizan funciones de aptitud para cuantificar el grado de adecuación de una determinada solución.

Haciendo un resumen de los puntos anteriores, se puede decir que una metaheurística es una estrategia de alto nivel que explora el espacio de búsqueda por medio de diferentes métodos. En otras palabras, una metaheurística es una plantilla general no determinista que debe ser rellenada con datos específicos del problema adaptados para la misma: representación de las soluciones, operadores para manipularlas, función de adecuación para el proceso de optimización, etc. Permite abordar problemas con espacios de búsqueda de gran tamaño, donde el número de posibles soluciones es extremadamente grande. Por lo tanto es de especial interés el correcto equilibrio que haya entre diversificación e intensificación. El término diversificación se refiere a la exploración del espacio de búsqueda, mientras que intensificación se refiere a la explotación de algún área concreta de ese espacio. El equilibrio entre estos dos aspectos contrapuestos es de gran importancia, ya que por un lado deben identificarse rápidamente la regiones prometedoras del espacio de búsqueda global y por el otro lado no se debe malgastar tiempo en las regiones que ya han sido exploradas o que no contienen soluciones de alta calidad.

Existen diversas formas de clasificar y describir las técnicas metaheurísticas [42]. Dependiendo de las características que se seleccionen se pueden obtener diferentes taxonomías: basadas en la naturaleza o no basadas en la naturaleza, basadas en memoria o sin memoria, con función objetivo estática o dinámica, etc.

En lo que continúa del capítulo se ha elegido clasificarlas de acuerdo a si en cada paso manipulan un único punto del espacio de búsqueda o trabajan sobre un conjunto (población) de ellos. Es decir que, se detallará la agrupación que divide a las metaheuísticas en aquellas basadas en la trayectoria y aquellas basadas en la población, como se muestra en la Figura 4.2. En dicha figura se pueden observar las principales metaheurísticas que se decribirán brevemente en las siguientes subsecciones.

4.2.1. Metaheurísticas Basadas en Trayectoria

La principal característica de estos métodos es que parten de un punto y mediante la exploración del vecindario van actualizando la solución actual, formando una trayectoria. La mayoría de estos algoritmos surgen como extensiones de los métodos de búsqueda local simples a los que se les añade alguna característica para escapar de los mínimos locales. Esto implica la necesidad de una condición de parada más compleja que la de encontrar un mínimo local. Normalmente se termina la búsqueda cuando se alcanza un número máximo predefinido de iteraciones, se encuentra una solución con una calidad aceptable o se detecta un estancamiento del proceso.

- El Enfriamiento Simulado o Simulated Annealing (SA) es una de las más antiguas

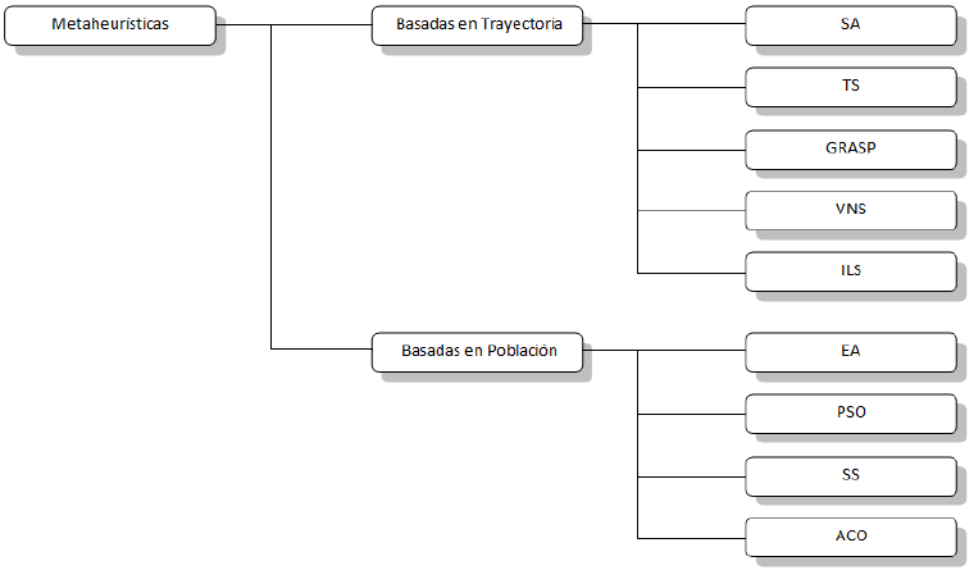


Figura 4.2: Clasificación de las metaheurísticas

entre las metaheurísticas y seguramente es el primer algoritmo con una estrategia explícita para escapar de los óptimos locales. El SA fue inicialmente presentado en [43]. La idea del SA es simular el proceso de recocido del metal y del cristal. Para evitar quedar atrapado en un óptimo local, el algoritmo permite elegir una solución peor que la solución actual. En cada iteración se elige, a partir de la solución actual s, una solución s' del vecindario $N(s)$. Si s' es mejor que s (es decir, tiene un mejor valor en la función de fitness), se sustituye s por s' como solución actual. Si la solución s' es peor, entonces es aceptada con una determinada probabilidad que depende de la temperatura actual T y de la variación en la función de fitness, $f(s') - f(s)$ (caso de minimización). Esta probabilidad generalmente se calcula siguiendo la distribución de Boltzmann:

$$p(s' \mid T, s) = e^{\frac{f(s')-f(s)}{T}}$$

- La Búsqueda Tabú o Tabu Search (TS) es una de las metaheurísticas que se han aplicado con más éxito a la hora de resolver problemas de optimización combinatoria. Los fundamentos de este método fueron introducidos en [37]. La idea básica de la búsqueda tabú es el uso explícito de un historial de la búsqueda (una memoria de corto plazo), tanto para escapar de los óptimos locales como para implementar su estrategia de exploración y evitar buscar varias veces en la misma región. Esta memoria de corto plazo es implementada en esta técnica como una lista tabú, donde se mantienen las soluciones visitadas más recientemente para excluirlas de los próximos movimientos. En cada iteración se elige la mejor solución entre las permitidas y la solución es añadida a la lista tabú.

- El Procedimiento de Búsqueda Miope Aleatorizado y Adaptativo o The Greedy

Randomized Adaptive Search Procedure (GRASP) [44] es una metaheurística simple que combina heurísticos constructivos con búsqueda local. GRASP es un procedimiento iterativo compuesto de dos fases: primero una construcción de una solución y después un proceso de mejora. La solución mejorada es el resultado del proceso de búsqueda.

- La Búsqueda en Vecindario Variable o Variable Neighborhood Search (VNS) es una metaheurística propuesta en [45], que aplica explícitamente una estrategia para cambiar entre diferentes estructuras de vecindario de entre un conjunto de ellas definidas al inicio del algoritmo. Este algoritmo es muy general y con muchos grados de libertad a la hora de diseñar variaciones e instanciaciones particulares.

- La Búsqueda Local Iterada o Iterated Local Search (ILS) [46] es una metaheurística basada en un concepto simple pero muy efectivo. En cada iteración, la solución actual es perturbada y a esta nueva solución se le aplica un método de búsqueda local para mejorarla. Este nuevo óptimo local obtenido por el método de mejora puede ser aceptado como nueva solución actual si pasa un test de aceptación.

4.2.2. Metaheurísticas Basadas en Población

Los métodos basados en población se caracterizan por trabajar con un conjunto de soluciones (población) en cada iteración, a diferencia de los métodos observados anteriormente que únicamente utilizan un punto del espacio de búsqueda por iteración. El resultado final proporcionado por este tipo de algoritmos depende fuertemente de la forma en que manipula la población.

- Los Algoritmos Evolutivos o Evolutionary Algorithms (EA) [47] están inspirados en la capacidad de la naturaleza para evolucionar seres adaptándolos a los cambios de su entorno. Esta familia de técnicas siguen un proceso iterativo y estocástico que opera sobre una población de individuos. Cada individuo representa una solución potencial al problema que se está resolviendo. Inicialmente, la población es generada aleatoriamente (quizás con ayuda de un heurístico de construcción). Cada individuo en la población tiene asignado, por medio de una función de aptitud (fitness), una medida de su bondad con respecto al problema bajo consideración. Este valor es la información cuantitativa que el algoritmo usa para guiar su búsqueda. En los métodos que siguen el esquema de los algoritmos evolutivos, la modificación de la población se lleva a cabo mediante tres operadores: selección, recombinación y mutación. Estos algoritmos establecen un equilibrio entre la explotación de buenas soluciones (fase de selección) y la exploración de nuevas zonas del espacio de búsqueda (fase de reproducción), basados sobre el hecho de que la política de reemplazo permite la aceptación de nuevas soluciones que no mejoran necesariamente la existentes. En la literatura se han propuesto diferentes algoritmos basados en este esquema general. Básicamente, estas propuestas se pueden clasificar en tres categorías que fueron desarrolladas de forma independiente. Estas categorías son la Programación

Evolutiva o Evolutionary Programming (EP) desarrollada por Fogel [48], las Estrategias Evolutivas o Evolution Strategies (ES) propuestas por Rechenberg en [49], y los Algoritmos Genéticos o Genetic Algorithms (GA) introducidos por Holland en [50].

- La Búsqueda Dispersa o Scatter Search (SS) [41] es una metaheurística cuyos principios fueron presentados en [51] y que actualmente está recibiendo una gran atención por parte de la comunidad científica. El algoritmo se basa en mantener un conjunto relativamente pequeño de soluciones tentativas (llamado conjunto de referencia) que se caracteriza tanto por contener buenas soluciones como soluciones diversas. Este conjunto se divide en subconjuntos de soluciones a las cuales se les aplica una operación de recombinación y mejora. Para realizar la mejora o refinamiento de soluciones se suelen utilizar mecanismos de búsqueda local.

- Los sistemas basados en Colonias de Hormigas o Ant Colony Optimization (ACO) [52] son unas metaheurísticas inspiradas en el comportamiento de las hormigas reales cuando realizan la búsqueda de comida. Este comportamiento es el siguiente: inicialmente, las hormigas exploran el área cercana a su nido de forma aleatoria. Tan pronto como una hormiga encuentra la comida, la lleva al nido. Mientras que realiza este camino, la hormiga va depositando una sustancia química denominada feromona. Esta sustancia ayudará al resto de las hormigas a encontrar la comida. Esta comunicación indirecta entre las hormigas mediante el rastro de feromona las capacita para encontrar el camino más corto entre el nido y la comida. Esta funcionalidad es la que intenta simular este método para resolver problemas de optimización. En esta técnica, el rastro de feromona es simulado mediante un modelo probabilístico.

- Los Algoritmos Basados en Cúmulos de Partículas o Particle Swarm Optimization (PSO) [53] son técnicas metaheurísticas inspiradas en el comportamiento social del vuelo de las bandadas de aves o el movimiento de los bancos de peces. Se fundamenta en los factores que influyen en la toma de decisión de un agente que forma parte de un conjunto de agentes similares. La toma de decisión por parte de cada agente se realiza conforme a una componente social y una componente individual, mediante las que se determina el movimiento (dirección) de este agente para alcanzar una nueva posición en el espacio de soluciones. Simulando este modelo de comportamiento se obtiene un método para resolver problemas de optimización.

4.3. PSO - Particle Swarm Optimization

La optimización mediante el cúmulo de partículas o PSO (Particle Swarm Optimization) es una metaheurística poblacional propuesta por Kennedy y Eberhart [53]. Está basada en la simulación de modelos sociales simples e inspirada en el comportamiento social del vuelo de las bandadas de aves o el movimiento de los bancos de peces (Figura 4.3).

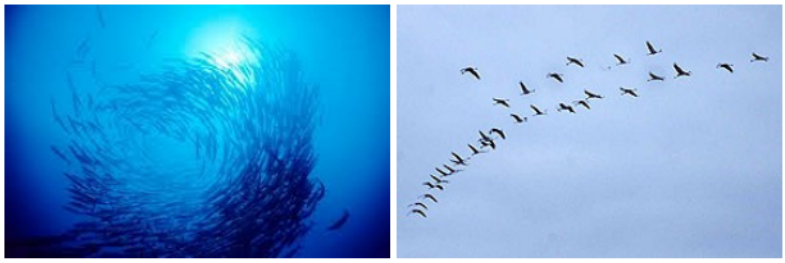


Figura 4.3: Ejemplos de *Inteligencia de Cúmulo* en la naturaleza

PSO utiliza lo que se conoce como *Inteligencia de Cúmulo* y plantea una "metáfora social" que se puede resumir de la siguiente forma: los individuos que conviven en una sociedad tienen una opinión que es parte de un "conjunto de creencias" (el espacio de búsqueda) compartido por todos los posibles individuos.

El libro de Kennedy y Eberhart [54] describe muchos aspectos filosóficos de PSO y la inteligencia de cúmulo. Poli hizo una cobertura exhaustiva de las aplicaciones de PSO en [55] y [56].

Cada individuo puede modificar su propia opinión basándose en tres factores:

- Su conocimiento sobre el entorno (su valor de aptitud o fitness).
- Su conocimiento histórico o experiencias anteriores (su memoria o conocimiento cognitivo).
- El conocimiento histórico o experiencias anteriores de los individuos situados en su vecindario (su conocimiento social).

es decir que intervienen aspectos sociales, culturales y de imitación.

