Universidad de las Ciencias Informáticas

Facultad 6

Título: "Herramienta para la generación de reglas de Clasificación basada en un algoritmo de Programación Genética"

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores

Yoander Cabanes Risco

José Leandro González González

Tutores

MSc. Gilberto Arias Naranjo

MSc. Tomás Gámez Acosta

Ing. Lesley Méndez Cáceres

Junio, 2008

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los _____ días del mes de _____ del _____.

Yoander Cabanes Risco

_____

Firma del autor

José Leandro González González

_____

Firma del autor

MSc. Gilberto Arias Naranjo

_____

Firma de la tutora

MSc. Tomás Gámez Acosta

_____

Firma del tutor

Ing. Lesley Méndez Cáceres

_____

Firma del tutor

Hay una fuerza motriz más poderosa que el vapor,
la electricidad y la energía atómica:
la voluntad.

Albert Einstein

DATOS DEL CONTACTO

Tutores:

MSc. Gilberto Arias Naranjo
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: gilbertoa@uci.cu

MSc. Tomás Gámez Acosta
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: tfgam@uci.cu

Ing. Lesley Méndez Cáceres
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: lcaceres@uci.cu

AGRADECIMIENTOS

A nuestros padres, por su constante apoyo y dedicación; por su ejemplo de superación incansable; por lo que hemos sido y seremos.

A nuestro tutor Gilberto, por brindarnos de forma desinteresada sus conocimientos y ayudarnos a lograr este sueño.

A nuestros profes Vilmavis, Julio, Noel y demás personas que tuvieron que ver de una forma u otra con este trabajo y nuestra formación personal y profesional.

A Marisleidys, Yadir y Maykel, Manolo y Manolita, Hermes, en fin, a todo el equipo del proyecto Graph-Tool por su apoyo y preocupación, ustedes son parte de esto.

A nuestras amistades por los buenos momentos que pasamos y los recuerdos que hoy llevamos con nosotros.

A nuestro Comandante en Jefe Fidel y a la Revolución Cubana por darnos la oportunidad de estudiar en una universidad de excelencia.

DEDICATORIAS

A mis dos queridas madres del alma, Maida y Ana Luisa, por todo su amor, cariño y dedicación que me han sabido inculcar, por ser protagonistas insignias de lo que he logrado ser hoy.

A mis abuelos Mariano y José, por su apoyo y constancia en todo momento, por comportarse como mis verdaderos padres, los quiero y respeto mucho.

A mi tía Edelma y mi primo Carlos Alberto, por confiar en mí y estimularme al sacrificio constante por lograr toda esta gran meta propuesta.

A mi tío José Luis que en paz descanse, que aunque no esté presente físicamente si lo está en mi mente.

A Ariadna por ocupar un lugar muy especial en mi vida.

A mis grandes amigos Manuel y Dionnys, por estar siempre a mi lado en los buenos y malos momentos, ustedes saben que son mis hermanos.

A todos mis amigos de forma general.

A todos gracias por existir.

Yoander

A mi mamá María Magdalena, por ser la estrella que más brilla en mi corazón, por brindarme su amor ágape en todos los momentos de mi vida. Te amo.

A mi papá Leandro, por ser un amigo, un ejemplo de padre y hombre que aspiro ser algún día, gracias por quererme como lo haces, sabes que te quiero de la misma forma aunque no te lo diga.

A mi hermano Leonel, por formar parte de mis mayores secretos, sabes que eres el mayor protagonista de este sueño.

A Yadira, mi cuñada del alma.

A mis grandes amigos, que no valdría la pena entrar en particularidades ya que sería demasiada extensa esta lista. A todos, gracias por brindarme ese granito de fe que me mantiene vivo.

A mi novia por su amor y comprensión en los momentos que más lo necesité.

A mi familia toda, muchas gracias por existir.

José Leandro

RESUMEN

El presente trabajo surge producto de una investigación de colaboración conjunta entre el Centro Nacional de Bioinformática y la Facultad 6 de la Universidad de las Ciencias Informáticas, debido a que en la actualidad nacional existen muy pocos trabajos que validen la aplicación de los Algoritmos Evolutivos en tareas de Minería de Datos. Por tal situación, la investigación está orientada a las Técnicas de Minería de Datos, aplicadas específicamente a la tarea de Clasificación. Para ello se realiza la propuesta de un algoritmo basado en Programación Genética para la generación de modelos formados por reglas de clasificación. Este algoritmo será capaz de buscar potenciales soluciones, el cual es validado mediante el desarrollo del ciclo completo de una herramienta informática. Esta herramienta, además de implementar el algoritmo, tiene la funcionalidad de clasificar nuevas instancias, de acuerdo a la base de reglas generadas, la misma está implementada de forma genérica, permitiendo que pueda ser utilizada por diferentes líneas de trabajo como ayuda en el proceso de toma de decisiones.

PALABRAS CLAVES: Proceso de extracción del conocimiento, Minería de Datos, Programación Genética, Clasificación, reglas de clasificación.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Durante los últimos años, la comunidad científica ha trabajado con grandes volúmenes de información que han ido almacenando en bases de datos. Esto no ha tenido efectos positivos del todo, ya que muchos científicos se quejan de la creciente complejidad que representa encontrar información útil en este "laberinto de datos". Para mejorar esta situación, se desarrollan técnicas que integran la información dispersa, gestionan bases de datos distribuidas, evalúan su calidad y facilitan su accesibilidad a los investigadores. Además, para realizar un análisis y procesamiento de toda esta información, se ha insertado a las computadoras como medio necesario y a la Minería de Datos como vía fundamental para la solución de estos problemas.

La Minería de Datos es actualmente un tema muy difundido, el cual engloba una serie de procesos y técnicas muy novedosas basadas en la Inteligencia Artificial y el Análisis Estadístico, encaminadas a la extracción de "conocimiento" procesable, nuevo y útil, implícito en grandes almacenes de datos y/o bases de datos. Para encontrar este conocimiento implícito en la información, esta rama se apoya en la realización de diferentes tareas, en dependencia del problema que se plantee, tales como: Asociación, Agrupamiento (clustering en ingles), Clasificación, entre otras, siendo esta última una de las más usadas. Como métodos de solución a estas tareas se encuentran los algoritmos estadísticos, redes neuronales, Algoritmos Evolutivos o algoritmos de búsqueda basados en reglas probabilísticas, entre otros, y estos últimos presentan variantes como los Algoritmos Genéticos y la Programación Genética. Dada la efectividad de las tareas antes mencionadas, se sugiere su aplicación en herramientas que apoyen las investigaciones realizadas en diferentes áreas científicas, ya que permiten procesar grandes bases de datos y son capaces de manejar indisolublemente datos de altas complejidades. [1]

Cuba desde hace varios años ha comenzado a incursionar en este gran mundo de la Minería de Datos, realizando estudios e investigaciones, fomentando de esta manera el desarrollo de herramientas y aplicaciones en diferentes esferas científicas. Ejemplo de ello, son las recientes investigaciones que, junto al Centro Nacional de Bioinformática, se desarrollan con el propósito de demostrar la efectividad de los Algoritmos Evolutivos en tareas de Minería de Datos.

Los paradigmas evolutivos actuales están inspirados en la teoría de la evolución de Darwin e intentan simular en lo posible los procesos que ocurren en la naturaleza, basando la resolución de problemas

computacionales en mecanismos que simulan la evolución biológica. [2]

Los Algoritmos Genéticos son uno de los tipos de Algoritmos Evolutivos más utilizados, no siendo así la Programación Genética, que cuenta con menos resultados probados que estos, pero puede encontrarse entre las técnicas más prometedoras.

El Centro Nacional de Bioinformática en conjunto con la Facultad 6 de la Universidad de las Ciencias Informáticas está llevando a cabo una investigación para demostrar la validez o comprobar la factibilidad del uso de la Programación Genética en tareas de Minería de Datos, sin embargo no existe una propuesta concreta para enfrentar, con esta técnica, la tarea de Clasificación.

Por todo lo antes planteado, se define como **Problema Científico**:
¿Cómo generar reglas de clasificación utilizando Programación Genética?

Siendo **Objeto de Estudio** de este trabajo:
Las Técnicas de Minería de Datos aplicadas a la tarea de Clasificación.

A partir del objeto de estudio se delimita el siguiente **Campo de Acción**:
La Programación Genética en la tarea de Clasificación.

Teniendo en cuenta lo anterior y como solución al problema antes enunciado, se plantea como **Objetivo General:**
Desarrollar una herramienta informática para la generación de reglas de clasificación mediante Programación Genética.

Para cumplir este objetivo general se trazaron los siguientes **Objetivos Específicos**:

- Diseñar un algoritmo para la generación de reglas.
- Implementar el algoritmo diseñado.
- Identificar los requerimientos de la herramienta.
- Diseñar la herramienta.
- Implementar la herramienta diseñada.
- Probar la herramienta implementada.
- Analizar los resultados obtenidos.

Para alcanzar los objetivos trazados y dar solución al problema planteado se elaboraron las siguientes **Tareas**:

- Estudio de fases del proceso de descubrimiento de conocimiento en bases de datos y de la tarea de Minería de Datos: Clasificación.
- Estudio de Algoritmos Evolutivos, particularmente algoritmos de Programación Genética aplicados a tareas de Minería de Datos.
- Diseño de un algoritmo para la generación de reglas de clasificación basado en Programación Genética.
- Implementación del algoritmo diseñado.
- Definición de los requisitos funcionales y no funcionales de la herramienta.
- Definición de los Casos de Uso del Sistema.
- Elaboración del Diagrama de Casos de Uso del Sistema.
- Descripción de los Casos de Uso correspondientes al sistema.
- Elaboración de los Diagramas de Clases del Diseño.
- Implementación de los Casos de Uso del Sistema.
- Desarrollo de una estrategia de prueba sobre el sistema.
- Análisis comparativo de los resultados obtenidos.

Finalmente el trabajo de diploma está estructurado de la siguiente manera: introducción, tres capítulos principales que constituyen el cuerpo del mismo, conclusiones, recomendaciones, referencias bibliográficas, bibliografía y glosario de términos. A continuación se da un breve resumen de los capítulos.

En el **Capítulo 1 Fundamentación Teórica:** se hace un análisis del estado del arte del objeto de estudio, se investiga acerca de las herramientas vinculadas al campo de acción y se fundamentan las metodologías, tecnologías y herramientas utilizadas para el desarrollo de la aplicación informática propuesta.