Los seres humanos también utilizan la *inteligencia de cúmulo* en la resolución de problemas de optimización. Por ejemplo, en un incendio, ante la imposibilidad de encontrar una vía de escape rápida, los humanos tienden a seguir el comportamiento de la mayoría; por tal motivo, si alguien advierte que todos corren en determinada dirección y no dispone de conocimiento cierto que le permite salvarse, lo natural es que adopte el comportamiento de la mayoría sin saber efectivamente si esa dirección es la correcta. Nótese que el conocimiento actual de cada individuo se ve afectado por el entorno.

Según se define en [53], PSO utiliza una población de tamaño fijo. Cada elemento de la población, denominado partícula, es una solución del problema y sus movimientos se encuentran acotados al espacio de búsqueda. Este espacio se encuentra definido de antemano y no se permite que las partículas se desplacen fuera de él.

En PSO, cada individuo o partícula está siempre en continuo movimiento explorando el espacio de búsqueda y nunca muere.

Cada partícula está compuesta por tres vectores y dos valores de fitness:

- Vector $X_i = (x_{i1}, x_{i2}, \ldots, x_{in})$ almacena la posición actual de la partícula en el espacio de búsqueda.

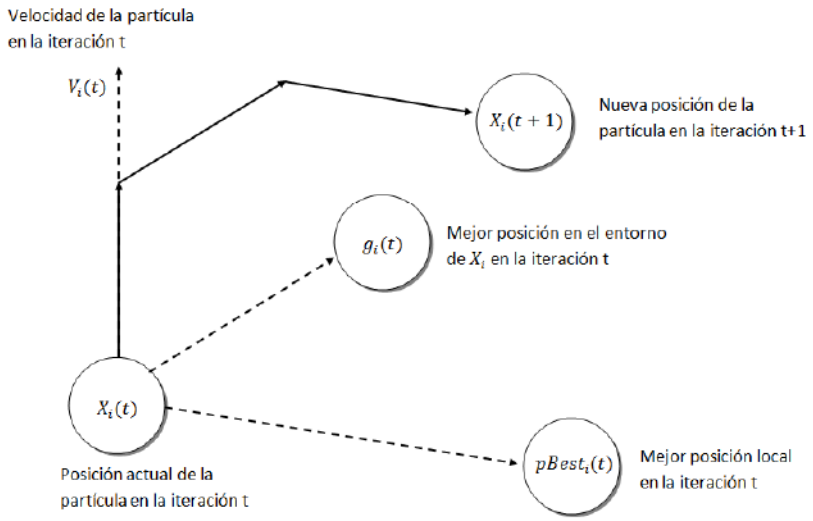


Figura 4.4: Movimiento de una partícula en el espacio de soluciones

- Vector $pBest_i = (p_{i1}, p_{i2}, \ldots, p_{in})$ almacena la mejor solución encontrada por la partícula hasta el momento.
- Vector velocidad $V_i = (v_{i1}, v_{i2}, \ldots, v_{in})$ almacena el gradiente (dirección) según el cual se moverá la partícula.
- El valor fitness $fitness\_x_i$ almacena el valor de aptitud de la solución actual (vector X_i).
- El valor fitness $fitness\_pBest_i$ almacena el valor de aptitud de la mejor solución local encontrada hasta el momento (vector $pBest_i$)

En la figura 4.4 se detalla una representación gráfica del movimiento de una partícula en el espacio de soluciones. Las flechas de línea punteada representan la dirección de los vectores de velocidad actual. La flecha de línea completa representa la dirección que toma la partícula para moverse desde la posición $X_i(t)$ hasta la posición $X_i(t+1)$. El cambio de dirección de esta flecha depende de la influencia de las demás direcciones que intervienen en el movimiento.

La población se inicializa generando las posiciones y las velocidades iniciales de las particulas aleatoriamente. Una vez que la población está inicializada, los individuos comienzan a moverse por el espacio de búsqueda por medio del proceso iterativo.

Con la nueva posición del individuo, se calcula y actualiza su fitness ($fitness\_x_i$). Además, si el nuevo fitness del individuo es el mejor encontrado hasta el momento, se actualizan los valores de mejor posición $pBest_i$ y fitness $fitness\_pBest_i$.

Como se explicó anteriormente, el vector velocidad es modificado tomando en cuenta su experiencia y su entorno.

La expresión es:

$$V_i(t+1) = w.V_i(t) + \varphi_1.rand_1.(pBest_i - X_i(t)) + \varphi_2.rand_2.(g_i - X_i(t)) \qquad (4.1)$$

donde w representa el factor inercia [57], φ_1 y φ_2 las constantes de aceleración, $rand_1$ y $rand_2$ son valores aleatorios pertenecientes al intervalo (0,1) y g_i representa la posición de la particula con el mejor $pBest$ en el entorno de X_i ($lBest$ o $localbest$) o del todo el cúmulo ($gBest$ o $globalbest$). Los valores de w, φ_1 y φ_2 son importantes para asegurar que converja el algortimo. Para más detalles sobre la elección de estos valores consultar [58] y [59].

Finalmente, se actualiza la posición de la partícula de la siguiente forma:

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{4.2}$$

El algoritmo 4 describe el pseudocódigo del algoritmo de búsqueda de PSO

Algoritmo 4: Algoritmo PSO Básico

```
Pop = CrearPoblacion(N);
while no se alcance la condición de terminación do
    for i = 1 to size(Pop) do
        Evaluar Partícula X_i del cúmulo Pop;
        if fitness(X_i) es mejor que fitness(pBest_i) then
            pBest_i ← X_i;
            fitness(pBest_i) ← fitness(X_i);
    for i = 1 to size(Pop) do
        Elegir g_i de acuerdo al criterio de vecindario usado;
        V_i ← w.V_i + (φ1.rand1.(pBest_i − X_i) + φ2.rand2.(g_i − X_i));
        X_i ← X_i + V_i;
Result: la mejor solución encontrada
```

Es importante notar que PSO es una metaheurística que hace muy pocas suposiciones con respecto al problema que se busca resolver y es capaz de buscar en espacios de soluciones muy grandes.

Como contrapartida, debe mencionarse que, metaheurísticas como PSO no garantizan que la solución óptima sea alcanzada. PSO no utiliza la información de ningún gradiente para realizar la búsqueda, esto implica que la función de aptitud a utilizar no necesita ser diferenciable como es requerido por los métodos de optimización clásicos, como descenso de gradiente y los métodos cuasi-Newton.

Esto último permite que PSO sea utilizado en problemas de optimización cuya función de aptitud sea ruidosa o se encuentre parcialmente definida.

Existen diferentes versiones que fueron desarrolladas a partir de la idea original, la mayoría de ellas relacionadas con la variación de diversos parámetros del algoritmo o a la combinación de varias topologías, tamaños y número de poblaciones [60][61][62][63].

También se han propuesto cambios en la velocidad de las partículas con el fin de lograr diversidad y evitar el estancamiento en el proceso de búsqueda [64] [65] [66].

4.3.1. Tipos de Algoritmos basados en PSO

El algoritmo PSO original puede alterarse modificando su configuración original y, de esta manera, dar origen a nuevas aproximaciones. A continuación se detallan variaciones clásicas basadas en alteraciones sobre el tamaño de la vecindad y otras basadas en alteraciones de los pesos del conocimiento cognitivo y social.

Dependiendo de la influencia de los pesos cognitivo y social (valores φ_1 y φ_2 respectivamente) sobre la dirección de la velocidad que toma una partícula en el movimiento (ecuación 4.1), Kennedy [67] se pueden distinguir cuatro tipo de algoritmos:

- Modelo Completo: $\varphi 1 > 0$ y $\varphi 2 > 0$. Tanto el componente cognitivo como el social intervienen en el movimiento.
- Modelo sólo Cognitivo: $\varphi 1 > 0$ y $\varphi 2 = 0$. Únicamente el componente cognitivo interviene en el movimiento.
- Modelo sólo Social: $\varphi 1 = 0$ y $\varphi 2 > 0$. Únicamente el componente social interviene en el movimiento.
- Modelo sólo Social exclusivo: $\varphi_1 = 0$, $\varphi_2 > 0$ y $g_i \neq x_i$. La posición de la partícula en sí no puede ser la mejor de su entorno.

Los parámetros $\varphi 1$ y $\varphi 2$ ponderan la importancia que se le otorga al conocimiento adquirido por la partícula o por el mejor del entorno respectivamente. A mayor valor de $\varphi 1$ o $\varphi 2$ mayor presión se realiza para que la partícula sea atraída por su mejor solución encontrada o por el mejor del entorno, respectivamente.

Por otro lado, desde el punto de vista del vecindario, el algoritmo puede ser clasificado en dos tipos de algoritmos: *PSO Local* y *PSO Global*.

En la variación *PSOLocal*, el conocimiento social de la partícula es evaluado sólo en el entorno inmediatamente próximo a la misma. En cambio, para la variación *PSOGlobal*, el conocimiento social para cada partícula es evaluado en la población completa.

El tamaño de la vecindad influye en la velocidad de convergencia del algoritmo así como en la diversidad de los individuos de la población. A mayor tamaño de vecindad, la convergencia del algoritmo es más rápida pero la diversidad de los individuos es menor.

4.3.2. Topologías de Cúmulos de Partículas

Un aspecto muy importante a considerar es la manera en la que una partícula interacciona con las demás partículas de su vecindario. Las topologías definen el entorno de interacción de una partícula individual con su vecindario, por lo que los entornos pueden ser de dos tipos [68]:

- Geográficos: se calcula la distancia de la partícula actual al resto y se toman las más cercanas para componer su entorno.

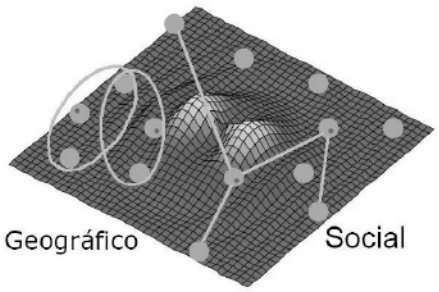


Figura 4.5: Ejemplos de entornos sociales y geográficos en un espacio de soluciones

- Sociales: se define a priori una lista de vecinas para cada partícula, independientemente de su posición en el espacio.

En la figura 4.5 se muestran gráficamente ejemplos de entornos geográficos y sociales

4.4. PSO Binario

En diversas aplicaciones prácticas cada vez es más frecuente la presencia de problemas de optimización que involucran variables que deben tomar valores discretos. Se trata de problemas de naturaleza combinatoria donde una de las formas de resolverlos es la discretización de valores continuos. Considerando que cualquier problema de optimización, discreto o continuo, puede ser expresado en notación binaria, Kennedy y Eberthart en [69], consideraron de utilidad el poder disponer de una versión binaria discreta de PSO para problemas de optimización discretos que denominaron *PSO Binario*.

En un espacio binario, una partícula se mueve en las esquinas de un hipercubo, cambiando los valores de los bits que determinan su posición. En este contexto, la velocidad de una particula puede ser vista como la cantidad de cambios ocurridos por iteración o la distancia de Hamming entre las posiciones de la partículas en el instante t y el $t + 1$. Una partícula con 0 bits cambiados, de una iteración a la otra, no se mueve mientras que otra partícula que cambia por el valor contrario todos sus bits, se desplaza a velocidad máxima.

Aún no se ha dicho en que consiste el vector velocidad. Repitiendo la notación del capítulo anterior, la partícula de PSO binario está formada por

- Vector $X_i = (x_{i1}, x_{i2}, \ldots, x_{in})$ almacena la posición actual de la partícula en el espacio de búsqueda. Es decir que se trata de un vector binario.
- Vector $pBest_i = (p_{i1}, p_{i2}, \ldots, p_{in})$ almacena la mejor solución encontrada por la partícula hasta el momento. Por lo tanto, también es binario.
- Vector velocidad $V_i = (v_{i1}, v_{i2}, \ldots, v_{in})$ es un vector de valores reales.
- El valor fitness $fitness\_x_i$ almacena el valor de aptitud de la solución actual (vector x_i).

- El valor fitness $fitness\_pBest_i$ almacena el valor de aptitud de la mejor solución local encontrada hasta el momento (vector $pBest_i$)

El valor de la velocidad para la partícula i en la dimensión d se actualiza según (4.3)

$$v_{id}(t+1) = w.v_{id}(t) + \varphi_1.rand_1.(p_{id} - x_{id}(t)) + \varphi_2.rand_2.(g_{id} - x_{id}(t)) \tag{4.3}$$

Nótese que si bien las ecuaciones (4.3) y (4.1) coinciden, ahora p_{id} y x_{id} son enteros en $\{0, 1\}$ y v_{id} es utilizada como argumento de la función sigmoide de (4.4) a fin de obtener un valor acotado en $[0, 1]$ equivalente a la probabilidad de que la posición de la partícula tome el valor 1.

$$sig(v_{id}(t)) = \frac{1}{1 + e^{-v_{id}(t)}} \tag{4.4}$$

Luego, el vector posición de la partícula se actualiza de la siguiente forma

$$x_{id}(t+1) = \begin{cases} 1 & if\ rand() < sig(v_{id}(t+1)) \\ 0 & if\ not \end{cases} \tag{4.5}$$

donde $rand()$ es un número random con distribución uniforme en [0,1] distinto para cada dimensión.

El algoritmo 5 detalla el pseudocódigo correspondiente.

En la versión continua de PSO el valor de v_{id} también se encontraba acotado a un intervalo cuyos valores eran parámetros del algoritmo.