En el **Capítulo 2 Desarrollo Ingenieril de la herramienta propuesta:** se reflejan las características y funcionalidades del sistema a partir de los requisitos funcionales y no funcionales identificados, así como la agrupación de los mismos en términos de Casos de Uso del Sistema, donde se elaboran las realizaciones de cada uno de ellos tanto en descripciones textuales como en el Modelo de Diseño. Se presenta el Diagrama de Componentes como artefacto del Modelo de Implementación. Por último se le

realizan a la herramienta distintos niveles y tipos de pruebas para comprobar las funcionalidades de la misma así como la calidad de su implementación.

En el **Capítulo 3 Resultados y Discusión:** se realiza la descripción detallada del algoritmo implementado en la herramienta así como su estructura general. Se describe la herramienta haciendo mención de sus principales características. Se estudian los resultados obtenidos realizando corridas con diferentes fuentes de datos y comparándolos con otros Algoritmos Evolutivos existentes.

CAPÍTULO 1
FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

El uso de Algoritmos Evolutivos como técnica de Minería de Datos es una de las tendencias actuales que más se está explotando para el descubrimiento de reglas de clasificación, en las grandes bases de datos con que se cuenta actualmente.

En el presente capítulo se aborda acerca del Descubrimiento de Conocimiento en Bases de Datos, la Minería de Datos y unas de sus más populares tareas: Clasificación. Se hace un análisis de los Algoritmos Evolutivos y dentro de ellos específicamente de la Programación Genética. Se realiza un estudio profundo de las tecnologías, tendencias y distintas aplicaciones que se utilizan para la solución del problema científico planteado.

1.2 Proceso de Descubrimiento de Conocimiento en Bases de Datos

El "Descubrimiento de Conocimiento en Bases de Datos", más conocido como KDD por sus siglas en inglés (Knowledge Discovery in Databases), se define como "el proceso no trivial de identificar patrones válidos, novedosos, potencialmente útiles y en última instancia, comprensibles a partir de los datos".

Se dice que estos patrones son [3]:

- Válidos: Cuando deben seguir siendo precisos para datos nuevos (con un cierto grado de certidumbre), y no sólo para aquellos que han sido utilizados en su obtención.
- Novedosos: Cuando aportan algo desconocido tanto para el sistema y preferiblemente para el usuario.
- Potencialmente útiles: Cuando la información que generan debe conducir a acciones que reporten algún tipo de beneficio para el usuario.
- Comprensibles: Pues la extracción de patrones no comprensibles dificulta o imposibilita su

interpretación, revisión, validación y uso en la toma de decisiones. De hecho, una información incomprensible no proporciona conocimiento (al menos desde el punto de vista de su utilidad).

Este proceso está formado por un conjunto de etapas o fases para la final obtención del conocimiento, estas quedan reflejadas en la Fig. 1 que se muestra a continuación.

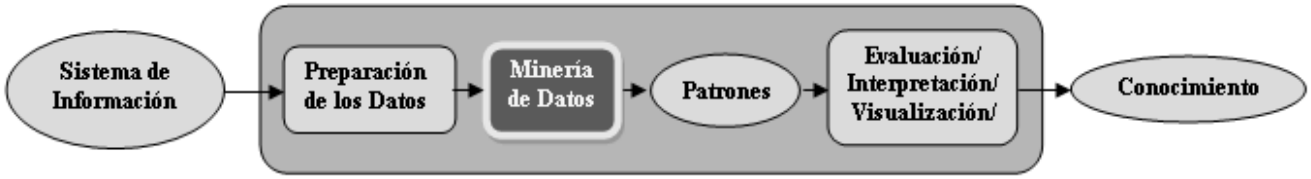

Fig. 1: Esquema que representa el KDD

1.3 Minería de Datos

Se define la Minería de Datos como el proceso de extraer conocimiento útil y comprensible, previamente desconocido, desde grandes cantidades de datos almacenados en distintos formatos. [3]

La Minería de Datos no surge como una nueva tecnología, sino que se crea, por la aparición de nuevas necesidades y principalmente, por el reconocimiento de un nuevo potencial: el valor, que generalmente está infrautilizado, de la gran cantidad de datos almacenados informáticamente en los sistemas de información de instituciones y empresas. Los datos pasan de ser un "producto" (el resultado histórico de los sistemas de información) a ser una "fuente de materia prima", que se explotan para obtener el verdadero producto elaborado, “el conocimiento”, el cual ha de ser especialmente valioso para la ayuda en la toma de decisiones sobre el ámbito en el que se han recopilado o extraído los datos.

En muchas situaciones, el método tradicional de convertir los datos en conocimiento consiste en un análisis e interpretación realizada de forma manual. El especialista en la materia, analiza los datos y elabora un informe o hipótesis que refleja las tendencias de los mismos. Esta forma de actuar es lenta, cara y altamente subjetiva, de hecho, el análisis manual es impracticable en dominios donde el volumen de los datos crece exponencialmente, debido a que la enorme abundancia de datos desborda la capacidad humana de comprenderlos sin la ayuda de herramientas potentes. Consecuentemente,

muchas decisiones importantes se realizan, no sobre la base de la gran cantidad de datos disponibles, sino siguiendo la propia intuición del usuario al no disponer de las herramientas necesarias. Éste es el principal cometido de la Minería de Datos: resolver problemas analizando los datos presentes en las bases de datos.

La Minería de Datos, al contrario del análisis de datos estadísticos tradicionales, trabaja sobre la totalidad de los datos y no con una muestra por lo que los resultados tienen una fiabilidad muy superior, lo que permite tomar decisiones con mucho menos riesgo de cometer errores. Esta posee dos retos fundamentales: por un lado, trabajar con grandes volúmenes de datos, y por el otro utilizar técnicas adecuadas para analizar los mismos y extraer conocimiento novedoso y útil.

Existen distintos tipos de tareas de Minería de Datos, cada una de las cuales puede considerarse como un tipo de problema a ser resuelto por un algoritmo de minería de datos, estas pueden ser:

Predictivas

- *Clasificación (o discriminación)*
- Clasificación suave
- Estimación de probabilidad de clasificación
- Categorización
- Preferencias o priorización
- Regresión

.

Descriptivas

- Agrupamiento (clustering)
- Correlaciones y factorizaciones
- Reglas de asociación
- Dependencias funcionales
- Detección de valores e instancias anómalas

Cada una de las tareas anteriores requiere de métodos, técnicas o algoritmos para ser resueltas. Una tarea puede tener muchos métodos diferentes que la resuelva y también se tiene que el mismo método (o al menos el mismo tipo de técnica) puede resolver un gran conjunto de tareas.

A continuación se enuncian algunos tipos de técnicas existentes para llevar a cabo las tareas anteriores.

Técnicas o Métodos:

- Algebraicas y estadísticas
- Bayesianas
- Conteos de frecuencias y tablas de contingencia
- Árboles de decisión y sistemas de aprendizaje de reglas
- Relacionales, declarativas y estructurales
- Redes neuronales artificiales
- Basadas en núcleo y máquinas de soporte vectorial
- Técnicas difusas y estocásticas (*Algoritmos Evolutivos*)

1.3.1 Tarea de Clasificación

Este trabajo se enfoca principalmente en el estudio de una de las tareas clásicas de la Minería de Datos:

Clasificación: Esta es quizás la tarea más utilizada en la Minería de Datos. En ella, cada instancia (o registro de la base de datos) pertenece a una clase, la cual se indica mediante el valor de un atributo que llamamos la clase de la instancia. Este atributo puede tomar diferentes valores discretos, cada uno de los cuales corresponde a una clase. El resto de los atributos de la instancia (los relevantes a la clase) se utilizan para predecir la clase. El objetivo es predecir la clase de nuevas instancias de las que se desconoce la clase. Más concretamente, el objetivo del algoritmo es maximizar la razón de precisión de la clasificación de las nuevas instancias, la cual se calcula como el cociente entre las predicciones correctas y el número total de predicciones (correctas e incorrectas). Como es una tarea predictiva, su fin es predecir uno o más valores para uno o más ejemplos. Estos ejemplos, que frecuentemente se dividen en dos, para prueba y para entrenamiento, se presentan como un conjunto de pares de elementos de dos conjuntos (pares etiquetados), $\delta = \{<e, s>: e \in E, s \in S\}$, donde S es el conjunto de valores de salida y E el conjunto de valores de entrada. El objetivo es aprender una función $\lambda: E \rightarrow S$, denominada clasificador, que represente la correspondencia existente en los ejemplos, es decir, para cada valor de E se tiene un único valor para S. [3]

1.3.2 Formas de representación del conocimiento

La tarea más importante a la hora de desarrollar un sistema basado en el conocimiento consiste en la modelización del problema que se quiere resolver, esto es, representar el conocimiento que posee el experto en un lenguaje que permita razonar al sistema. Consiste en formalizar y estructurar los objetos y sus relaciones en la base de conocimiento.

Existen dos tipos de representación fundamentales [4]:

- Métodos procedurales ("saber como"): El conocimiento está descrito mediante un conjunto de procedimientos que permiten resolver un problema.
- Métodos declarativos ("saber que"): El conocimiento se representa mediante un conjunto de sentencias junto con unos procedimientos generales que las manipulan.

A continuación se reflejan algunos ejemplos de representación del conocimiento para distintos tipos de problemas:

- Lógica de predicados
- Redes Semánticas
- Guiones
- Objetos
- Marcos
- Árboles
- Tablas
- *Reglas de producción*

De la misma forma que no existe ningún lenguaje de programación universal, tampoco se ha encontrado un lenguaje ideal para poder representar el conocimiento formalmente. La elección dependerá, principalmente, de la naturaleza y magnitud del problema.

Existe una estrecha relación entre las reglas de producción y la tarea de Minería de Datos que se aborda en este trabajo, Clasificación, ya que tienen generalmente la siguiente forma: SI Premisa ENTONCES Consecuencia, esto refleja el gran poder de expresividad, claridad y facilidad de

entendimiento del conocimiento aprendido. Por otra parte ofrecen una gran facilidad para la creación y la modificación de la base de conocimiento.

Existen diferentes tipos de reglas de producción y cada una es aplicada a un tipo específico de problema, para el caso de Clasificación se resaltan las reglas de: asociación, predicción, con excepciones, *clasificación,* entre otras.