En el caso discreto, el valor de la velocidad es acotado según(4.6)

$$|v_{id}| < Vmax \tag{4.6}$$

siendo $Vmax$ un parámetro del algoritmo. Es importante notar que $Vmax$ debe tomar valores adecuados para permitir que las partículas sigan explorando mínimamente alrededor del óptimo. Por ejemplo, si $Vmax = 6$ las probabilidades de cambio $sig(v_{id})$ estarán limitadas a 0,9975 y 0,0025. Esto permite una reducida probabilidad de cambio. Pero si $Vmax$ tomara un valor extremo, por ejemplo 50, sería muy poco probable que una partícula cambie su posición luego de alcanzar un óptimo (que tal vez sea local).

Es importante remarcar que la incorporación de la función sigmoide cambia radicalmente la manera de utilizar el vector velocidad para actualizar la posición de la partícula. En PSO continuo, el vector velocidad toma valores mayores al inicio para facilitar la exploración del espacio de soluciones y al final se reduce para permitir que la partícula se estabilice.

En PSO binario ocurre precisamente todo lo contrario. Los valores extremos, al ser mapeados por la función sigmoide, producen valores de probabilidad similares, cercanos a 0 o a 1, reduciendo la chance de cambio en los valores de la partícula. Por otro lado, valores del vector velocidad cercanos a cero incrementan la probabilidad de cambio. Es importante considerar que si la velocidad de una partícula es el vector nulo, cada uno de los dígitos binarios que determina su posición tiene probabilidad 0.5 de cambiar a 1. Es la situación más aleatoria que puede ocurrir.

Algoritmo 5: Algoritmo PSO Binario

```
Pop = CrearPoblacion(N);
while no se alcance la condición de terminación do
    for i = 1 to size(Pop) do
        Evaluar Partícula X_i del cúmulo Pop;
        if fitness(X_i) es mejor que fitness(pBest_i) then
            pBest_i ← X_i;
            fitness(pBest_i) ← fitness(X_i);
    for i = 1 to size(Pop) do
        Elegir g_i de acuerdo al criterio de vecindario usado;
        V_i ← w.V_i + (φ1.rand1.(pBest_i − X_i) + φ2.rand2.(g_i − X_i);
        for d = 1 to n do
            if rand() < sig(V_id) then
                X_id = 1
            else
                X_id = 0
```

Result: la mejor solución encontrada

Cada partícula incrementa su capacidad exploratoria a medida que el vector velocidad reduce su valor; es decir que , cuando v_{id} tiende a cero, $\lim_{t\to\infty} sig(v_{id}(t)) = 0{,}5$, permitiendo que cada dígito binario tome el valor 1 con probabilidad 0,5. Es decir que puede tomar cualquiera de los dos valores.
Por el contrario, cuando los valores del vector velocidad se incrementan, $\lim_{t\to\infty} sig(v_{id}(t)) = 1$, y por lo tanto todos los bits tienden a 1, mientras que cuando el vector velocidad decrece, tomando valores negativos, $\lim_{t\to\infty} sig(v_{id}(t)) = 0$ y todos los bits cambian a 0. Es importante remarcar que limitando los valores del vector velocidad entre -3 y 3, $sig(v_{id}) \in [0{,}0474, 0{,}9526]$, mientras que para valores superiores a 5, $sig(v_{id}) \simeq 1$ y para valores inferior a -5, $sig(v_{ij}) \simeq 0$.

4.5. PSO Binario con control de velocidad

Cuando se busca una solución óptima utilizando cúmulos de partículas, el vector velocidad es el encargado de realizar los desplazamientos dentro del espacio de búsqueda. Cuando se trabaja sobre un espacio discreto resulta de sumo interés evitar posiciones extremas que dificulten la convergencia.

Esta sección describe una variante del método PSO Binario que modifica el vector velocidad utilizando una versión modificada del algoritmo gBest PSO continuo. Es decir que, bajo esta propuesta cada partícula tendrá dos vectores velocidad, $V1$ y $V2$. El primero se actualiza según (4.7).

$$V1_i(t+1) = w.V1_i(t) + \varphi_1.rand_1.(2 * pBesti - 1) + \varphi_2.rand_2.(2 * gBest - 1) \qquad (4.7)$$

donde las variables $rand_1$, $rand_2$, φ_1 y φ_2 funcionan de la misma forma que en (4.3). Los valores p_i y g_i corresponden al i-ésimo dígito binario de los vectores $pBest_i$ y $gBest$, respectivamente.
La diferencia más importante entre (4.3) y (4.7) es que en la segunda, el desplazamiento del vector $V1$ en las direcciones correspondientes a la mejor solución encontrada por la partícula y al mejor global no dependen de la posición actual de dicha partícula.
Luego, cada elemento del vector velocidad $V1$ es controlado según (4.8)

$$v1_{ij}(t) = \begin{cases} \delta 1_j & if\ v1_{ij}(t) > \delta 1_j \\ -\delta 1_j & if\ v1_{ij}(t) \leq -\delta 1_j \\ v1_{ij}(t) & if\ not \end{cases} \tag{4.8}$$

donde

$$\delta 1_j = \frac{limit1_{upper_j} - limit1_{lower_j}}{2} \tag{4.9}$$

Es decir que, el vector velocidad $V1$ se calcula según (4.7) y se controla según (4.8). Su valor se utiliza para actualizar el valor del vector velocidad $V2$, como se indica en (4.10).

$$V2(t+1) = V2(t) + V1(t+1) \tag{4.10}$$

El vector $V2$ también se controla de manera similar al vector $V1$ cambiando $limit1_{upper_j}$ y $limit1_{lower_j}$ por $limit2_{upper_j}$ y $limit2_{lower_j}$ respectivamente. Esto dará lugar a $\delta 2_j$ que será utilizado como en (4.8) para acotar los valores de $V2$. Luego se le aplica la función sigmoide y se calcula la nueva posición de la partícula según (4.5).

El concepto de control de velocidad se basa en el método descripto en [70] donde se utiliza un control similar para evitar las oscilaciones finales de las partículas alrededor del óptimo.

Los resultados obtenidos aplicando este método para la optimización de un conjunto de funciones matemáticas de prueba han sido muy satisfactorios [71].

4.6. Conclusiones

Este capítulo ha presentado distintas técnicas informáticas que pueden ser utilizadas para hallar la solución casi óptima de un problema.

Algo que caracteriza al conjunto de métodos descriptos es que brindan soluciones aproximadamente óptimas. Esto último no es una desventaja ya que, en la mayoría de los casos, la búsqueda del óptimo requiere un tiempo de cálculo muy grande y la solución provista por las técnicas aproximadas es más que adecuada para los límites de tolerancia de error del problema.

En especial se ha hecho hincapié en la metaheurística PSO por ser la base de la técnica utilizada en el método propuesto en esta tesina. Se trata de un mecanismo de búsqueda que a partir de un conjunto de soluciones aleatorias, las mejora iterativamente utilizando el conocimiento propio de cada solución y el conocimiento del cúmulo.

Esta tesina combina dos de las variantes de PSO explicadas previamentes, para dar lugar a un nuevo método capaz de encontrar reglas de clasificación operando con atributos nominales y numéricos. Esto último requiere de algunas adaptaciones que serán descriptas con detalle en el siguiente capítulo.

Capítulo 5

SOM+PSO: método propuesto

Este capítulo constituye el aporte central de esta tesina y presenta un nuevo método de obtención de reglas de clasificación capaz de operar sobre atributos nominales y numéricos.

Su funcionamiento se basa en la combinación de una Red Neuronal competitiva SOM (Self-organizaing Maps) con una técnica de optimización PSO (Particle Swarm Optimization).

El énfasis está puesto en alcanzar una buena cobertura utilizando un número reducido de reglas donde cada una de ellas posea un número mínimo de conjunciones en su antecedente.

A continuación se describe la representación elegida para las reglas y las adaptaciones realizadas sobre las variantes de PSO existentes a fin de poder obtener el conjunto de reglas buscado.

5.1. Representación de Reglas

5.1.1. Enfoque seleccionado

Como se explicó en el capítulo 4, PSO es una técnica de optimización poblacional que parte de un conjunto de soluciones aleatorias y las mejora utilizando un proceso de búsqueda basado en el desempeño del grupo. Por lo tanto, la primera tarea que debe llevarse a cabo para aplicar esta técnica es construir el conjunto de soluciones aleatorias inicial, también denominado "población" inicial.

En este punto debe analizarse que información se almacenará en cada individuo de la población. Aquí existen dos opciones: cada individuo puede corresponderse con el conjunto de reglas completo (enfoque *Pittsburgh* [72]) o cada individuo podrá contener la información de una única regla. Además, si se decide utilizar individuos que representen una única regla, hay dos posibilidades según la manera en que se obtenga el resultado:

- En el enfoque *Michigan* cada individuo codifica una única regla pero la solución final será la población final o un subconjunto de la misma [73].

- En el enfoque *IRL (por sus siglas del inglés Iterative Rule Learning)* nuevamente cada individuo representa una regla pero la solución de la estrategia poblacional es el mejor individuo y la solución global está formada por los mejores individuos obtenidos en una secuencia de ejecuciones [74].

Disponer de individuos que posean en su representación el conjunto de reglas completo facilita la evaluación del resultado obtenido ya que es todo el conjunto el que se evalúa en forma simultánea. Por el contrario, si cada individuo de la población sólo representa una regla, su desempeño estará condicionado al de otros individuos (esto dependerá del enfoque seleccionado).

Sin embargo, bajo el enfoque Pittsburgh, los individuos tendrán una representación más extensa (incluso podría ser variable) lo cual implica un mayor costo en su evaluación y en la aplicación de la técnica de optimización. Por su parte, un esquema Michigan o IRL, poseerá individuos más cortos y por consiguiente más fáciles de evaluar y de ser modificados por la técnica seleccionada.

Si se decide utilizar individuos formados por una única regla, es importante considerar la presión selectiva ejercida por la técnica a aplicar ya que en la mayoría de los casos, la población tiende a la convergencia prematura y por lo tanto el mejor individuo se repite en un porcentaje importante de la población.

En esta tesina se decidió utilizar el enfoque *IRL (Iterative Rule Learning)* por considerarlo adecuado para una técnica de optimización adaptativa. El problema de la convergencia prematura a un único individuo se resolverá mediante ejecuciones sucesivas del mismo proceso. En cada paso de iteración, se obtendrá una única regla equivalente al mejor individuo de la población y los ejemplos que dicha regla cubra correctamente serán suprimidos de los datos de entrada. Con los ejemplos restantes se repetirá nuevamente todo el proceso. Esto se aplicará hasta terminar de cubrir los datos disponibles o hasta alcanzar una cobertura mínima. Nótese que este es el enfoque ´´separa y vencerás" visto en la sección 2.3.3 a través del cual se construye una lista de decisión.

Decidido el enfoque de la representación, se procederá a analizar la manera de codificar el antecedente y el consecuente de las reglas.

5.1.2. Opciones para la representación de una regla

Cada una de las reglas que se desean obtener tendrá la siguientes forma

```
SI <condición 1> Y <condición 2> Y ... Y <condición N> entonces
                        <consecuente>
```

donde:

- Las condiciones implican atributos nominales o numéricos. Para estos últimos es preciso determinar el rango de valores que dará lugar a dicha condición. Dicho rango puede obtenerse a través de un proceso de discretización inicial o bien, como se propone en esta tesina, puede ser la misma técnica de optimización la que se encargue de determinar cuales son los intervalos adecuados en los que debe ubicarse cada atributo numérico.

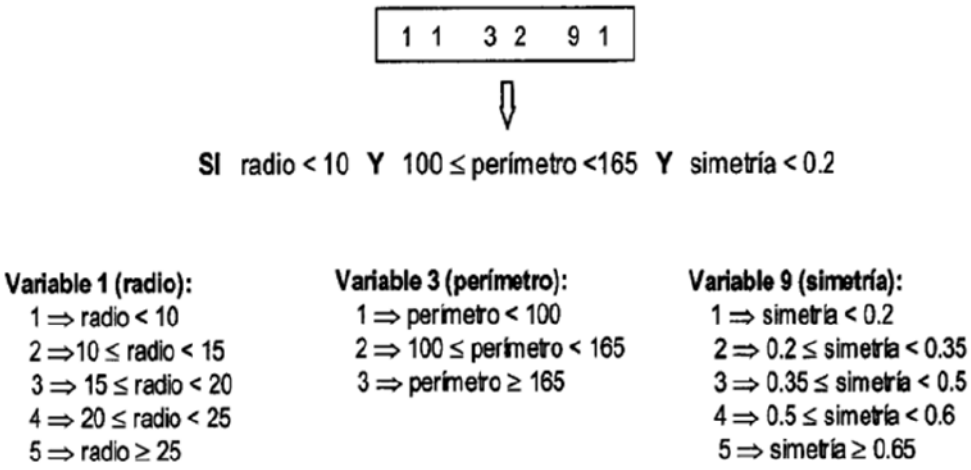


Figura 5.1: Codificación entera del antecedente de una regla de longitud variable

- La cantidad de condiciones que forman el antecedente cambia de una regla a otra.
- Cada condición podría estar formada por una disyunción de condiciones individuales. Este aspecto depende de la flexibilidad que se busque dar a la regla y de la capacidad de la técnica a utilizar para operar con esta representación. En esta tesina, como el énfasis está puesto en la obtención de reglas sencillas y con antecedentes cortos no se permiten las disyunciones entre valores de un mismo atributo.
- Dado que la regla que se desea obtener es de clasificación el consecuente es un único atributo que indica la clase resultado, el cual no puede estar en el antecedente.

Se comenzará trabajando con la codificación del antecedente de la regla y luego se discutirá si es conveniente o no considerar en dicha representación el consecuente.