1.3.2.1 Reglas de clasificación

En las reglas de clasificación el antecedente está formado por una serie de condiciones que debe cumplir un objeto para que se considere que pertenece a la clase indicada en el consecuente, estas pueden ser descritas como se muestra a continuación según el formato concreto de reglas en Extended Backus-Naur Form (EBNF):

<reglaClasificacion>	→	"**IF**" (<antecedente>) "**THEN**" <consecuente>
<antecedente>	→	<condicion> | <condicion> <operadorLogico> <antecedente>
<operadorLogico>	→	"**AND**" | "**OR**" | "**NOT**"
<consecuente>	→	atributoClase "=" etiquetaClase
<condicion>	→	<atributo> < operadorAritmetico > <valor>
<atributo>	→	Cada uno de los atributos del conjunto
<valor>	→	Valor del dominio correspondiente
<operadorAritmetico>	→	"=" | "<>" | "<=" | ">="

1.3.3 Algoritmos evolutivos

La Computación Evolutiva es un área de investigación dentro de la ciencia de la computación que trata de resolver problemas inspirándose en la teoría de la evolución de Darwin. Existen diferentes paradigmas dentro de esta, como son los Algoritmos Evolutivos que son un conjunto amplio de algoritmos de optimización estocástico, que se basan en los procesos biológicos permitiendo a las poblaciones de organismos adaptarse a su entorno y además se consideran como algoritmos de búsqueda de soluciones.

La tarea del descubrimiento de reglas ha sido abordada desde diversos paradigmas: construcción de árboles de decisión, aprendizaje inductivo, aprendizaje basado en instancias y, más reciente, redes neuronales y Algoritmos Evolutivos. El tipo de búsqueda que realizan cada uno de estos algoritmos va a determinar dónde se encuentran localizados dentro del panorama de la minería de reglas y desde el punto de vista de la minuciosidad de la búsqueda.

En la Fig. 2 se muestran tres de los algoritmos de inducción de reglas existentes. Primeramente mediante árboles de decisión, que utilizan heurísticas altamente voraces y realizan una búsqueda irrevocable, son muy rápidos y efectivos para encontrar clasificadores precisos, además de clasificar completamente los datos. Luego se encuentran los algoritmos convencionales de aprendizaje de reglas, que abarcan una amplia variedad cuya característica común es la de ser más minuciosos que los anteriores. Finalmente tenemos los Algoritmos Evolutivos, que son capaces de conducir muchas búsquedas minuciosas y realizar un retroceso implícito en la búsqueda del espacio de reglas que va a permitir encontrar interacciones complejas que los otros tipos de algoritmos no son capaces de hallar. [5]

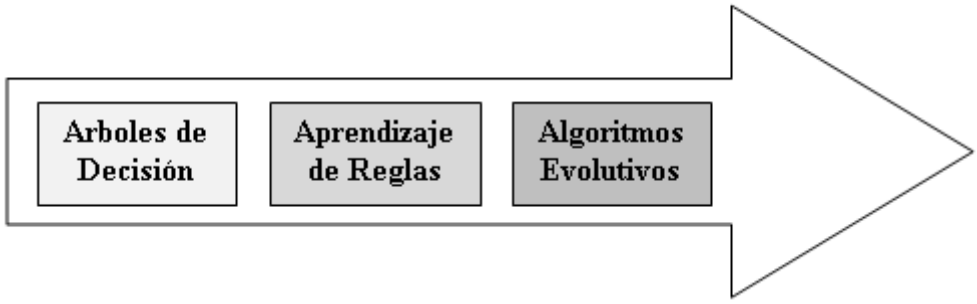

Fig. 2: Minuciosidad de búsqueda de los algoritmos de descubrimiento de reglas

Existen tres características básicas comunes en todos los Algoritmos Evolutivos : trabajan con una población de individuos (soluciones candidatas) a la vez, generan nuevos individuos por medio de un mecanismo de herencia a través de operadores estocásticos (los más usados son la recombinación o cruce y la mutación) aplicados a individuos de la generación actual y utilizan un método de selección determinado en base al ajuste del individuo y una medida de calidad que determina hasta qué punto es buena una solución.

Existen una gran variedad de Algoritmos Evolutivos que han sido propuestos pero principalmente pueden clasificarse en: Programación Evolutiva, Estrategias Evolutivas, Sistemas Clasificadores, Algoritmos Genéticos y *Programación Genética*. En este último se enfoca nuestro campo de acción.

1.3.4 Programación genética

La Programación Genética es una técnica que fue creada por John Koza [6] a finales de los años 80, que consiste en la evolución automática de programas usando ideas basadas en la selección natural de Darwin. No sólo se ha utilizado para generar programas, sino cualquier otro tipo de soluciones cuya estructura sea similar a la de un programa. Por ejemplo, fórmulas matemáticas, circuitos electrónicos, entre otros.

Según algunos autores "...la meta de la Programación Genética es lograr que las computadoras aprendan a resolver problemas sin ser explícitamente programadas, generando soluciones a problemas a partir de la inducción de programas. El programador no especifica el tamaño, forma y complejidad estructural de los programas-solución, sino que los programas evolucionan hasta generar soluciones satisfactorias...". [7]

En esencia se busca que poblaciones de programas evolucionen transmitiendo su herencia de manera que se adapten mejor al medio. Los mejores individuos tienen mayores probabilidades de reproducirse. La medida de calidad del individuo depende del tipo de problema. El espacio de búsqueda en la Programación Genética es el espacio de todos los posibles programas de computador compuestos de funciones y terminales apropiados al dominio del problema. Las funciones pueden ser operaciones aritméticas, operaciones de programación, funciones matemáticas, funciones lógicas, o funciones específicas del dominio.

Algunas características de la Programación Genética son:

- ✓ Material genético
  - no lineal y generalmente estructurado en árbol.
  - de longitud variable.
  - ejecutable.
- ✓ Cruce que mantiene la sintaxis.

En la aplicación de la Programación Genética a un problema, hay cinco pasos preparatorios importantes a definir [8]:

- El conjunto de terminales.
- El conjunto de funciones primitivas.
- La medida de la aptitud.
- Los parámetros para controlar la corrida.
- El método para designar un resultado y el criterio para terminar la ejecución del programa.

1.3.4.1 Pasos para la solución de problemas por Programación Genética

Como se ha comentado anteriormente, la Programación Genética engloba programas de computador para la solución de problemas, para esto es necesario ejecutar los siguientes pasos [8]:

- Generar una población inicial de composiciones aleatorias de funciones y terminales del problema (es decir, programas).
- Ejecutar iterativamente los siguientes pasos hasta que se satisfaga el criterio de terminación:
    - Ejecutar cada programa de la población y asignarle un valor de aptitud, de acuerdo a su comportamiento frente al problema.
    - Crear una nueva población de programas aplicando las siguientes tres operaciones primarias, a los programas escogidos, con una probabilidad basada en la aptitud.
        - Reproducir un programa existente copiándolo en la nueva población.
        - Crear dos programas a partir de dos programas existentes, realizando la combinación genética entre partes escogidas de los dos programas de forma aleatoria, usando la operación cruce aplicada a un punto de cruce, escogido aleatoriamente dentro de cada programa.
        - Crear un programa a partir de otro seleccionado aleatoriamente, cambiando aleatoriamente un gen (función o terminal). Es posible que se requiera un procedimiento de reparación para que se satisfaga la condición de clausura.
    - El programa identificado con la mayor aptitud (el mejor hasta la última generación), se designa como el resultado de la corrida de Programación Genética. Este resultado puede representar una solución (o una solución aproximada) al problema.

1.4 Búsqueda bibliográfica

Las experiencias y referencias tenidas en el campo de los Algoritmos Evolutivos en la Minería de Datos no son numerosas debido entre otras razones a la novedad de su uso. Algunas referencias encontradas son:

En [9] se describe la utilización de técnicas de Minería de Datos en sistemas de aprendizaje, presentando una herramienta visual para el descubrimiento de conocimiento en forma de reglas de predicción en la mejora de sistemas hipermedia adaptativos educativos basados en Web.

En [10] y [11] se presenta un análisis de los beneficios de la Programación Genética Gramatical para la tarea de Clasificación en la Minería de Datos, se compara esta técnica con algoritmos para la inducción de árboles de decisión y reglas, usando como criterio de comparación la precisión de los elementos clasificados.

En [12] se presenta un estudio para la obtención de un clasificador mediante el uso de la Programación Genética centrando el ámbito del problema en bases de datos etiquetadas, con información sobre la pertenencia de los puntos a clasificar a una clase concreta, enmarcándose por tanto la obtención del clasificador dentro de un aprendizaje supervisado.

En [13] se hace alusión a la Programación Genética, que es utilizada en el trabajo citado, como herramienta para construir un Sistema de Minería de Datos que permite definir un conjunto de patrones para clasificar los datos contenidos en una base de datos. Para realizar esta tarea se construyó una gramática, la cual es usada por la Programación Genética para construir las reglas que representan los patrones de datos contenidos en la base de datos. De esta forma se pueden agrupar los datos en clases, de manera que se simplifica el contenido de la base de datos en un conjunto de patrones informativos.

En la búsqueda de software que permitieran realizar tareas de Minería de Datos, fueron encontradas algunas herramientas entre las que se destacan DBMiner [14], Weka [15] y Keel [16]. Estos sistemas son de dominio público, un tanto populares por su entorno gráfico integrado y otra serie de características significativas. Un inconveniente que presentan estos tipos de herramientas es que son complejas de manejar para una persona no experta en Minería de Datos.

En el caso de Weka y DBMiner son herramientas que abarcan diferentes algoritmos basados en tareas de Minería de Datos muy populares en la comunidad científica, pero guardan poca relación con los Algoritmos Evolutivos vinculados a la tarea de Clasificación. Una característica especial de la herramienta Keel es que incursiona en los Algoritmos Evolutivos como los Algoritmos Genéticos y la Programación Genética, presentando de esta última un algoritmo para la generación de reglas de clasificación, el cual no está documentado imposibilitando su estudio en futuras investigaciones.

Además en [5] se desarrolla un trabajo para el Descubrimiento de Reglas de Predicción mediante una herramienta gráfica denominada EPRules, que presenta validaciones de Programación Genética aplicada a la tarea de Clasificación, pero tiene la limitante que es para un problema en particular y dicha herramienta es privada, no se tiene acceso a ella y mucho menos a su código, solo se hace mención a ciertas características del algoritmo de Programación Genética Gramatical planteado sin entrar en detalles.

Debido a todo lo anterior se evidencia entonces lo necesario que resulta una herramienta que implemente el algoritmo de Programación Genética propuesto para la generación de reglas de clasificación de una forma sencilla de interpretar los resultados o al menos para la demostración de cuán eficiente puede ser el uso de la Programación Genética en tareas de Minería de Datos.

1.5 Algoritmo de Programación Genética para Clasificación

En esta sesión se comenta con detalle cada uno de los componentes y pasos del algoritmo que se propone para la generación de reglas de clasificación usando Programación Genética, así como sus descripciones.

Se muestra la forma de representación de la solución y luego se explica cada uno de los distintos componentes de la Programación Genética como la función de adaptación, el mecanismo de mantenimiento de la diversidad, los operadores genéticos y los parámetros que controlan la corrida del algoritmo.