Codificación del antecedente

Dado que el antecedente de una regla tiene una cantidad variable de condiciones, una primera representación a considerar es la ilustrada en la figura 5.1.

En ella, los atributos nominales se codifican con enteros donde cada uno representa un posible valor del atributo. Los numéricos, como se observa en la figura, pueden ser discretizados previamente asignándole a cada rango de valores un número entero. De esta manera se pueden tratar uniformemente tanto los atributos nominales como los numéricos.

El uso de una representación de longitud variable como el ejemplo de la figura 5.1 requiere de técnicas adecuadas para poder operar con ella.

Una forma de simplificar esto es utilizar una representación de longitud fija formada por tantos elementos como atributos existan en los datos de entrada y para cada uno de ellos se indicará un número entero positivo representando el intervalo a utilizar. El valor 0 representará que el atributo no forma parte del antecedente. Esto se ilustra en la figura

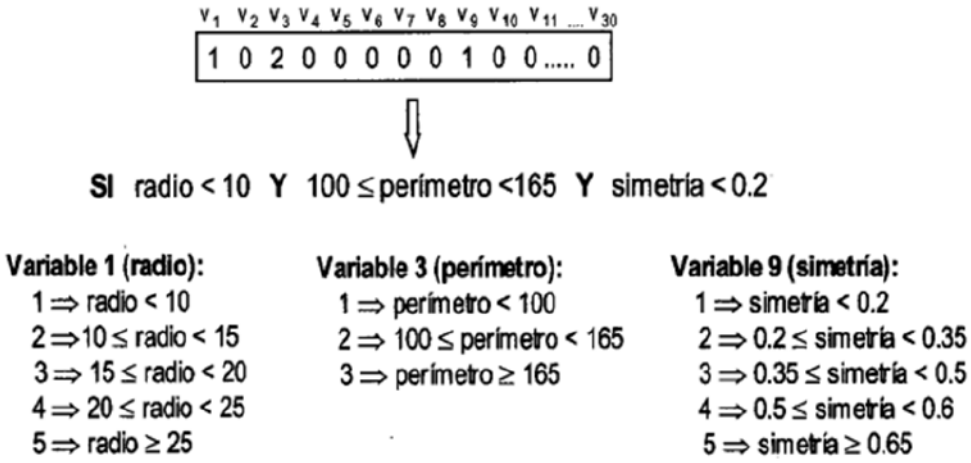


Figura 5.2: Ejemplo de codificación entera de longitud fija para el antecedente de una regla

5.2. De esta manera la codificación del antecedente con las variables alineadas facilita la aplicación de la técnica de optimización.

En la codificación anterior posee la limitación de no permitir que un mismo atributo sea consultado por más de un valor a través de una o varias disyunciones. Para ello se puede utilizar un esquema de codificación binaria, en el que cada atributo tiene un conjunto de bits asociados por cada valor posible del mismo. Si el bit tiene valor 0 implica que no participa en la regla y si tiene valor 1 sí pertenece a la regla. La figura 5.3 muestra un ejemplo de una regla con codificación binaria.

En esta tesina se ha optado por utilizar una representación binaria de longitud fija que diferencia los atributos nominales de los numéricos. En el caso de los atributos nominales utiliza tantos dígitos binarios como valores diferentes pueda tomar cada atributo nominal y en el caso de los atributos numéricos utiliza siempre dos bits a fin de identificar si debe usarse el límite inferior y/o superior de un intervalo. El objetivo de tratar a los atributos numéricos de esta forma es evitar tener que discretizarlos previamente. Es importante destacar que es la técnica de optimización la que establecerá los límites del intervalo a utilizar.

Por lo tanto, se agregará a la representación binaria una secuencia de valores reales que tendrá la misma cantidad de elementos que la binaria pero que sólo será utilizada para los atributos numéricos. Mantener la igualdad en la dimensión de ambas secuencias facilita el funcionamiento del algoritmo.

La figura 5.4 ejemplifica la representación descripta para un caso de seis atributos de los cuales tres son numéricos (v_1, v_5 y v_6) y tres son nominales (v_2, v_3 y v_4). Nótese que en el caso de V1 sólo se utiliza el límite inferior del intervalo mientras que en V5 se utilizan ambos. Igual que en los casos anteriores, el valor nulo indica que el atributo no participa en la construcción del antecedente. Esto último se cumple para ambos tipos de atributos.

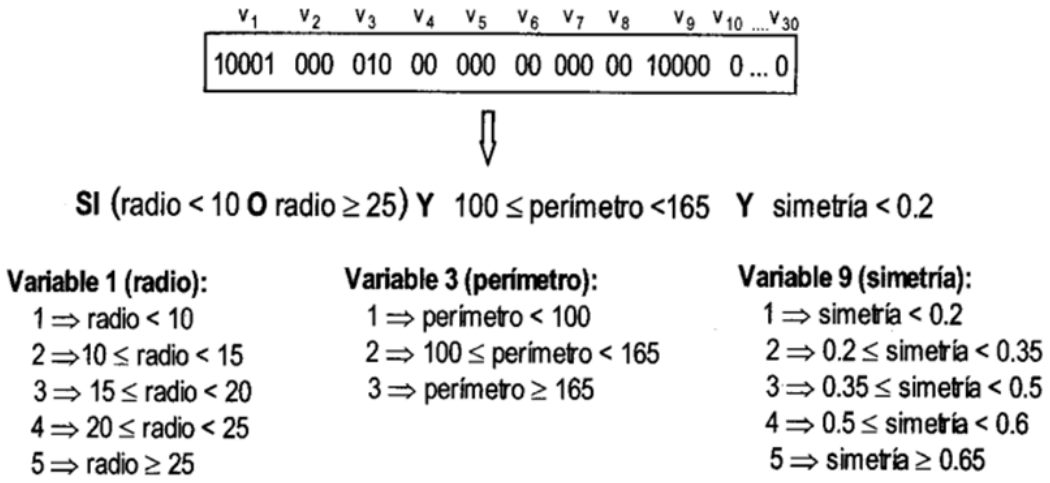


Figura 5.3: Ejemplo de codificación binaria de longitud fija para antecedentes de una regla

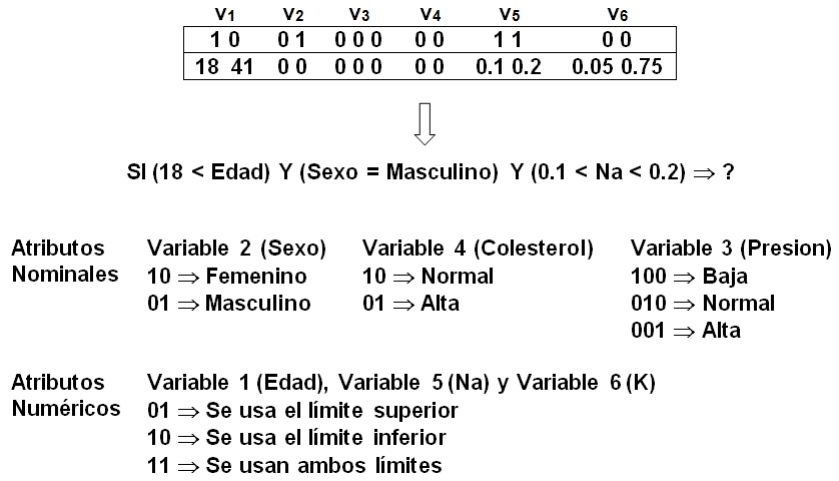


Figura 5.4: Ejemplo de codificación utilizada en esta tesina. Nótese el agregado de un vector real a fin de no forzar la discretización inicial de los atributos numéricos. El vector binario identifica los atributos que componen el antecedente y el vector real determina los intervalos a considerar (si corresponde)

Codificación del consecuente

Con respecto al consecuente, si se busca obtener reglas de clasificación se puede optar por alguno de estos tres enfoques de codificación:

- Codificarlo en el individuo [75] [76].
- Elegir la clase más adecuada para el antecedente de la regla. Por ejemplo se puede seleccionar la clase que tenga mayor cantidad de ejemplos que verifican la regla [77].
- Asociar todos los individuos de la población con la misma clase y ejecutar el algoritmo tantas veces como clases existan [78] [79].

Las dos primeras clases permiten encontrar reglas para distintas clases sin tener que ejecutar el algoritmo tantas veces como clases existen en el problema.

En esta tesina, se ha decidido utilizar una representación de longitud fija donde sólo se codificará el antecedente de la regla y dado que se aplicará el enfoque IRL, se efectuará un proceso iterativo afectando todos los individuos de la población a una clase predeterminada lo cual no requiere de la codificación del consecuente.

5.2. Obtención de reglas de clasificación con PSO

La obtención de reglas de clasificación utilizando PSO, capaces de operar sobre atributos nominales y numéricos, requiere de una combinación de los métodos citados anteriormente (secciones 4.5 y 4.3) ya que es preciso decir cuáles serán los atributos que formarán parte del antecedente (discreto) y para los atributos numéricos es preciso determinar el intervalo a utilizar (continuo)

La i-ésima partícula de la población se representará de la siguiente forma:

- $pBin^i = (pBin^i_1, pBin^i_2, \ldots, pBin^i_n)$ almacena la posición actual de la partícula.
- $v1^i = (v1^i_1, v1^i_2, \ldots, v1^i_n)$ y $v2^i = (v2^i_1, v2^i_2, \ldots, v2^i_n)$ se combinan para determinar la dirección en la cual se moverá la partícula.
- $pBestBin^i = (pBestBin^i_1, pBestBin^i_2, \ldots, pBestBin^i_n)$ almacena la mejor solución encontrada por la partícula hasta el momento.
- $fitness^i$ es el valor de aptitud del individuo.
- $fitness\_pBest^i$ es el valor de aptitud de la mejor solución local encontrada (vector $pBestBin^i$)
- $pReal^i = (pReal^i_1, pReal^i_2, \ldots, pReal^i_n)$ sólo se utiliza para los atributos numéricos y contiene los límites actuales de los intervalos.
- $v3^i = (v3^i_1, v3^i_2, \ldots, v3^i_n)$ indica la dirección de cambio de $pReal^i$.

- $pBestReal^i = (pBestReal^i_1, pBestReal^i_2, \ldots, pBestReal^i_n)$ almacena la mejor solución encontrada por la partícula para los límites de los intervalos.

Cada vez que la i-ésima partícula se mueve se modifica su posición actual y los intervalos correspondientes a los atributos numéricos de la siguiente forma:

Parte binaria

$$v1^i_j(t+1) = w_{bin} \cdot v1^i_j(t) + \varphi_1 \cdot rand_1 \cdot (2\ pBestBin^i_j - 1) + \varphi_2 \cdot rand_2 \cdot (2\ lBestBin^i_j - 1) \quad (5.1)$$

donde, w_{bin} representa el factor de inercia, $rand_1$ y $rand_2$ son valores aleatorios con distribución uniforme en [0,1] y φ_1 y φ_2 son valores constantes de indican la importancia que se desea darle a las respectivas soluciones halladas previamente. Los valores $pBin^i_j$ y $lBest^i_j$ corresponden al j-ésimo dígito de los vectores binarios $pBestBin^i$ y $lBestBin^i$ respectivamente. En el método propuesto, cada partícula tendrá en cuenta la posición de su vecino más cercano con un valor de aptitud superior al suyo; por lo tanto el valor de $lBestBin^i$ corresponde al vector de $pBin^j$ de la partícula más cercana a $pBin^i$ siempre que $fitness^j$ sea mayor que $fitness^i$ usando distancia euclídea.

Nótese que, a diferencia del PSO Binario de [12], el desplazamiento del vector $v1^i$ en las direcciones correspondientes a la mejor solución encontrada por la partícula y al mejor local no dependen de la posición actual de dicha partícula, como se indica en [13]. Luego, cada elemento del vector velocidad $v1^i$ es controlado según 5.2

$$v1^i_j(t) = \begin{cases} \delta 1_j & si\ v1^i_j(t) > \delta 1_j \\ -\delta 1_j & si\ v1^i_j(t) \leq \delta 1_j \\ v1^i_j(t) & en\ caso\ contrario \end{cases} \quad (5.2)$$

donde

$$\delta 1_j = \frac{limite1_{superior_j} - limite1_{inferior_j}}{2} \quad (5.3)$$

Es decir que, el vector velocidad veloc1i se calcula según 5.1 y se controla según 5.2. Su valor se utiliza para actualizar el valor del vector velocidad $v2^i$, como se indica en 5.4.

$$v2^i_j(t+1) = v2^i_j(t) + v1^i_j(t+1) \quad (5.4)$$

El vector $v2^i$ también se controla de manera similar al vector $v1^i$ cambiando $limite1_{superior_j}$ y $limite1_{inferior_j}$ por $limite2_{superior_j}$ y $limite2_{inferior_j}$ respectivamente. Esto dará lugar a $\delta 2_j$ que será utilizado como en 5.2 para acotar los valores de $v2^i$. Luego se le aplica la función sigmoide 5.5 y se calcula la nueva posición de la partícula según (7).

$$sig(x) = \frac{1}{1 + e^{-x}} \quad (5.5)$$

$$pBin^i_j(t+1) = \begin{cases} 1 & si\ rand^i_j < sig(v2^i_j(t+1)) \\ 0 & si\ no \end{cases} \tag{5.6}$$

donde $rand^i_j$ es un número aleatorio con distribución uniforme en [0,1].

Parte continua

$$pReal^i_j(t+1) = pReal^i_j(t) + v3^i_j(t+1) \tag{5.7}$$

$$\begin{aligned} v3^i_j(t+1) &= w_{Real} \cdot v3^i_j(t) + \varphi_3 \cdot rand_3 \cdot (pBestReal^i_j - pReal^i_j) + \\ &\quad \varphi_4 \cdot rand_4 \cdot (lBestReal^i_j - pReal^i_j) \end{aligned} \tag{5.8}$$

donde nuevamente, w_{Real} representa el factor de inercia, $rand_3$ y $rand_4$ son valores aleatorios con distribución uniforme en [0,1] y φ_3 y φ_4 son valores constantes de indican la importancia que se desea darle a las respectivas soluciones halladas previamente. En este caso, $lBestReal^i$ corresponde al vector $pReal^j$ de la partícula más cercana a $pReal^i$ donde $fitness^j$ es mayor que $fitness^i$ usando distancia euclídea. Esta es la misma partícula de la que se tomó el vector $pBin^i$ para realizar el ajuste de $v1^i$ en 5.1. Los valores asignados a w_{Real} [15], $\varphi 1$, $\varphi 2$, $\varphi 3$ y $\varphi 4$ son importantes para garantizar la convergencia del algoritmo. Puede encontrar información más detallada referida a la selección de estos valores en [53] y [54]. Además, es importante remarcar que la incorporación de la función sigmoide 4.4 cambia radicalmente la manera de utilizar el vector velocidad para actualizar la posición de la partícula. En PSO continuo, el vector velocidad toma valores mayores al inicio para facilitar la exploración del espacio de soluciones y al final se reduce para permitir que la partícula se estabilice. En PSO binario ocurre precisamente todo lo contrario. Los valores extremos, al ser mapeados por la función sigmoide, producen valores de probabilidad similares, cercanos a 0 o a 1, reduciendo la chance de cambio en los valores de la partícula. Por otro lado, valores del vector velocidad cercanos a cero incrementan la probabilidad de cambio. Es importante considerar que si la velocidad de una partícula es el vector nulo, cada uno de los dígitos binarios que determina su posición tiene probabilidad 0.5 de cambiar a 1. Es la situación más aleatoria que puede ocurrir.

En este trabajo, los valores de $limite1$ y $limite2$ son iguales para todas las dimensiones; estos son [0,1] y [0,6] respectivamente. Por lo tanto, los valores de los vectores velocidad $v1$ y $v2$ fueron limitados a los rangos [-0.5, 0.5] y [-3,3] respectivamente. Es decir que pueden obtenerse probabilidades en el intervalo [0.0474, 0.9526]. Los valores para $\varphi 1$, $\varphi 2$, $\varphi 3$ y $\varphi 4$ fueron establecidos en 0.25, 0.25, 0.5 y 0.25 respectivamente. Los valores de w_{bin} y w_{Real} fueron establecidos entre 1.25 y 0.25 de manera lineal y proporcional a la cantidad de iteraciones realizadas, en forma ascendente para w_{bin} y en forma descendente para w_{Real}. El fitness de cada partícula se calcula en base al soporte, la confianza de la regla y la cantidad de atributos utilizados en el antecedente como se indica en 5.9

$$Fitness = penalizacion * \frac{soporte}{L} * confianza - factor * \frac{NumAtribs}{MaxAtribs} \tag{5.9}$$

donde *soporte* y *confianza* son las métricas correspondientes a la regla que representa la partícula, L es la cantidad de ejemplos a considerar, $factor$ es una constante que cuantifica la importancia que se le da a la longitud del antecedente, *NumAtribs* es la cantidad de comparaciones que intervienen en el antecedente de la regla (una por atributo) y *MaxAtribs* es la máxima cantidad de comparaciones que podrían presentarse.

La *penalizacion* es un valor entre 0.1 y 1 que reduce el producto de soporte y confianza si es necesario. El algoritmo 6 indica la forma de calcularlo.

Algoritmo 6: Pseudocódigo correspondiente al cálculo de la penalización de la regla

```
Function Penalizar(valor, Umbral1, Umbral2,Min, Max)
begin
    if valor < Umbral1 then
        Factor = Min
    else
        if valor < Umbral2 then
            Factor = Min + (Max − Min) * (valor−Umbral1)/(Umbral2−Umbral1) + Umbral
        else
            Factor = 1
    return (Factor)
```

Cálculo de la Penalizacion de la regla
SOP_MIN1 = max(2, 5 % de los ejemplos de la clase)
SOP_MIN2 = max(4, 7.5 % de los ejemplos de la clase)
Pena_SOP = Penalizar(soporte, SOP_MIN1, SOP_MIN2,0.1,0.75)
CONF_MIN1 = límite de confianza inferior (ej: 0.4)
CONF_MIN2 = límite de confianza superior (ej: 0.7)
Pena_CONF = Penalizar(confianza, CONF_MIN1, % CONF_MIN2,0.1,0.7)
Penalizacion = min(Pena_SOP, Pena_CONF)

A modo de ejemplificar lo dicho anteriormente, se utilizará la información de pacientes afectados con rinitis alérgica, contenida en la base Drug5, para los cuales se han relevado los siguientes atributos: Edad (Age), Sexo (Sex), Presión arterial (BP), Nivel del colesterol (Cholesterol), Nivel de sodio (Na). En base a los síntomas observados, los pacientes han sido diagnosticados con una de cinco drogas porsibles: DrugA, DrugB, DrugC, DrugX, DrugY. Esta última información se encuentra almacenada en el campo Clase.

La figura 5.5 muestra los metadatos correspondientes. En ella pueden observarse los seis atributos que contienen la información relavada del paciente y uno que contiene la droga suministrada. También puede verse que no hay datos faltantes y que los rangos de los atributos numéricos son: [15;74] para la edad, [0.5; 0.896] para el nivel de sodio en sangre y [0.02; 0.08] para el nivel de potasio. Los demás atributos son nominales. El sexo toma 2 valores posibles, la presión 3 valores distintos (high, normal y Low) y el colesterol toma 2 valores posibles (high, normal).

Según lo explicado anteriormente, el vector *pBin* de una partícula determinada, utiliza una representación binaria de longitud fija. En este caso, los tres atributos nominales requieren

Meta Data View | Data View | Plot View | Advanced Charts | Annotations

ExampleSet (200 examples, 1 special attribute, 6 regular attributes)

Name	Type	Statistics	Range	Missings
clase	polynominal	mode = drugY (91), least = drugC (16)	drugY (91), drugC (16), drugX (54), drugA (23), drugB (16)	0
Age	integer	avg = 44.315 +/- 16.544	[15.000 ; 74.000]	0
Sex	binominal	mode = M (104), least = F (96)	F (96), M (104)	0
BP	polynominal	mode = HIGH (77), least = NORMAL (59)	HIGH (77), LOW (64), NORMAL (59)	0
Cholesterol	binominal	mode = HIGH (103), least = NORMAL (97)	HIGH (103), NORMAL (97)	0
Na	real	avg = 0.697 +/- 0.119	[0.500 ; 0.896]	0
K	real	avg = 0.050 +/- 0.018	[0.020 ; 0.080]	0

Figura 5.5: Metadatos de la base de datos Drug5

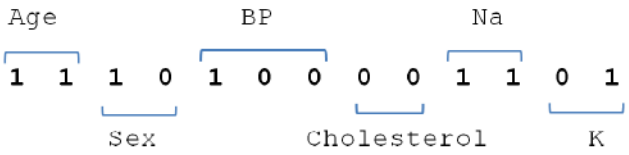


Figura 5.6: Vector *pBin* correspondiente a una partícula genérica para los datos de la base Drug5

7 dígitos binarios (2 para el sexo, 3 para el nivel de presión y 2 para el nivel de colesterol) y los tres atributos numéricos requieren otros 6 dígitos binarios más (2 para cada uno) dando lugar a un vector de longitud 13. La figura 5.6 ejemplifica esta situación.

La figura 5.7 ejemplifica un individuo completo de la población a evolucionar utilizando PSO donde la secuencia binaria indicada en la figura 5.6 es el contenido del vector *indivBinario*.

El vector *indivReal* contiene los límites de los intervalos de los atributos numéricos. Es decir que siguiendo la ubicación de los atributos indicada en la figura 5.6 de las 13 posiciones de este vector sólo se utilizan las dos primeras para limitar el atributo Age y las 4 últimas para los atributos Na y K respectivamente.

Los valores almacenados se encuentran escalados linealmente entre 0 y 1. Por ejemplo, las dos primeras posiciones del vector *indivReal* contienen los valores 0,06 y 0,45 como límites para la edad. Recuérdese que el rango original de este atributo es el intervalo [15; 74]. Es decir que si el 0 corresponde al valor 15 y el 1 al valor 74, el valor 0.06 corresponde a una edad de $15+0{,}06*(74-15) = 18{,}54$ años. De igual forma, el valor 0.45 corresponde a 41,75 años.

Nótese que la información almacenada en los campos $indivSoporte$, $soporte$ y $confianza$ no están incluídos en la descripción de la información de la partícula dada al inicio de esta sección. Esto se debe a que se trata de información adicional almacenada sólo para ahorrar tiempo de cálculo ya que se requiere al momento de visualizar la regla obtenida.

En el caso particular del campo $indivSoporte$ su valor emplea la misma representación que *pBin* pero sólo utiliza el valor 1 para aquellos valores de atributos o rangos de intervalos que tienen una probabilidad de ser incluídos dentro del antecedente superior al 80 %. Esto último se controla a través del respectivo valor del vector $v2$ el cual se

```
indivBinario: [1 1 1 0 1 0 0 0 0 1 1 0 1]
      veloc1: [0.02 -0.01 0.2 -0.06 -0.5 -0.01 -0.04 0.11 -0.21 0 0.06 0.06 0.04]
      veloc2: [1.46 1.43 3 -3 -3 -3 -3 -2.89 -3 0.1 0.16 1.84 1.83]
     fitness: 0.072219

pBestBinario: [1 1 1 0 1 0 0 0 0 0 0 1 1]
    fitpBest: 0.060311

   indivReal: [0.06 0.45 0.67 0.66 0.95 0.42 0.64 0.88 0.09 0.17 0.88 0.03 0.42]
   velocReal: [0.06 0.08 -0.15 0.11 0.05 0.01 0.04 -0.05 -0.06 0.03 0.08 -0.03 0.05]
   pBestReal: [0 0.37 0.82 0.55 0.9 0.41 0.6 0.93 0.16 0.14 0.8 0.05 0.36]

indivSoporte: [1 1 1 0 0 0 0 0 0 0 0 0 1]
     soporte: 17
   confianza: 0.85

-------------------------
Regla obtenida de indivSoporte
SI(Sex = F) and (18.78 < Age < 41.72) and (K < 0.04) --> drugY    (sop = 17; conf = 0.85)
```

Figura 5.7: Información almacenada en una partícula

utiliza como argumento para la sigmoide indicada en 5.5.

Es precisamente el vector *indivSoporte* el que determina la regla correspondiente al individuo y el cual se calcula el fitness almacenado en la partícula.

5.3. Método de obtención de reglas propuesto

Las reglas se obtienen a través de un proceso iterativo que analiza los ejemplos no cubiertos de cada clase comenzando por las más numerosas. Cada vez que se obtiene una regla, los ejemplos cubiertos correctamente por dicha regla son retirados del conjunto de datos de entrada. El proceso continúa hasta lograr cubrir todos los ejemplos o hasta que la cantidad de ejemplos no cubiertos de cada clase se encuentre por debajo del respectivo soporte mínimo establecido o hasta que se hayan realizado la máxima cantidad de intentos por obtener una regla, lo que ocurra primero. Es importante tener en cuenta es que, dado que los ejemplos son retirados del conjunto de datos de entrada a medida que son cubiertos por las reglas, las mismas constituyen una lista de clasificación. Es decir que, para clasificar un ejemplo nuevo, las reglas deben ser aplicadas en el orden en que fueron obtenidas y el ejemplo será clasificado con la clase correspondiente al consecuente de la primera regla cuyo antecedente se verifique para el ejemplo en cuestión.

Antes de comenzar con el proceso iterativo de obtención de reglas, el método inicia con el entrenamiento no supervisado de una red SOM utilizando el conjunto completo de ejemplos que se espera cubrir. La red se entrena de manera no supervisada siguiendo el algoritmo 3 y su función es identificar las zonas más prometedoras del espacio de búsqueda.

Dado que la red SOM sólo opera con datos numéricos, los atributos nominales son representados en forma binaria. Además, antes de iniciar el entrenamiento, cada dimensión correspondiente a un atributo numérico es escalada linealmente en [0,1]. La

medida de similitud utilizada es distancia euclídea. Una vez finalizado el entrenamiento, cada centroide contendrá aproximadamente el promedio de los ejemplos que representa.

Para obtener cada una de las reglas se determina, en primer lugar, cual es la clase correspondiente al consecuente. Buscando obtener reglas con soporte alto, el método propuesto comenzará a analizar primero las clases que posean un mayor número de ejemplos no cubiertos. El soporte mínimo que debe cumplir una regla es proporcional a la cantidad de ejemplos no cubiertos de la clase al momento en que fue obtenida. Es decir, que el soporte mínimo requerido para cada clase disminuye a lo largo de las iteraciones, a medida que los ejemplos de la clase correspondiente se van cubriendo. De esta forma, es de esperar que las primeras reglas posean mayor soporte que las últimas.