1.5.1 Representación de los individuos

Un factor de especial interés en el diseño de todo proceso de aprendizaje evolutivo de reglas es el esquema de representación usado para codificar cada una de las soluciones. Las diferentes propuestas evolutivas siguen dos enfoques distintos [17]:

- "Cromosoma = Base de Reglas" o enfoque Pittsburgh, y
- "Cromosoma = Regla".

En el primer enfoque, cada individuo representa por si solo una solución completa al codificar un conjunto de reglas, mientras que en el segundo, cada individuo representa una parte de la solución (una regla), por lo que la solución final se obtiene combinando varios individuos.

Para este segundo enfoque existen tres propuestas genéricas:

- Michigan, en el que cada individuo codifica una única regla. Este tipo de sistemas habitualmente se denominan sistemas clasificadores.
- IRL (Iterative Rule Learning), en el que el proceso evolutivo devuelve sólo la mejor regla aprendida y la solución final se forma mediante la unión de los individuos obtenidos en una serie de ejecuciones sucesivas.
- "Cooperativo-Competitivo", en el que la base de reglas se codifica en toda la población o bien en un subconjunto de esta.

En este trabajo se optó por seguir el enfoque Michigan, debido a ser usado frecuentemente en problemas de clasificación.

Cada individuo se expresa como una regla de clasificación, o sea del tipo Si-Entonces (IF-THEN), descrita en la sesión 1.4.1. En esta primera variante solo se consideran reglas en Forma Normal Disyuntiva, o sea, reglas que en su antecedente las condiciones estén relacionadas mediante el operador lógico "Y" (AND), por ejemplo:

IF A1 tiene valor V1 **AND** A2 tiene valor V2 **THEN** AC tiene valor VC

El conjunto de símbolos terminales estará formado por los nombres de los atributos y cierto número de valores tomados de los respectivos dominios sobre los que se definen los atributos. El conjunto de funciones o no terminales está formado por los operadores relacionales y lógicos. Los operadores relacionales incluidos en el conjunto de funciones dependen de los tipos de datos de los atributos, en este caso se trata con atributos categóricos (nominales), y se ha definido como operador relacional a utilizar solo el "=" (igual). Al tratarse de reglas en Forma Normal Disyuntiva, el único operador lógico empleado en este algoritmo es AND.

Como esta propuesta está basada en un algoritmo de Programación Genética, los individuos están representados por árboles, o sea el sistema de codificación es mediante un árbol binario donde los nodos internos están formados por los operadores ya antes mencionados (AND y "="), el nodo raíz (THEN) para combinar el antecedente y el consecuente de la regla, y los nodos hojas formados por los atributos y sus valores. En la Fig. 3 se muestra como quedaría codificada la regla del ejemplo anterior:

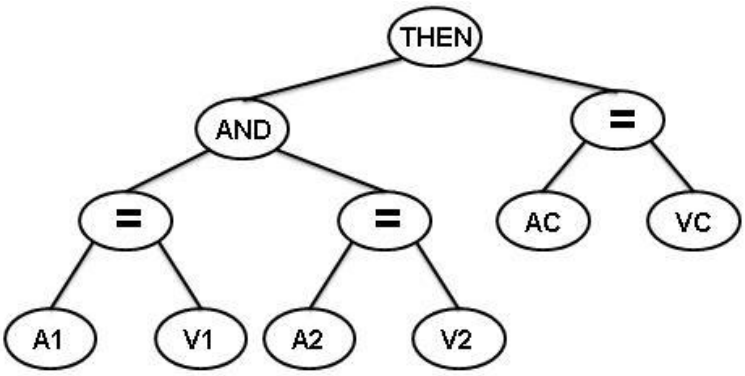

Fig. 3: Representación de un individuo para una regla de clasificación

1.5.2 Medidas de calidad de los individuos

Cada regla tiene asociada una medida de calidad, función de aptitud (fitness, en ingles) que es utilizada para la selección de los individuos mejores adaptados como futuros candidatos en las sucesivas generaciones de la evolución de la población. Esta medida de calidad puede ser calculada de diferentes formas, ya sea mediante el porcentaje de aciertos del antecedente y el consecuente de la regla con respecto a los datos, por el error cuadrático medio o error lineal medio, entre otros. Como se trata de un problema de clasificación se ha preferido usar medidas relacionadas con el primer criterio,

ya que estos tipos de problemas son difíciles de resolver usando medidas basadas en el error. Su mecanismo de cálculo está detallado en la sesión 3.1.

1.5.3 Mecanismo de mantenimiento de la diversidad

Algunos trabajos que usan Algoritmos Genéticos han diseñado varias propuestas para mantener la diversidad en la población, basándose en los dos siguientes principios:

1. Los padres se encuentran entre los individuos más similares a sus hijos.
2. El cálculo de alguna medida de similitud entre individuos.

Sin embargo, al usar estos principios con la Programación Genética se presentan problemas debido a que los padres y los hijos pueden ser totalmente diferentes por la naturaleza variable de los individuos (representación en forma de árbol). Es por esto, que en esta propuesta se usa la denominada Competición de Tokens, que no necesita tener en cuenta la estructura de los individuos. En [17] está bien detallado este proceso, pero en la sesión 3.2 se muestra un resumen de como se lleva a cabo en este trabajo.

1.5.4 Operadores Genéticos

Como ya se ha comentado en apartados anteriores, en Programación Genética se busca que poblaciones de programas evolucionen, transmitiendo su herencia de manera que se adapten mejor al medio. En este proceso de evolución de una población a otra para generar los descendientes, se emplean fundamentalmente los siguientes tres tipos de operadores genéticos:

- Cruzamiento: Este operador combina el material genético de individuos intercambiando partes de dos progenitores para producir dos descendientes. Ambas partes se eligen de forma aleatoria pero bajo la restricción de que los hijos generados sean válidos con la estructura de la regla descrita en la sesión 1.5.1.

A continuación se muestra una figura con un ejemplo de cómo cruzan dos individuos y se obtienen dos nuevos descendientes.

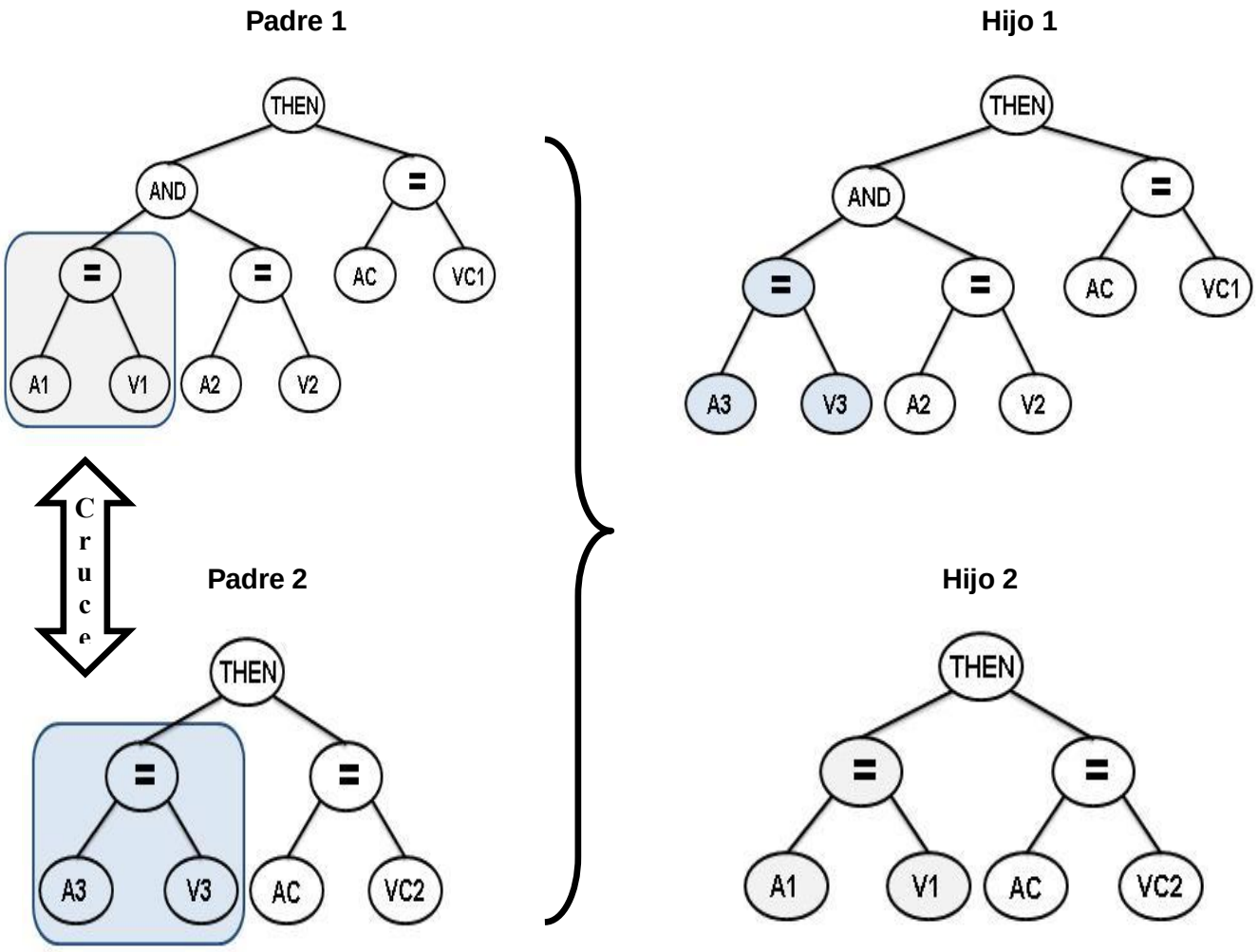

Fig. 4: Esquema de representación de un cruzamiento entre dos árboles

- Mutación: El objetivo es cambiar una parte de un árbol por otra generada aleatoriamente. Aquí también debe de cumplirse la validez del hijo obtenido con respecto a la estructura de la regla. Existen varios tipos de mutación como son la mutación puntual, permutación, levantamiento, mutación de subárbol, entre otros, en este trabajo se emplea esta última variante. En la siguiente figura se muestra un ejemplo de este operador.

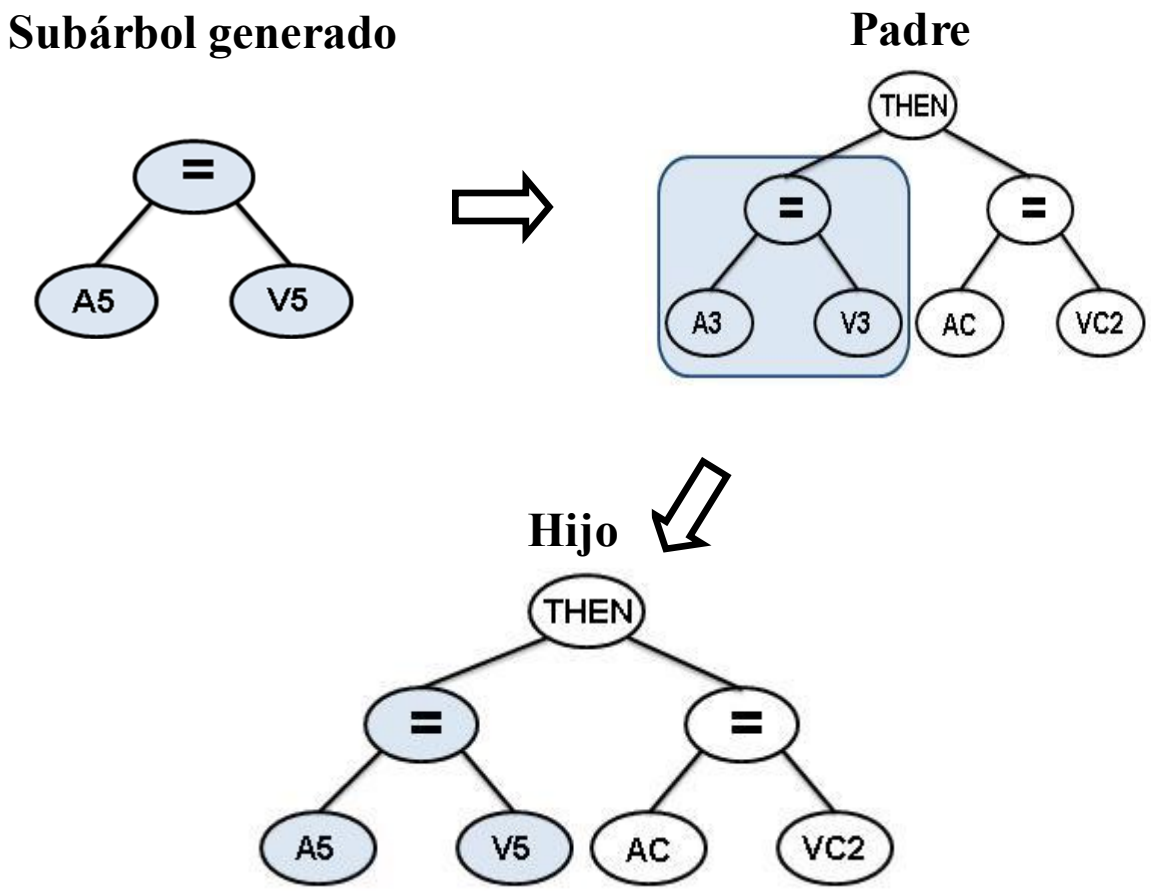

Fig. 5: Esquema de representación del operador genético mutación

- Reproducción: Este operador simplemente genera un descendiente de un progenitor con el mismo material genético. En la siguiente figura se puede observar este operador.

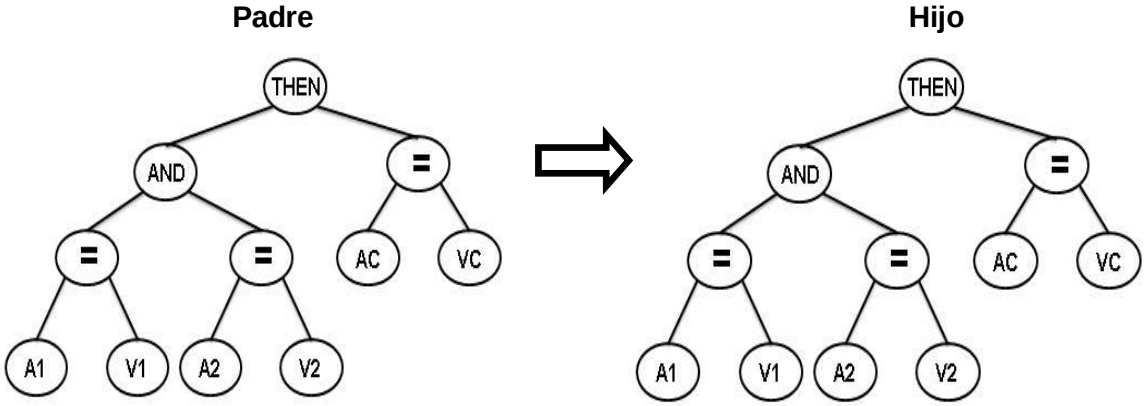

Fig. 6: Esquema de representación del operador genético reproducción

1.5.5 Método de selección

Existen diversos métodos de selección que pueden ser utilizados durante la elección de progenitores y de sobrevivientes. Algunos de ellos son [18]:

- Selección elitista**:** se garantiza la selección de los k miembros más aptos de cada generación.
- Selección proporcional a la aptitud**:** los individuos más aptos tienen más probabilidad de ser seleccionados, pero no la certeza.
- Selección por torneo**:** se eligen subgrupos de individuos de la población, y los miembros de cada subgrupo compiten entre ellos. Sólo se elige a un individuo de cada subgrupo para la reproducción.
- Selección por torneo binario**:** un caso particular del método de selección anterior, en el cual se eligen aleatoriamente k pares de individuos de la población base, y se constituye la muestra seleccionando el mejor de cada par.
- Selección por ruleta**:** una forma de selección proporcional a la aptitud en la que la probabilidad de que un individuo sea seleccionado es proporcional a la diferencia entre su aptitud y la de sus competidores. Conceptualmente, esto puede representarse como un juego de ruleta donde cada individuo obtiene una sección de la ruleta, pero los más aptos obtienen secciones mayores que las de los menos aptos. Luego la ruleta se hace girar, y en cada ocasión se elige al individuo que «posea» la sección en la que se asiente la bola que viaja por la ruleta.

La propuesta de selección de los individuos que se propone en este trabajo es esta última, o sea selección por ruleta.

1.5.6 Parámetros del Algoritmo Evolutivo

Los parámetros que controlan la ejecución del algoritmo son:

- Tamaño de la población.
- Número máximo de generaciones (este será el criterio de parada del algoritmo).
- Probabilidad de realizar cruzamiento.
- Probabilidad de realizar mutación.
- Probabilidad de realizar reproducción.

Estos valores pueden ser definidos por el usuario aunque tienen valores establecidos por defecto.

1.6 Tecnologías y herramientas a utilizar

Uno de los aspectos fundamentales en la elaboración de un software es seleccionar las tecnologías y herramientas que mayores beneficios aporten. Para llevar a cabo la implementación y documentación de este trabajo, se realizó un estudio de las mismas, teniéndose en cuenta las tendencias actuales, novedades y facilidades de cada una de ellas para llegar a la selección de las que se enuncian a continuación.

1.6.1 Metodología de desarrollo

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos de software. Indican paso a paso todas las actividades a realizar para lograr el producto informático deseado, además propone qué personas deben participar en el desarrollo de las actividades y qué papel deben jugar.

¿Qué metodología se debe usar para el desarrollo de esta aplicación?

OpenUP es una familia de plugging de procesos de fuente abierta. OpenUP/Basic es el proceso principal en OpenUP y es orientado hacia un equipo pequeño y organizado.

¿Qué es OpenUP/Basic?

OpenUP/Basic es un proceso de desarrollo de software de código abierto diseñado para pequeños equipos organizados quienes quieren tomar una aproximación ágil del desarrollo. OpenUP/Basic es un proceso iterativo que es mínimo, completo y extensible. Se valora la colaboración y el aporte de los stakeholders sobre los productos entregables y las formalidades innecesarias. Está organizado dentro de cuatro áreas principales de contenido: Comunicación y Colaboración, Intención, Solución y Administración.

OpenUP/Basic se caracteriza por cuatro principios básicos que se soportan mutuamente [19]:

- Colaboración para alinear los intereses y un entendimiento compartido.

- Balance para confrontar las prioridades (necesidades y costos técnicos).
- Se enfoca en articular la arquitectura para facilitar la colaboración técnica, reducir los riesgos y minimizar excesos y trabajo extra.
- Evolución continúa para reducir riesgos, demostrar resultados y obtener retroalimentación de los clientes

OpenUP/Basic está listo para ser usado; no se requiere adicionar o remover nada. Este puede ser extendido en grandes o pequeñas formas para adicionar nuevos contenidos de desarrollo o personalizar el proceso para su entorno específico. Además utiliza el Lenguaje Unificado de Modelado (Unified Modeling Language, UML) como lenguaje de notación.

Esta metodología tiene características perfectamente adaptables al trabajo en cuestión, no solo por la simplicidad de los procesos sino por los sólidos conceptos que la sustentan, posee gran flexibilidad y está creada específicamente para proyectos pequeños y de poca envergadura. Por todo lo anterior se ha determinado que esta es la metodología idónea para este tipo de trabajo bajo las condiciones del mismo.

1.6.2 Herramienta de modelado

Debido a la popularidad que ha ido adquiriendo UML en los últimos años, existen una gran variedad de herramientas CASE (Computer-Aided Software Engineering / Ingeniería de Software asistida por ordenador) que facilitan al ingeniero el modelado de cualquier proceso de desarrollo. Dentro de estas herramientas se encuentran Rational Rose, muy conocida mundialmente para los clientes de Windows, y Visual Paradigm, herramienta que está tomando auge en la Comunidad de Software Libre.

Visual Paradigm para UML (VP-UML) es una herramienta de modelado que apoya gran parte del ciclo de desarrollo del software. Se integra con los siguientes entornos de desarrollo:

- Eclipse/IBM WebSphere
- JBuilder
- NetBeans IDE
- Oracle JDeveloper
- BEA Weblogic
- Visual Studio
- IntelliJ IDEA

Permite modelar diferentes diagramas como: diagramas de clases, diagramas de interacción entre otros.

Entre sus principales características están [20]:

- Modelación de procesos del negocio
- Administración de requerimientos
- Generación de la capa Objeto-Relacional
- Generación de código e ingeniería inversa, incluye 10 lenguajes de programación, entre ellos: Java, C++, .NET, PHP y XML
- Generación de reportes PDF, MS Word, HTML
- Posee una interfaz de usuario amigable
- Es multiplataforma, disponible para los Sistemas Operativos Linux, Windows, y Mac OS

Por todo lo antes planteado se decide utilizar como herramienta de modelado para el desarrollo de la herramienta, Visual Paradigm.