Una vez seleccionada la clase, queda determinado el consecuente de la regla. Para obtener el antecedente se optimizará, utilizando el algoritmo descripto en 3.6.1, una población de partículas inicializada con la información de todos los centroides capaces de representar un número mínimo de ejemplos de la clase seleccionada y sus vecinos inmediatos. La información del centroide se utiliza para determinar el vector $v2$ descripto en la sección 5.2. Si se trata de un atributo nominal, la información del centroide se escala linealmente al intervalo $[limite2_{inferior_j}, limite2_{superior_j}]$ pero si se trata de un atributo numérico el valor a escalar es $(1 - 1{,}5 * desviacion_j)$ siendo $desviacion_j$ la j-ésima dimensión de la desviación de los ejemplos representados por el centroide. En ambos casos se pretende operar con un valor entre 0 y 1 que mida el grado de participación del atributo (si es numérico) o del valor del atributo (si es nominal) en la construcción del antecedente de la regla. Cuando se trata de atributos nominales es claro que el promedio indica la proporción de elementos representados por el centroide que coinciden en el mismo valor pero cuando son numéricos, esta proporción no se encuentra presente en el centroide sino en la desviación de los ejemplos (siempre considerando una dimensión específica). Si la desviación en cierta dimensión es cero, todos los ejemplos coindicen en el valor del centroide pero si es demasiado amplia, debería entenderse que no es representativo del grupo y por lo tanto no sería conveniente incluirlo en el antecedente de la regla. Utilizando $(1 - 1{,}5 * desviacion_j)$ si la desviación es alta, el valor de la velocidad $v2$, argumento de la función sigmoide, será menor y se reducirá la probabilidad de que el atributo sea utilizado. En todos los casos la velocidad $v1$ se inicializa en forma aleatoria en $[limite1_{inferior_j}, limite1_{superior_j}]$. El algoritmo 7 muestra el pseudocódigo del método propuesto.

5.4. Resultados obtenidos

En esta sección se compara la performance del método propuesto con el método PART [16] en la obtención de las reglas de clasificación para un conocido conjunto de 17 bases de datos del repositorio UCI [80] y las bases “Drug”s [81].

En la tabla 5.1 se indica para cada una de ellas la cantidad de ejemplos que contiene, la cantidad de atributos que poseen de cada tipo y la cantidad de clases en las que dichos ejemplos pueden ser clasificados.

Para ello, se realizaron 30 corridas independientes de cada método y se utilizó una red

Algoritmo 7: Pseudocódigo del método propuesto

```
Entrenar la red SOM utilizando todos los ejemplos de entrenamiento;
Calcular el soporte mínimo para cada clase;
while no se alcance el criterio de terminación do
    Elegir la clase con mayor nro de ejemplos no cubiertos;
    Construir una poblacion reducida de individuos a partir de los centroides;
    Evolucionar la población utilizando PSO según lo visto en la sección 5.2;
    Obtener la mejor regla de la población;
    if la regla cumple con el soporte y la confianza pedidos then
        Agregar la regla al conjunto de reglas ;
        Considerar como cubiertos los ejemplos correctamente clasificados por la
        regla anterior;
        Recalcular el soporte mínimo para esta clase;
```

Base de Datos	#Ejemplos	#Atrib.Numéricos	#Atrib.Nominales	#Clases
Adult	32561	6	8	2
Balance scale	625	4	0	3
Breast cancer	286	0	9	2
Breast w	683	9	0	2
Credit-a	653	6	9	2
Credit-g	1000	7	13	2
Diabetes	768	8	0	2
Drug5	400	3	3	5
DrugY	200	3	3	5
Heart-c	296	6	7	2
Heart-statlog	270	13	0	2
Iris	150	4	0	3
Kr_vs_kp	3196	0	36	2
Mushroom	5644	0	21	2
Promoters	106	0	57	2
Soybean	562	0	35	15
Slice	3190	0	60	3
Wine	178	13	0	3
Zoo	101	1	16	7

Tabla 5.1: Características de las bases de datos utilizadas para medir el desempeño del método propuesto.

SOM de 30 neuronas organizadas en 6 filas y 5 columnas con 4 vecinos por neurona.

El método PART fue ejecutado con un factor de confianza de 0.3 para el podado del árbol y con el resto de sus parámetros con sus valores por defecto.

La Tabla 5.2 resume los resultados obtenidos al aplicar ambos métodos. En cada caso se ha considerado no sólo la precisión de la cobertura del conjunto de reglas sino también la claridad del modelo obtenido; esto último se refleja en la cantidad promedio de reglas obtenidas y en la cantidad promedio de términos utilizados para formar el antecedente. Se ha realizado en cada caso un test de diferencia de medias de dos colas con un nivel de significación de 0.05 donde la hipótesis nula implica que las medias son iguales. Dado que en cada caso se disponen de 30 corridas independientes de cada uno de los métodos, por el teorema central del límite se ha asumido que la distribución de cada muestra es normal. En base a los resultados obtenidos, cuando la diferencia es significativa, según el nivel indicado, se ha marcado en la tabla la mejor opción en negrita.

Como puede observase, en los 6 casos en los que la precisión de PART es superior a la del método propuesto, la cobertura obtenida por SOM+PSO es muy buena si se tiene en cuenta el reducido número de reglas que utiliza. Por ejemplo, para la base *"balance_scale"*, SOM+PSO tiene una precisión inferior a PART en aproximadamente un 8 % pero utiliza un quinto de la cantidad de reglas con menos consultas en cada antecedente. Algo parecido ocurre en los otros 5 casos donde PART posee una precisión superior a SOM+PSO. Esto se debe al énfasis puesto en la simplificación del modelo. En los 13 casos restantes o no hay diferencias significativas en la precisión alcanza o SOM+PSO es mejor. Independientemente de la precisión, la longitud promedio del conjunto de reglas siempre ha sido menor en el método propuesto.

5.5. Conclusiones

Se han analizado distintas variantes de PSO y se propuso una representación para obtener reglas de clasificación que poseen la capacidad de operar con atributos numéricos y nominales.

En la representación elegida se combina una representación binaria que permite seleccionar los atributos que intervienen en la regla con una representación continua sólo utilizada para determinar los límites de los atributos numéricos que intervienen en el antecedente.

Se utilizó una red neuronal SOM para inicializar adecuadamente la población de reglas. Los centroides obtenidos al agrupar los datos disponibles de manera no supervisada permiten identificar la relevancia que cada atributo tiene para el cúmulo de ejemplos que lo forman. De todas formas, esta métrica no es suficiente para seleccionar los atributos que formarán una regla y es aquí donde el PSO toma el control para llevar a cabo la selección final.

Las mediciones realizadas permiten afirmar que el método SOM+PSO obtiene un modelo más simple ya que en promedio utiliza aproximadamente el 20 % de la cantidad de reglas que genera PART, con antecedentes formados por pocas condiciones y una precisión aceptable.

Base de datos	**Precisión**		**Long.Promedio antecedente**		**Tamaño Conj.Reglas**	
	SOM+PSO	PART	SOM+PSO	PART	SOM+PSO	PART
Balance scale	$0,713 \pm 0,01$	$\mathbf{0,815 \pm 0,02}$	$\mathbf{1,93 \pm 0,1}$	$3,098 \pm 0,06$	$\mathbf{7,106 \pm 0,51}$	$38,71 \pm 1,18$
Breast cancer	$\mathbf{0,708 \pm 0,02}$	$0,657 \pm 0,02$	$\mathbf{1,164 \pm 0,03}$	$2,039 \pm 0,07$	$\mathbf{5,033 \pm 0,19}$	$19,243 \pm 1,49$
Breast w	$0,945 \pm 0,00$	$\mathbf{0,955 \pm 0,00}$	$\mathbf{1,88 \pm 0,15}$	$2,115 \pm 0,1$	$\mathbf{2,853 \pm 0,18}$	$10,11 \pm 0,54$
credit_a	$\mathbf{0,859 \pm 0,01}$	$0,738 \pm 0,02$	$\mathbf{1,278 \pm 0,11}$	$2,46 \pm 0,08$	$\mathbf{3,446 \pm 0,21}$	$32,95 \pm 1,88$
Credit-g	$0,703 \pm 0,01$	$0,701 \pm 0,01$	$\mathbf{1,055 \pm 0,05}$	$2,581 \pm 0,07$	$\mathbf{2,323 \pm 0,2}$	$62,543 \pm 1,65$
Diabetes	$\mathbf{0,738 \pm 0,01}$	$0,731 \pm 0,01$	$\mathbf{1,176 \pm 0,08}$	$1,98 \pm 0,1$	$\mathbf{2,55 \pm 0,2}$)	$7,723 \pm 0,64$
Heart-c	$0,721 \pm 0,02$	$\mathbf{0,764 \pm 0,02}$	$\mathbf{1,565 \pm 0,08}$	$2,564 \pm 0,12$	$\mathbf{3,19 \pm 0,17}$	$19,24 \pm 0,75$
Heart-statlog	$0,727 \pm 0,02$	$\mathbf{0,767 \pm 0,01}$	$\mathbf{1,616 \pm 0,13}$	$2,875 \pm 0,1$	$\mathbf{4,276 \pm 0,35}$	$17,616 \pm 0,68$
Iris	$0,924 \pm 0,02$	$\mathbf{0,939 \pm 0,02}$	$1,132 \pm 0,05$	$\mathbf{1,02 \pm 0,03}$	$\mathbf{3,623 \pm 0,15}$	$4,113 \pm 0,32$
Mushroom	$0,939 \pm 0,00$	$\mathbf{0,997 \pm 0,00}$	$1,34 \pm 0,1$	$\mathbf{1,259 \pm 0,02}$	$\mathbf{3,503 \pm 0,14}$	$11,196 \pm 0,24$
Promoters	$0,65 \pm 0,03$	$0,67 \pm 0,05$	$1,097 \pm 0,02$	$\mathbf{1,03 \pm 0,04}$	$\mathbf{7,013 \pm 0,26}$	$7,243 \pm 0,41$
Soybean	$\mathbf{0,856 \pm 0,01}$	$0,577 \pm 0,07$	$5,962 \pm 0,26$	$\mathbf{2,731 \pm 0,05}$	$\mathbf{24,856 \pm 0,5}$	$31,92 \pm 0,65$
Kr_vs_kp	$0,933 \pm 0,00$	$\mathbf{0,99 \pm 0,00}$	$\mathbf{2,362 \pm 0,06}$	$3,122 \pm 0,08$	$\mathbf{3 \pm 0}$	$22,196 \pm 0,63$
Zoo	$\mathbf{0,923 \pm 0,02}$	$0,187 \pm 0,05$	$1,527 \pm 0,1$	$\mathbf{1,478 \pm 0,01}$	$\mathbf{6,993 \pm 0,1}$	$7,67 \pm 0,07$
Slice	$\mathbf{0,812 \pm 0,01}$	$0,722 \pm 0,01$	$\mathbf{1,485 \pm 0,05}$	$2,663 \pm 0,04$	$\mathbf{10,046 \pm 0,33}$	$103,303 \pm 1,58$
Wine	$0,867 \pm 0,03$	$\mathbf{0,888 \pm 0,01}$	$3,052 \pm 0,45$	$\mathbf{1,53 \pm 0,07}$	$\mathbf{4,803 \pm 0,38}$	$5,44 \pm 0,25$
Drug5	$0,85 \pm 0,01$	$0,87 \pm 0,06$	$\mathbf{1,741 \pm 0,07}$	$2,066 \pm 0,03$	$\mathbf{7,313 \pm 0,31}$	$15,42 \pm 0,45$
DrugY	$\mathbf{0,844 \pm 0,02}$	$0,668 \pm 0,09$	$\mathbf{1,657 \pm 0,13}$	$1,868 \pm 0,03$	$\mathbf{6,45 \pm 0,23}$	$11,79 \pm 0,47$
Adult	$0,802 \pm 0,00$	$0,801 \pm 0,00$	$\mathbf{2,229 \pm 0,22}$	$4,679 \pm 0,05$	$\mathbf{2,153 \pm 0,14}$	$975,423 \pm 10,59$

Tabla 5.2: Resultados obtenidos al aplicar los métodos SMO+PSO y PART a un conjunto de 17 bases del repositorio UCI, además de las bases Drug5 y DrugY. Se ha medido la precisión, la longitud promedio del antecedente de cada regla y la cantidad de reglas utilizadas en cada caso. Se indica en negrita la mejor solución utilizando un test Student con nivel de significación 0.05

Capítulo 6

Conclusiones generales y trabajos futuros

La Minería de Datos es un tema de sumo interés en la actualidad. Todo aquel que disponga de información histórica de su proceso ya sea industrial, comercial, académico o educativo, estará interesado en obtener información que le ayude a tomar decisiones.

Esto ha hecho que el término *Minería de Datos* sea uno de los que más ha crecido en popularidad en los medios de difusión escrita y oral durante los últimos 7 años.

Dentro de la Minería de Datos, las reglas de clasificación constituyen un modelo generalmente aceptado ya que de su lectura se obtiene una justificación inmediata de las decisiones sugeridas.

Es importante recordar que, generalmente, quienes obtienen el o los modelos sobre los datos y quienes deben hacer uso de ellos para tomar decisiones son grupos de personas diferentes, no sólo por su composición sino por los intereses que tienen al respecto. Es por esto que resulta importante que las reglas sean fáciles de leer y que el conjunto de reglas tenga una cardinalidad baja. Esto último fue el objetivo central perseguido por el método presentado en esta tesina donde el énfasis estuvo puesto en la obtención de un pequeño grupo de reglas sencillas capaces de describir la información disponible.