1.6.3 Lenguaje de programación

Un lenguaje de programación puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Se clasifican según el nivel de abstracción (bajo, medio y alto), según la forma de ejecución (compilado e interpretado) y según el paradigma de programación (imperativo, funcional, lógico, orientado a objeto) que poseen cada uno de ellos.

En el proceso de implementación se decidió utilizar Java ya que es un lenguaje de programación de alto nivel y sigue el paradigma de programación orientado a objetos. A continuación se enuncian algunas de sus características que se consideran importantes en dicha elección.

Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems en 1991, empresa reconocida por sus estaciones de trabajo UNIX de alta calidad. Moldeado en C++, se diseñó para ser un lenguaje independiente de la plataforma y un entorno de ejecución (la Máquina Virtual de Java con siglas en ingles JVM) ligero y gratuito para las plataformas más populares de forma que los códigos binarios (bytecode) de las aplicaciones Java pudiesen ejecutarse en cualquier

plataforma. Este lenguaje tiene ventajas significativas con respecto a otros lenguajes y otros ambientes de programación que lo hacen adecuado para realizar cualquier tarea de programación.

Las ventajas más significativas para utilizar este lenguaje en este trabajo se enumeran a continuación [21]:

- Independiente de la plataforma: Tiene la capacidad de que el programa pueda trasladarse de un sistema computacional a otro sin necesidad de volver a escribirlo para que funcione.
- Orientado a objetos: Toma prestadas muchas ideas de entornos orientados a objetos de las últimas décadas. El modelo de objetos es sencillo y de fácil ampliación. Trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma orientado a objetos: encapsulación, herencia y polimorfismo.
- Sencillez: Característica que posibilita toda la funcionalidad de un lenguaje potente, pero sin las particularidades sofisticadas que provocan confusiones.
- Robusto: Para ganar confiabilidad, realiza verificaciones tanto en tiempo de compilación como en tiempo de ejecución y así consigue encontrar rápidamente los errores más comunes en el desarrollo del programa.
- Seguridad: El código en Java pasa por una serie de pruebas (un verificador de bytecodes) que comprueba el formato de los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de código ilegal, el cual viola derechos de acceso sobre objetos o intenta cambiar el tipo o clase de un objeto) antes de ejecutarse en una computadora.

Además de estas ventajas se consideró el uso de este lenguaje para la implementación del algoritmo propuesto dado que en el Centro Nacional de Bioinformática, lugar al que está vinculada esta tesis, se utiliza este lenguaje como base de sus aplicaciones. Por tanto el uso del mismo viene siendo un requerimiento solicitado por las partes interesadas.

1.6.4 Entorno de desarrollo

Las herramientas del tipo Entorno de Desarrollo Integrado (IDE del inglés Integrated Development Environment) consisten en un potente editor más un conjunto de herramientas añadidas que aumentan su potencia. Entre los más utilizados actualmente para la programación en Java, por las facilidades

que brindan y por ser de código abierto pueden mencionarse: Eclipse y NetBeans. Se consideró trabajar en el NetBeans IDE por una serie de aspectos que se detallan a continuación.

NetBeans

El NetBeans IDE es una herramienta OpenSource para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el IDE NetBeans. Es un producto libre y gratuito sin restricciones de uso.

Otras características de este IDE son:

- Desarrollo de aplicaciones multiplataforma sobre: MacOS, Windows, Linux.
- Dispone de todo un entorno para crear documentación (javadoc).
- Tiene editor de código sensible al contenido con soporte para autocompletar el código, coloreado de etiquetas, auto tabulación y uso de abreviaturas para varios lenguajes de programación.

La plataforma ofrece servicios comunes a las aplicaciones de escritorio, permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación. Entre las características de la plataforma están [22]:

- Administración de las interfaces de usuario (ej. menús y barras de herramientas).
- Administración de las configuraciones del usuario.
- Administración del almacenamiento (guardando y cargando cualquier tipo de dato).
- Administración de ventanas.
- Framework basado en asistentes (diálogo paso a paso).

Desde el punto de vista de creación de interface visual, NetBeans es una herramienta cómoda y práctica. Se consideró su uso también debido a que tanto los tutores como los autores del presente trabajo de diploma ya se habían iniciado en el aprendizaje del uso de este IDE.

1.7 Conclusiones

Con el desarrollo de este capítulo, como base para el entendimiento del tema en que se desenvolverá este trabajo, se describió brevemente el Proceso de Descubrimiento de Conocimiento en Bases de Datos y dentro de este la fase de Minería de Datos. Se enunciaron diferentes tareas de la Minería de Datos y específicamente características de la tarea de clasificación. Se mostró lo valioso y novedoso que son los Algoritmos Evolutivos y entre ellos la Programación Genética, así como sus principales características.

Después de un estudio detallado se puede llegar a la conclusión de que se realizara la implementación de un algoritmo de Programación Genética para la tarea de Clasificación. Para esto se hizo una descripción de las tendencias y tecnologías actuales para luego de un detallado y cuidadoso estudio, escoger el lenguaje de programación Java por las posibilidades multiplataforma que este brinda y su uso gratuito. Se ha seleccionado el NetBeans IDE como ambiente de desarrollo por sus grandes potencialidades. Se decidió optar por la Metodología de Desarrollo OpenUP/Basic por las características mencionadas anteriormente.

Capítulo 2
Desarrollo Ingenieril de la herramienta propuesta

2.1 Introducción

En el presente capítulo se define la especificación de los requerimientos del sistema, donde se presentan las características y funcionalidades del mismo a partir de los requisitos funcionales y no funcionales capturados. Además se identifica el actor que interactúa con la herramienta y se presenta el Diagrama de Casos de Uso del Sistema así como la realización de cada uno de ellos. Por otro lado, se representa el Diagrama de Componentes del sistema como parte del modelo de implementación elaborado. Por otro lado se reflejan de manera clara y explícita la estrategia de prueba trazada, así como los diferentes niveles y tipos de pruebas que se ejecutaron a la herramienta.

2.2 Especificación de Requerimientos del sistema

Todas las ideas que los clientes, usuarios y miembros del equipo de proyecto tengan acerca de lo que debe hacer el sistema, deben ser analizadas como candidatas a requerimientos. Los mismos se pueden clasificar en: Funcionales y No funcionales.

2.2.1 Requerimientos Funcionales

Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir, así como las tareas que debe de realizar el sistema para satisfacer las necesidades del usuario. A continuación se muestran los requerimientos funcionales identificados:

R.1: Gestionar fichero.
- R.1.1: Cargar fichero desde una ubicación especificada.
- R.1.2: Salvar en fichero las reglas generadas.

R.2: Obtener reglas de clasificación.
- R.2.1: Configurar parámetros de ejecución del algoritmo.

R.2.2: Generar reglas de clasificación.

R.2.3: Visualizar las reglas obtenidas del algoritmo ejecutado.

R.3: Realizar clasificación.

2.2.2 Requerimientos No Funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Representan las características del producto, así como las características para que este sea atractivo, confiable, usable y seguro.

De acuerdo a lo anterior planteado, se identificaron los siguientes Requisitos No Funcionales:

Software:

- Se debe disponer en las computadoras de Sistema Operativo Linux, Windows 95 o superior para la instalación de la aplicación, garantizando una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas.
- Se necesita tener instalada la Máquina Virtual de Java nombrada Java Runtime Enviroment (JRE) versión 1.5 o superior.

Hardware:

- Para el desarrollo y utilización del proyecto se requieren computadoras con Procesador Pentium 3 o superior, 256 MB de RAM o más y 50 MB de capacidad de disco duro.

Usabilidad:

- La herramienta debe poder ser utilizada por cualquier tipo de persona que posea conocimientos básicos en el manejo de la computadora, solo se necesita que posea conocimientos de reglas de clasificación de manera que pueda entender los resultados mostrados por la misma.

Soporte:

- Mantenimiento: La herramienta debe estar bien documentada de forma tal que el tiempo de mantenimiento sea mínimo en caso de necesitarse.
- Instalación: La instalación de la herramienta debe caracterizarse por su facilidad, claridad y sencillez.

Apariencia o interfaz externa:

- La herramienta debe estar diseñada con una interfaz amigable, de forma tal que el usuario navegue sin dificultad alguna.

Portabilidad:

- La herramienta puede ser ejecutada sobre los sistemas operativos Linux y Windows, por su característica de ser multiplataforma.

2.3 Definición de los Casos de Uso del Sistema

En este epígrafe se definen los actores que intervienen o interactúan con el sistema. Un actor es una entidad externa del sistema que participa o interactúa con cada Caso de Uso. O sea, es un rol de un usuario, que puede intercambiar información o puede ser un recipiente pasivo de información y representa a un ser humano, a un software o a una máquina que interactúa con el sistema.

2.3.1 Diagrama de Casos de Uso del Sistema

Un Diagrama de Casos de Uso del Sistema contiene actores, Casos de Uso del Sistema y las relaciones existentes entre los mismos. Este diagrama representa de forma gráfica cómo será concebido el sistema para una mejor comprensión además de que refleja cómo interactúan los usuarios con el sistema.

En el caso que se está tratando solo se identifica un actor, el Especialista, que es el usuario que interactúa con el sistema para validar el algoritmo implementado y así obtener reglas de clasificación.

Los casos de uso definidos son:

- Gestionar fichero.
- Obtener reglas de clasificación.
- Realizar clasificación.

En la siguiente figura se puede observar el Diagrama de Casos de Uso del Sistema.

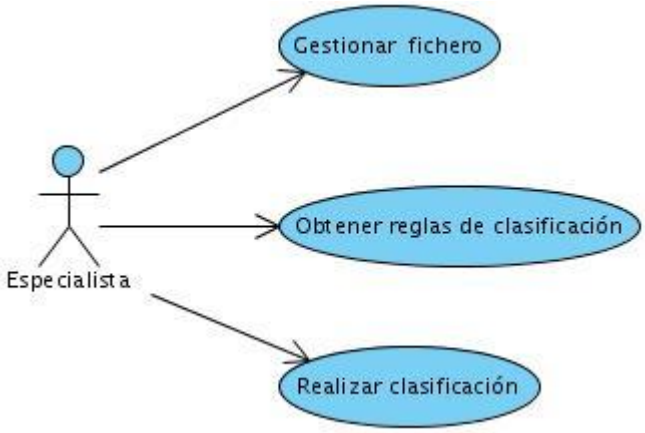

Fig. 7: Diagrama de Casos de Usos del Sistema

2.3.2 Descripción de los Casos de Uso del Sistema

Para entender la funcionalidad asociada a cada caso de uso no es suficiente con la representación gráfica del Diagrama de Casos de Uso, también se lleva a cabo la realización de los mismos elaborando las descripciones textuales de cada uno de ellos donde se especifican todas las acciones necesarias que realizan el actor y el sistema. Con las posteriores descripciones de los casos de uso, se obtendrá claramente la idea de cómo se realizarán las operaciones, cuándo y quienes intervienen en ellas.