El método propuesto en esta tesina comenzó utilizando únicamente la técnica de optimización para obtener el conjunto de reglas que operaban sobre atributos numéricos. Los inconvenientes aparecieron cuando se incorporaron los atributos nominales. El problema central de este enfoque es que para operar sobre atributos nominales se requiere de un gran número de ejemplos dentro de la base que permitan calificar a las reglas en formación (partículas). En este punto, el uso de la red SOM para inicializar las partículas permitió establecer un punto de partida favorable y así evitar explorar opciones no contempladas en la información disponible,

Una característica interesante de la combinación de SOM y PSO, que se ha observado a lo largo del proceso adaptativo, es la reducida cantidad de iteraciones que la técnica de optimización requiere para mejorar el conjunto de reglas iniciales sugerido a partir de los agrupamientos generados con la red SOM. Si bien no han sido incluídas en esta tesina, las mediciones realizadas aplicando el método propuesto pero omitiendo la optimización que

introduce PSO ha dado lugar a un conjunto de reglas de precisión similar pero con una cardinalidad ligeramente superior a PART. Esto demuestra que las pocas iteraciones que se realizan con PSO a la información de los centroides es fundamental para determinar un adecuado conjunto de reglas.

Por otro lado, si bien en base a las pruebas realizadas no se ha evidenciado ninguna dependencia entre los resultados obtenidos y el tamaño inicial de la red SOM, se considera de interés repetir las mediciones utilizando una red SOM dinámica.

Sin duda, el corazón de este tipo de técnicas de búsqueda reside en la función de aptitud. Ella es la encargada de decidir cuando un individuo es mejor que otro. En esta tesina, se decidió enfrentar el problema desde el punto de vista mono objetivo. Esto lleva a resolver durante el proceso, situaciones de compromiso que bajo otro enfoque podrían ser analizadas de manera independiente. La expresión actual de la función de fitness combina un par de métricas de la regla con la longitud de su antecedente. Bajo un enfoque multi objetivo, podrían coexistir soluciones más eficientes y complejas con otras de menor calidad y mayor simplicidad. Luego, analizando el espacio de soluciones podría surgir un conjunto de reglas de mejor calidad que el obtenido hasta el momento.

Los resultados de esta investigación han sido presentados por el autor de esta tesina en el *XIII Workshop de Agentes y Sistemas Inteligentes* realizado en el marco del *XVIII Congreso Argentino de Ciencias de la Computación* realizado en Bahía Blanca en Octubre de 2012 y publicados bajo el título ***Obtención de reglas de clasificación usando SOM+PSO*** [82].

Apéndice A

RapidMiner

RapidMiner [83], es una solución completa de Inteligencia Empresarial que hace foco en la minería de datos y análisis predictivo. Utiliza una amplia variedad de técnicas descriptivas y predictivas para ayudar en la toma de decisiones. Se distribuye bajo licencia de código abierto AGPL (Affero General Public License), una licencia derivada de la Licencia Pública General de GNU. Su primera versión fue creada por la Universidad de Dortmund (Alemania) en 2001. Está desarrollado en Java al igual que Weka [84], su predecesor.

El producto está disponible en la versión libre RapidMiner Community Edition (utilizada en este trabajo) puede ser descargada desde su sitio web de forma gratuita y también en la version RapidMiner Enterprise Edition (no gratuita), que combina las ventajas de la Community Edition con el soporte profesional con garantía de tiempo de respuesta ofrecido por la empresa.

Para utilizar la versión libre sólo es necesario descargar el paquete de instalación apropiado para el sistema operativo del que se dispone, e instalar de acuerdo a las instrucciones provistas en el sitio web. Se soportan todas las versiones de Windows, Macintosh, Linux y Unix. También es necesaria una maquina virtual java actualizada.

La interfaz gráfica de usuario de RapidMiner permite el diseño de procesos analíticos auto-documentados que pueden actualizarse y re-utilizarse fácilmente para nuevos problemas. Provee gran cantidad de métodos de integración y transformación de datos, selección de atributos, análisis y modelado, con herramientas para visualización de los resultados. La Figura A.1 presenta la interfaz gráfica de RapidMiner con el proyecto de clustering de alumnos que abandonaron, desarrollado en el presente trabajo.

La herramienta dispone de un amplio número de extensiones que pueden instalarse, entre ellas se destaca Weka, código abierto con licencia GNU, que ofrece una colección de algoritmos de aprendizaje para tareas de minería de datos. Otras extensiones incluyen Text (para análisis estadístico de textos), Web Mining (para análisis de páginas web), R-connector (integración con el lenguaje R de programación de análisis estadístico).

RapidMiner provee acceso a buena cantidad de tipos de archivos y bases de datos (Microsoft Excel, Microsoft Access, Oracle, IBM DB2, Microsoft SQL Server, MySQL, Postgres, Teradata, Ingres,VectorWise, SAP, paginas web, pdf, html, xml, etc.).

También ofrece más de 500 operadores para una amplia cantidad de tareas de minería de datos y otras características relacionadas como entrada, salida y procesamiento de datos. Entre ellas:

- Muestreo de datos
- Particionamiento de conjuntos de datos
- Transformaciones de datos
- Selección de atributos
- Generación de atributos
- Estadísticas descriptivas
- Gráficos y visualización
- Agrupamiento
- Reglas de asociación
- Arboles de decisión
- Reglas de inducción
- Modelos Bayesianos
- Regresión
- Redes Neuronales
- Máquinas de soporte vectorial
- Combinación de modelos
- Evaluación de modelos

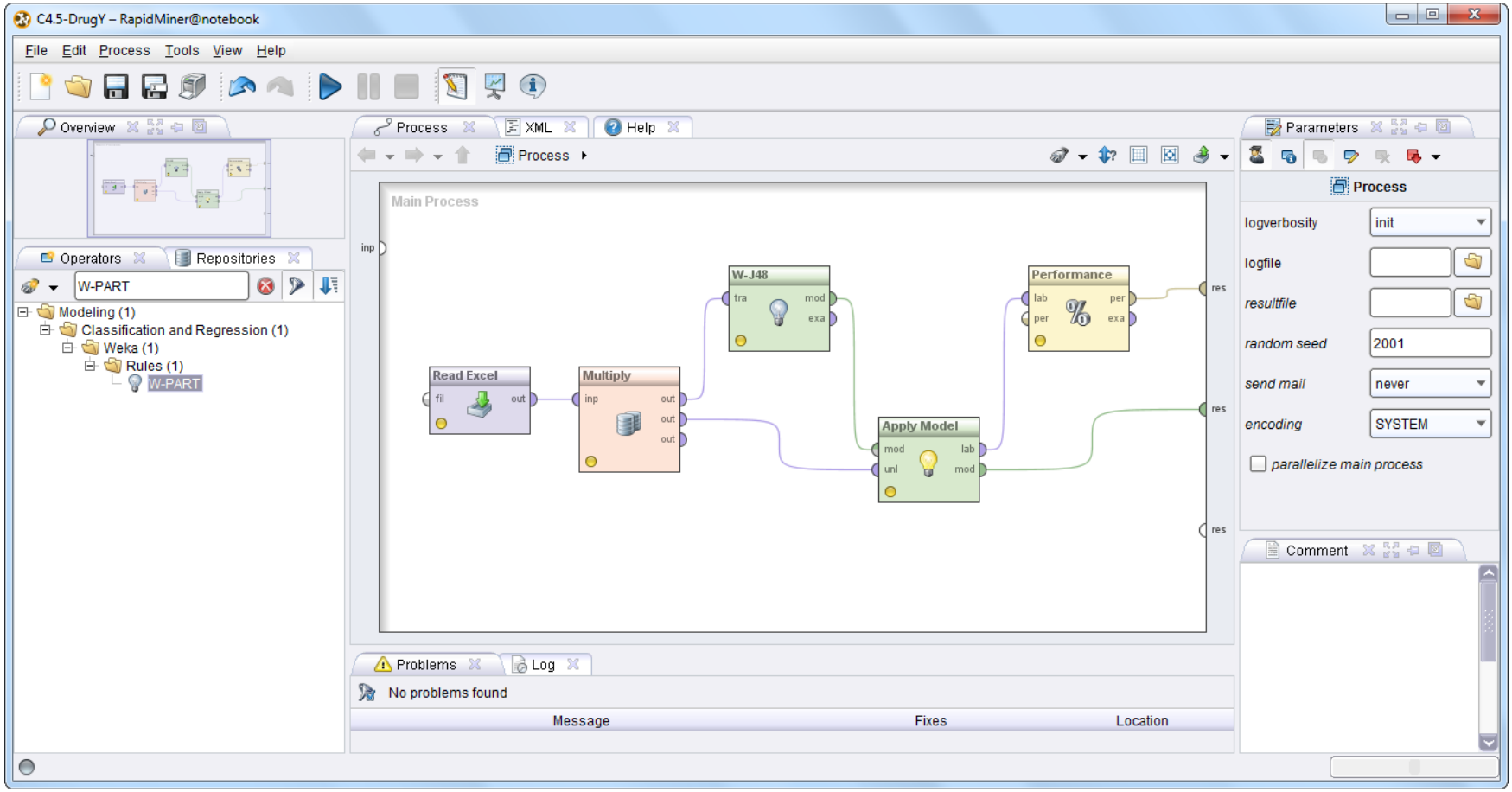


Figura A.1: Interfaz gráfica de RapidMiner

Indice de figuras

1.1. Fases que componen el proceso de $\mathcal{KDD}$ 11
1.2. Disciplinas que contribuyen a la Minería de Datos 12
1.3. Integración en un almacén de datos 14
1.4. Ejemplo de discretización del atributo `Nota` 17
1.5. Método de validación cruzada de k pliegues 19

2.1. Arbol de clasificación para el suministro de drogas a pacientes con rinitis alérgica 27
2.2. Matriz de confusión obtenida a partir del resultado de aplicar el árbol de la figura 2.1 a la base DrugY 31
2.3. Árbol de decisión para determinar si se juega o no al golf 31
2.4. Clasificación de ejemplos con dos atributos. A la izquierda se encuentra la partición que cada nodo del árbol realiza sobre los atributos y a la derecha se muestran dos reglas que permiten cubrirlos sin superposición 33
2.5. Lista de clasificación obtenida a partir de la tabla 2.1 36
2.6. Proceso de construcción de una lista de clasificación 37
2.7. Árbol de decisión redundante para una disyunción simple 39
2.8. Construcción de la regla 1 45
2.9. El error correspondiente a los datos de la rama "*(% materias aprobadas = [0, 0.25]) AND (Trabaja = SI)*" es 0.3469 mientras que el correspondiente al subárbol generado a partir del atributo '*Promedio*" es 0.4291. Por ser este último mayor, el subárbol se descarta 46
2.10. Construcción de la regla 1 47
2.11. El error correspondiente a los datos de la rama "*(% materias aprobadas = [0, 0.25]) AND (Trabaja = NO)*" es 0.4203 mientras que el correspondiente al subárbol generado a partir del atributo "*Promedio*" es 0.1834. Por ser este último menor, el subárbol no se descarta 48
2.12. Arbol parcial del que se extrae la regla 1 51

3.1. Agrupamiento jerárquico aglomerativo realizado sobre los datos de la tabla 3.1. Los números sobre el eje horizontal representan el número de ejemplo 55

3.2. Agrupamiento partitivo realizado sobre los datos de la tabla 3.1. Cada grupo posee un centroide que resume las características de los elementos que lo forman 56
3.3. Estructura clásica de una red SOM 61

4.1. Clasificación de las técnicas de optimización 68
4.2. Clasificación de las metaheurísticas 71
4.3. Ejemplos de *Inteligencia de Cúmulo* en la naturaleza 74
4.4. Movimiento de una partícula en el espacio de soluciones 75
4.5. Ejemplos de entornos sociales y geográficos en un espacio de soluciones . 78

5.1. Codificación entera del antecedente de una regla de longitud variable . . . 85
5.2. Ejemplo de codificación entera de longitud fija para el antecedente de una regla 86
5.3. Ejemplo de codificación binaria de longitud fija para antecedentes de una regla 87
5.4. Ejemplo de codificación utilizada en esta tesina. Nótese el agregado de un vector real a fin de no forzar la discretización inicial de los atributos numéricos. El vector binario identifica los atributos que componen el antecedente y el vector real determina los intervalos a considerar (si corresponde) 87
5.5. Metadatos de la base de datos Drug5 92
5.6. Vector *pBin* correspondiente a una partícula genérica para los datos de la base Drug5 92
5.7. Información almacenada en una partícula 93

A.1. Interfaz gráfica de RapidMiner 103

Lista de Algoritmos

1. Algoritmo genérico de construcción de un árbol de decisión 28
2. Expansión de conjunto de ejemplos para formar un árbol parcial 42
3. Pseudocódigo básico de entrenamiento de la red SOM 62
4. Algoritmo PSO Básico 76
5. Algoritmo PSO Binario 80
6. Pseudocódigo correspondiente al cálculo de la penalización de la regla . . 91
7. Pseudocódigo del método propuesto 95

Indice de tablas

2.1. Conjunto de datos de condiciones climáticas para jugar al golf formado sólo por atributos categóricos 32

2.2. Tabla cuyas filas se refieren a compras realizadas en un comercio y las columnas a cada uno de los productos en venta en dicho comercio 34

2.3. Versión de la tabla 2.1 con algunos atributos numéricos 35

2.4. Información referida a 24 alumnos que espera ser utilizada para determinar las características principales que determinan su condición de regularidad 44

2.5. Tasa de ganancia calculada para cada atributo de la tabla 2.4 con el objetivo de determinar la raíz del árbol parcial 44

2.6. Selección del atributo que mejor separa los ejemplos correspondientes a la rama "*(% materias aprobadas = [0, 0.25]) AND (Trabaja = NO)*". Este atributo se ubicará en el nivel 2 del árbol parcial 46

2.7. Selección del atributo que mejor separa los ejemplos correspondientes a la rama "*(% materias aprobadas = [0, 0.25]) AND (Trabaja = NO)*". Este atributo se ubicará en el nivel 2 del árbol parcial . 47

3.1. Información no etiquetada de 16 alumnos de una unidad académica . . . 54

5.1. Características de las bases de datos utilizadas para medir el desempeño del método propuesto. 95

5.2. Resultados obtenidos al aplicar los métodos SMO+PSO y PART a un conjunto de 17 bases del repositorio UCI, además de las bases Drug5 y DrugY. Se ha medido la precisión, la longitud promedio del antecedente de cada regla y la cantidad de reglas utilizadas en cada caso. Se indica en negrita la mejor solución utilizando un test Student con nivel de significación 0.05 97

Bibliografía

[1] Fayyad Usama, Gregory Piatetsky-Shapiro, and Padhraic Smyth. The kdd process for extracting useful knowledge from volumes of data. *Commun. ACM*, 39(11):27–34, November 1996.

[2] José Hernández Orallo, M.José Ramírez Quintana, and Cèsar Ferri Ramírez. *Introducción a la Minería de Datos*. Editorial Pearson, 2004.

[3] Christopher Westphal and Blaxton Teresa. *Data Mining Solutions. Methods and Tools for Solving Real-World Problems*. John Wiley & Sons Inc., 1998.

[4] Laura Lanzarini, Pedro Estelrrich, and Raúl Champredonde. Sistema automático de diagnóstico de alteraciones del equilibrio, 2000.

[5] Javier López, Laura Lanzarini, and Armando De Giusti. Evolutionary multiobjective optimization for emergency medical services. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, GECCO '11, pages 83–84, New York, NY, USA, 2011. ACM.

[6] Augusto Villa Monte, César Estrebou, and Laura Lanzarini. E-mail processing using data mining techniques. In EDULP, editor, *Computer Science & Technology Series: XVI Argentine Congress of Computer Science - Selected Papers*, pages 109–120, 2011.

[7] Laura Lanzarini, Augusto Villa Monte, and César Estrebou. E-mail processing with fuzzy soms and association rules, Abril 2011.

[8] César Estrebou, Laura Lanzarini, and Waldo Hasperué. Voice recognition based on probabilistic som. In *XXXVI Congreso Latinoamericano de Informática, CLEI*, Asunción, Paraguay, Octubre 2010.

[9] Juan Maulini and Laura Lanzarini. Face recognition using sift descriptors and binary pso with velocity control. In EDULP, editor, *Computer Science & Technology Series: XVII Argentine Congress of Computer Science - Selected Papers*, pages 43–53, 2012.

[10] Bernarda Albanesi, Nadia Funes, Franco Chichizola, and Lanzarini Laura. Reconocimiento de objetos en video utilizando sift paralelo. In Universidad de Morón, editor, *XVI Congreso Argentino de Ciencias de la Computación, CACIC 2010*, pages 475–482, Morón, Buenos Aires (Argentina), Octubre 2010.

[11] Laura Lanzarini, Juan Maulini, Augusto Villa Monte, Leonardo Corbalán, and María Delia Grossi. Técnicas de minería de datos aplicadas al diseño de un curso de estadística. In *V Congreso de Tecnología en Educación y Educación en Tecnología (TE&ET 2010)*, pages 365–372, El Calafate, Santa Cruz (Argetina), Mayo 2010.

[12] Waldo Hasperué, Laura Lanzarini, and Armando De Giusti. Rule extraction on numeric datasets using hyper-rectangles. *Computer and Information Science*, 5(4):116–131, 2012.

[13] Franco Ronchetti and Laura Lanzarini. Automatic vehicle parking using an evolution-obtained neural controller isbn=978-950-34-0756-1, pages=71-80, book-title = XVII Congreso Argentino de Ciencias de la Computación, CACIC 2011, month=Octubre, year = 2011, address = La Plata, Buenos Aires (Argetina),.

[14] Longbing Cao. Domain-driven data mining: Challenges and prospects. *Knowledge and Data Engineering, IEEE Transactions on*, 22(6):755 –769, june 2010.

[15] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[16] Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 144–151, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[17] Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[18] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[19] Ultima version del código fuente del c4.5 publicada por su autor. `http://www.rulequest.com/Personal/c4.5r8.tar.gz`.

[20] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, San Francisco, CA, 3th edition, 2011.

[21] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[22] Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–91, 1993.

[23] Tobias Scheffer. Finding association rules that trade support optimally against confidence. In *5th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 424–435. Springer-Verlag, 2001.

[24] Jadzia Cendrowska. Prism: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370, 1987.

[25] William W. Cohen. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[26] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982.

[27] J. C. Gower. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 53(3/4):325–338, 1966.

[28] T. Kohonen, M. R. Schroeder, and T. S. Huang, editors. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.

[29] Bernd Fritzke. Growing self-organizing networks - why? In *In ESANN?96: European Symposium on Artificial Neural Networks*, pages 61–72. Publishers, 1996.

[30] John Moody and Christian Darken. Learning with localized receptive fields. In *In Proceedings of the 1988 Connectionist Models Summer School*, pages 133–143. Morgan Kaufmann, 1988.

[31] Bernd Fritzke. Growing cell structures: A self organizing network for supervised and un-supervised learning. *Neural networks*, 7(9):1441–1460, 1993.

[32] T. Martinetz and K. Schulten. A "neural-gas"network learns topologies. *Artificial Neural Networks*, I:397–402, 1991.

[33] Thomas Martinetz. Competitive hebbian learning rule forms perfectly topology preserving maps. In *ICANN ?93*, pages 427–434. Springer London, 1993.

[34] Bernd Fritzke. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, 1995.

[35] Damminda Alahakoon, Saman K. Halgamuge, and Bala Srinivasan. Dynamic self-organizing maps with controlled growth for knowledge discovery. *Neural Networks, IEEE Transactions on*, 11(3):601 –614, may 2000.

[36] R. Marti. Procedimientos metaheurísticos en optimización combinatoria.

[37] F. Glover. Future paths for integer programming and links to artifcial intelligence. *Computers and Operations Research*, (13):533–549, 1986.

[38] C.R. Reeves. Modern heuristic techniques for combinatorial problems. *Blackwell Scientific Publishing*, 1993.

[39] E. Alba. Parallel metaheuristics: A new class of algorithms. *John Wiley and Sons*, October 2005.

[40] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[41] F. Glover and G. Kochenberger. Handbook of metaheuristics. *Kluwer Academic Publishers*, 2002.

[42] T. Crainic and M. Toulouse. Handbook of metaheuristics. *chapter Parallel Strategies for Metaheuristics*, pages 475–513, 2003.

[43] C. Gelatt S. Kirkpatrick and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[44] T. Feo and M. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optmization*, (6):109–133, 1999.

[45] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers Oper. Res*, (24):1097–1100, 1997.

[46] T. Stützle. Local search algorithms for combinatorial problems analysis, algorithms and new applications. *Technical report, DISKI Dissertationen zur Künstliken Intelligenz.*, 1999.

[47] D. Fogel T. Bäck and Z. Michalewicz. Handbook of evolutionary computation. *IOP Publishing and Oxford University Press*, Feb 1997.

[48] J. Owens L. Fogel and M. Walsh. Artificial intelligence through simulated evolution. 1966.

[49] I. Rechenberg. Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution. *Fromman-Holzboog Verlag*, 1973.

[50] J. Holland. Adaptation in natural and artificial systems. *The MIT Press*, 1975.

[51] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, (8):156–166, 1977.

[52] M. Dorigo. Optimization, learning and natural algorithms. *PhD thesis, Dipartamento di Elettronica*, 1992.

[53] J. Kennedy and R. Eberhart. Particle swarm optimization. *IEEE International Conference on Neural Networks*, IV:1942–1948, 1995.

[54] J. Kennedy and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 2001.

[55] R.Poli. An analysis of publications on particle swarm optimisation applications. Technical Report CSM-469, Department of Computer Science, University of Essex, UK, 2007.

[56] R.Poli. Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications*, pages 1–10, 2008.

[57] Shi Y. and Eberhart R. Parameter selection in particle swarm optimization. *7th International Conference on Evolutionary Programming.*, pages 591–600, 1998.

[58] Kennedy J. Clerc M. The particle swarm Â– explosion, stability and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation.*, 6(1):58–73, 2002.

[59] Van den Bergh F. An analysis of particle swarm optimizers. *Ph.D. dissertation. Department Computer Science. University Pretoria. South Africa.*, 2002.

[60] Esquivel S. Cagnina L. and Coello Coello C. A bi-population pso with a shake-mechanism for solving constrained numerical optimization. *IEEE Congress on Evolutionary Computation, CEC 2007*, pages 670–676.

[61] Ward C. Mohais A, Mendes R. and Postho C. Neighborhood re-structuring in particle swarm optimization. advances in artificial intelligence. *Lecture Notes in Computer Science*, 3809/2005:776–785, 2005.

[62] Atyabi A. et all. Particle swarm optimizations: A critical review. *5th International Conference on Information and Knowledge Technology*, 2007.

[63] De Giusti A. Lanzarini L., Leza V. Particle swarm optimization with variable population size. *Artificial Intelligence and Soft Computing Â– ICAISC 2008.*, 5097/2008:438–449, 2008.

[64] Vesterstrom J. Riget J. A diversity-guided particle swarm optimizer Â– the arpso. *EVALife Project Group Department of Computer Science*, 2002.

[65] Clerc M. Stagnation. Analysis in particle swarm optimisation or what happens when nothing happens. *Department of Computer Science; University of Essex. Technical Report CSM-460.*, 2006.

[66] Van den Bergh F.and Engelbrecht A. Peer E. Using neighbourhoods with the guaranteed convergence pso. *Swarm Intelligence Symposium. SIS '03.*, pages 235–242, 2003.

[67] J. Kennedy. The particle swarm: Social adaptation of knowledge. *IEEE International Conference on Evolutionary Computation*, pages 303–308, 1997.

[68] J. García Nieto. *Algoritmos basados en Cúmulos de Partículas para la resolución de problemas complejos*. PhD thesis, Universidad de Málaga, 2006.

[69] Kennedy J. and Eberhart R. A discrete binary version of the particle swarm algorithm. *World Multiconference on Systemics, Cybernetics and Informatics (WMSCI)*, pages 4104–4109, 1997.

[70] A.De Giusti J. López, L. Lanzarini. Particle swarm optimization with oscillation control. *ACM Special Interest Group on Genetic and Evolutionary Computation. Springer*, pages 1751–1752, 2009.

[71] Laura Lanzarini, Javier López, Juan Andrés Maulini, and Armando De Giusti. A new binary pso with velocity control. In *Proceedings of the Second international conference on Advances in swarm intelligence - Volume Part I*, ICSI'11, pages 111–119, Berlin, Heidelberg, 2011. Springer-Verlag.

[72] Stephen Frederick Smith. *A learning system based on genetic adaptive algorithms*. PhD thesis, Pittsburgh, PA, USA, 1980. PhD thesis.

[73] John H. Holland and Judith S. Reitman. Cognitive systems based on adaptive algorithms. *SIGART Bull.*, (63):49–49, 1977.

[74] Gilles Venturini. Sia: A supervised inductive algorithm with genetic search for learning attributes based concepts. In *Proceedings of the European Conference on Machine Learning*, ECML '93, pages 280–296, London, UK, UK, 1993. Springer-Verlag.

[75] Kenneth A. De Jong, William M. Spears, and Diana F. Gordon. Using genetic algorithms for concept learning. *Mach. Learn.*, 13(2-3):161–188, November 1993.

[76] DavidPerry Greene and StephenF. Smith. Competition-based induction of decision models from examples. *Machine Learning*, 13:229–257, 1993.

[77] Attilio Giordana and Filippo Neri. Search-intensive concept induction. *Evol. Comput.*, 3(4):375–416, December 1995.

[78] Cezary Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Mach. Learn.*, 13(2-3):189–228, November 1993.

[79] Wojciech Kwedlo and Marek Kr?towski. Discovery of decision rules from databases: An evolutionary approach. In *Principles of Data Mining and Knowledge Discovery*, volume 1510 of *Lecture Notes in Computer Science*, pages 370–378. Springer Berlin Heidelberg, 1998.

[80] Uc irvine machine learning repository. `http://archive.ics.uci.edu/ml/`. [Ultimo acceso : 11-Feb-2013].

[81] `http://www.facweb.iitkgp.ernet.in/~sudeshna/courses/ml08/drugte.arff`. [Ultimo acceso : 11-Feb-2013].

[82] Augusto Villa Monte, Franco Ronchetti, Laura Lanzarini, and Marcela Jeréz. Obtención de reglas de clasificación usando som+pso. In Universidad Nacional del Sur, editor, *XVIII Congreso Argentino de Ciencias de la Computación, CACIC 2012*, pages 210–219, Bahía Blanca, Buenos Aires (Argentina), Octubre 2012.

[83] Rapidminer. `http://rapid-i.com/content/view/181/190/`.

[84] Machine Learning Group at the University of Waikato. Weka. `http://www.cs.waikato.ac.nz/~ml/weka/`.