Caso de Uso: Obtener reglas de clasificación

Caso de Uso:	Obtener reglas de clasificación
Actores:	Especialista (Inicia el CU)
Propósito	Generar reglas de clasificación mediante un algoritmo de Programación Genética.
Resumen:	El caso de uso se inicia cuando el Especialista selecciona la opción Obtener Reglas y termina cuando el sistema genera un conjunto de reglas por el algoritmo ejecutado.
Precondiciones:	Debe existir un fichero con extensión .arff que haya sido previamente cargado.
Referencias	R.2.1-2.3
Prioridad	Crítico.
Flujo Normal de Eventos	

Acción del Actor	Respuesta del Sistema
1. El caso de uso inicia cuando el Especialista selecciona la opción Obtener Reglas.	2. El sistema muestra una interfaz donde se indican los parámetros de control del algoritmo con sus valores predeterminados.
3. El Especialista indica Ejecutar Algoritmo.	4. El sistema ejecuta el algoritmo seleccionado.
	5. El caso de uso finaliza cuando el sistema muestra el conjunto de reglas generadas y otros parámetros asociados.
Flujos Alternos	
Acción del Actor	**Respuesta del sistema**
3. a) El Especialista modifica parámetros de control del algoritmo.	
3. b) Ir a la acción 3 del flujo normal de eventos.	
Prototipo de Interfaz	

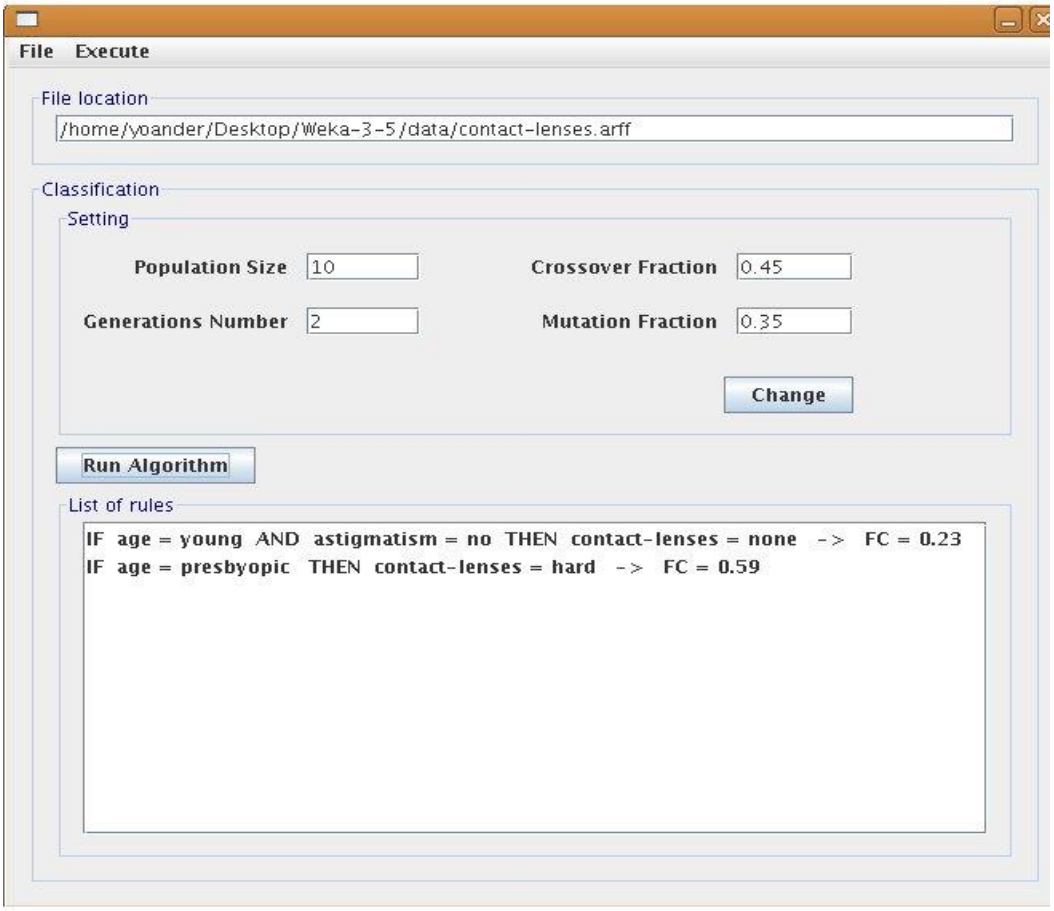

Fig. 8: Prototipo No Funcional CU Obtener reglas de clasificación

Poscondiciones	Se obtienen las reglas generadas por el algoritmo ejecutado.

Caso de Uso: Gestionar fichero

Caso de Uso:	Gestionar fichero
Actores:	Especialista (Inicia el CU)
Resumen:	El caso de uso se inicia cuando el Especialista hace referencia a una de las opciones Cargar o Salvar de la pestaña Archivo del menú principal y termina cuando el sistema realiza la acción seleccionada.
Precondiciones:	-
Referencias	R.1.1, R.1.2
Prioridad	Crítico.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**

1. El Especialista selecciona una de las siguientes acciones: • Cargar • Salvar	2. El sistema realiza una de las siguientes acciones: • Si se indicó Cargar, ver sección “Cargar fichero”. • Si se indicó Salvar, ver sección “Salvar fichero”.
Sección “Cargar fichero”	
Acción del Actor	**Respuesta del Sistema**
1. El Especialista indica localizar el fichero.	2. El sistema muestra la interfaz para localizar el fichero a cargar.
3. El Especialista selecciona el fichero a cargar	
4. El Especialista selecciona cargar el fichero.	5. El sistema verifica que la información contenida en el fichero sea correcta.
	6. Finaliza el caso de uso cuando el sistema carga el fichero y visualiza la ubicación del mismo.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	5. a) El sistema detecta que la información contenida en el fichero no es correcta.
	5. b) El sistema muestra un mensaje informando los errores detectados.
5 c) Ir a la acción 4 del flujo normal de eventos.	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
4. El caso de uso finaliza cuando el Especialista selecciona otra opción del sistema.	
Prototipo de Interfaz	
	

Fig. 9: Prototipo No Funcional CU Gestionar fichero

Poscondiciones	Se carga el fichero seleccionado.

Sección "Salvar fichero"	
Acción del Actor	**Respuesta del Sistema**
1. El Especialista selecciona salvar fichero.	2. EL sistema muestra una interfaz para localizar la ubicación del directorio donde se va a salvar.
3. El Especialista selecciona la ubicación.	
4. El Especialista indica salvar el fichero.	5. Finaliza el caso de uso cuando el sistema salva el fichero.

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
4. El caso de uso finaliza cuando el Especialista selecciona otra opción del sistema.	

Prototipos de Interfaz

Fig. 10: Prototipo No Funcional CU Gestionar fichero

Poscondiciones	Se salva en un fichero las reglas obtenidas.

Caso de Uso: Realizar clasificación

Caso de Uso:	Realizar clasificación
Actores:	Especialista (Inicia el CU)
Propósito	Predecir el comportamiento de atributos, clasificando a qué clase pertenece un determinado dato.
Resumen:	El caso de uso se inicia cuando el Especialista selecciona en el menú la opción de Clasificación y termina cuando el sistema clasifica la clase a la que pertenece el dato seleccionado.
Precondiciones:	Debe existir una base de reglas generadas producto de la corrida del

	algoritmo.
Referencias	R.3
Prioridad	Secundario.

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el Especialista selecciona la opción de Clasificar.	2 El sistema muestra una interfaz donde se indican los diferentes nombres de atributos de la nueva instancia a clasificar con sus respectivos valores.
3. El Especialista selecciona el valor de cada atributo de la nueva instancia a clasificar.	
4. El Especialista indica Clasificar dato.	5. El caso de uso finaliza cuando el sistema clasifica la clase a la que pertenece el dato solicitado.

Prototipos de Interfaz

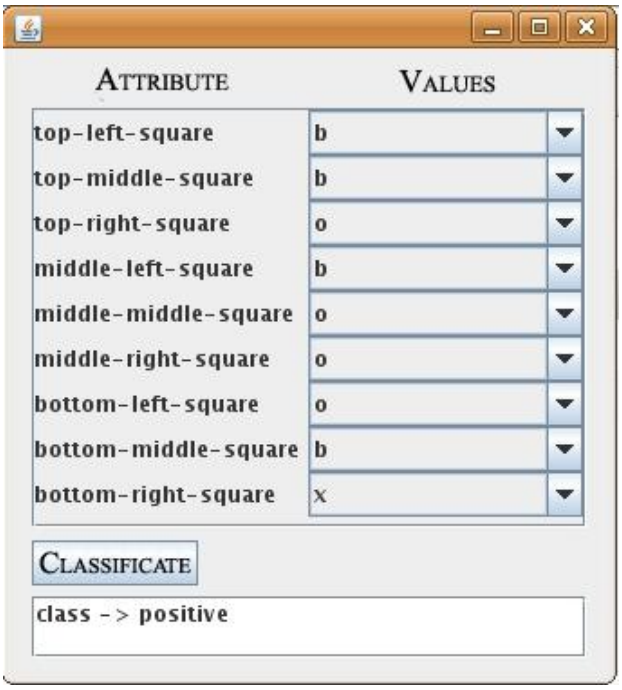

Fig. 11: Prototipo No Funcional CU Realizar clasificación

Poscondiciones	Se obtiene la clase a la que pertenece el dato solicitado.

2.4 Diseño de la herramienta

El objetivo de este flujo de trabajo es llevar los requisitos a un estado que queden listos para ser implementados en términos de programación, desarrollando un modelo fundamental que dará paso a una vista más aterrizada de la solución que se pretende, llevar un diseño enfocado a un entorno de programación.

En el diseño se reflejan tanto los requerimientos funcionales como los no funcionales, se define como van a ser desarrollados los mismos en un ambiente de programación, o sea ya se enfoca a un lenguaje específico y a una plataforma determinada.

2.4.1 Arquitectura implementada

La Arquitectura de Software consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema de información. Además establece los fundamentos para que analistas, diseñadores, programadores, etc. trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades. [23]

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias. [24]

Arquitectura en capas.

La programación por capas es un estilo de programación en la que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño, un ejemplo básico de esto es separar la capa de datos de la capa de presentación al usuario.

La ventaja principal de este estilo arquitectónico, es que el desarrollo se puede llevar a cabo en varios niveles o capas (presentación, negocio y acceso a datos) y en caso de algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado las tres capas.

Capa de presentación: Es la que ve el usuario (hay quien la denomina "capa de usuario"), presenta el sistema al usuario, le comunica la información y captura la información del usuario dando un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). Esta capa se comunica únicamente con la capa de negocio. También es conocida como interfaz grafica y debe tener la característica de ser amigable (entendible y fácil de usar) para el usuario. [25]

La capa de presentación está representada por la interfaz principal en la siguiente figura:

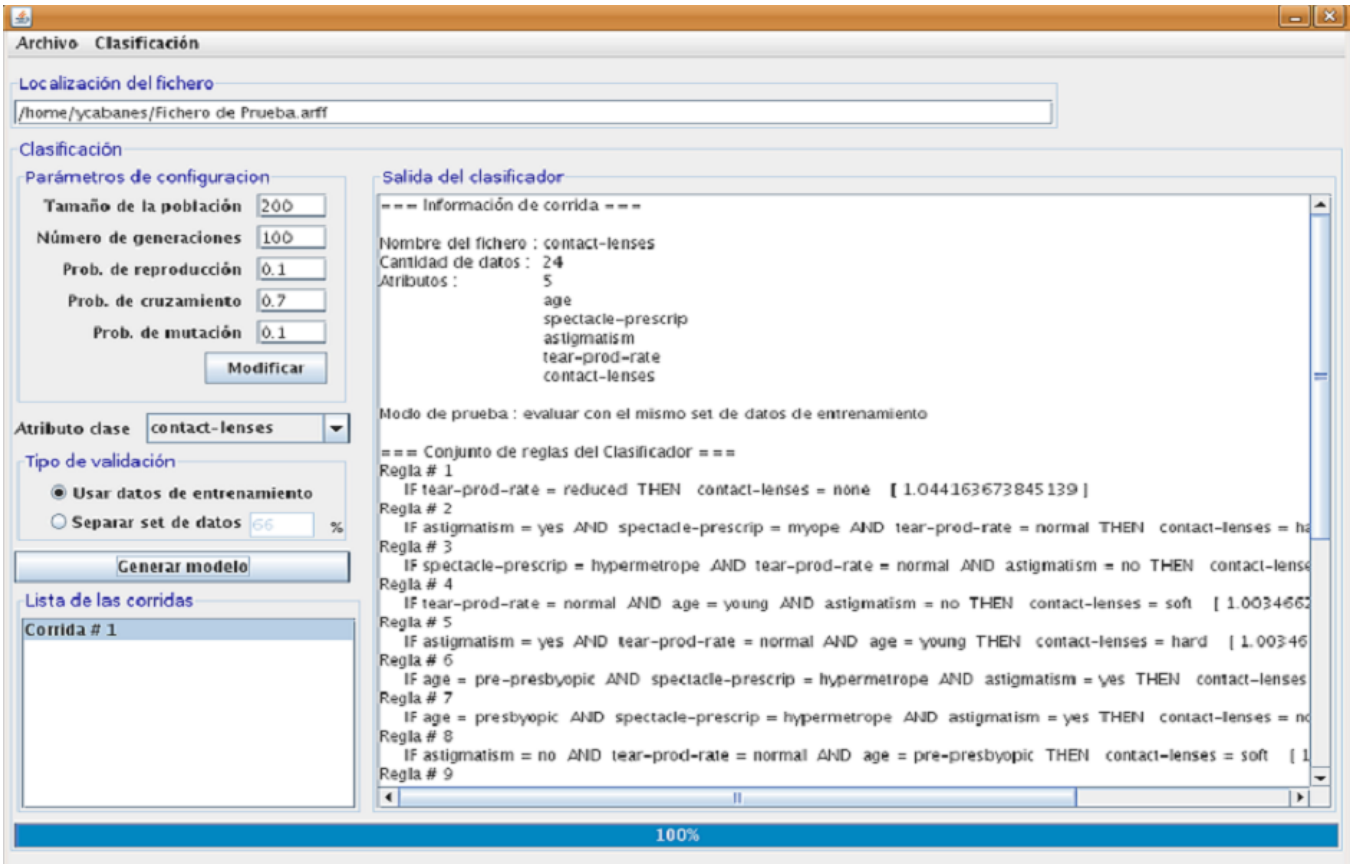

Fig. 12: Interfaz Principal

Capa de negocio: Es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (o lógica del negocio) pues es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos para almacenar o recuperar datos de él. [25]

La capa de negocio está representada por la Clase controladora: *GPController.java*

Capa de acceso a datos: Es donde residen los datos y es la encargada de acceder a los datos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio. Aunque lo más usual es que haya multitud de ordenadores donde se pueda procesar cada capa individualmente, estas capas pueden residir en un único ordenador. [25]

Esta capa está representada por los ficheros manejados así como las clases que mantienen información que será persistente: *ArffFile.java y GPRule.java*

2.4.2 Patrones de Diseño utilizados

Un patrón de diseño es una solución a un problema de diseño no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y reusable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias). Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios.

Los patrones que se utilizaron en el diseño de esta aplicación informática fueron los llamados patrones de asignación de responsabilidades (GRASP):

Experto: Propone asignar la responsabilidad a la clase que cuenta con la información necesaria para cumplir la responsabilidad. Permitiendo que se conserve el encapsulamiento, soportando un bajo acoplamiento y una alta cohesión, como se muestra en la siguiente figura:

Fig. 13: Subdiagrama donde se utiliza el patrón GRASP Experto

En este ejemplo la clase CC_GPRun cuenta con la información necesaria para modificar una población ya que contiene información referente a la clase CE_GPPopulation.

Creador: Propone asignarle a una clase la responsabilidad de crear los objetos de la otra en los casos de contener, agregar, registrar o utilizar. Brindando soporte de bajo acoplamiento, lo cual supone menos dependencias entre clases y posibilidades, [26] como se ilustra en la siguiente figura:

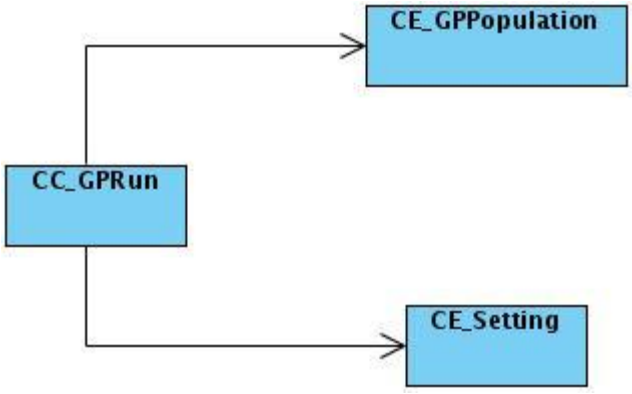

Fig. 14: Subdiagrama donde se utiliza el patrón GRASP Creador

En este caso la clase GP_Run tiene la responsabilidad de crear y actualizar las poblaciones y los parámetros del algoritmo.

Bajo Acoplamiento: Brinda como solución asignar responsabilidades de manera que las clases no dependan fuertemente de otras. Ofreciendo como beneficio que son fáciles de entender por separadas, fáciles de reutilizar y no se afectan por cambios de otros componentes. [26] La siguiente figura muestra un ejemplo:

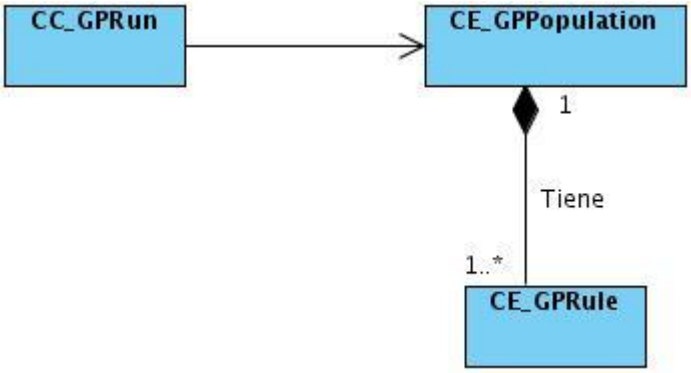

Fig. 15: Subdiagrama donde se utiliza el patrón GRASP Bajo Acoplamiento

En este caso estas clases no tienen relaciones innecesarias con otras clases.

Alta Cohesión: Este patrón propone asignar la responsabilidad de manera que la complejidad se mantenga dentro de límites manejables asumiendo solamente las responsabilidades que deben manejar, evadiendo un trabajo excesivo. Su utilización mejora la claridad y facilidad con que se

entiende el diseño, simplifica el mantenimiento y las mejoras de funcionalidad, generan un bajo acoplamiento, soporta mayor capacidad de reutilización. [26]

Esta clase, que parece en la figura que se muestra a continuación, contiene solamente la responsabilidad e información que ella debe manejar o sea todo las funcionalidades relacionadas con las reglas.

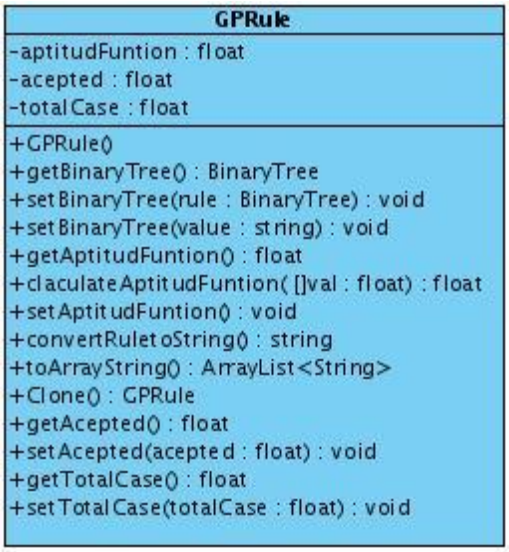

Fig. 16: Subdiagrama donde se utiliza el patrón GRASP Alta Cohesión

Controlador: Sugiere asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente un manejador de los eventos del sistema. Este patrón ofrece mayor potencial de los componentes reutilizables garantizando que los procesos de dominio sean manejados por la capa de los objetos del dominio. [26] Ejemplo de esto está lo refleja la siguiente figura.

Fig. 17: Subdiagrama donde se utiliza el patrón GRASP Controlador

En esta ocasión la clase GP_Run es la que maneja todos los eventos y funcionalidades relacionadas con las negociaciones asociadas a la clase GP_Population.

Los patrones de diseño son empleados en muchos desarrollos de Software ya que ofrecen una mejor práctica entorno al problema que se intenta resolver, de alguna manera son las experiencias continuas que han resultado al enfrentarse a determinado problema.

2.4.2 Diagrama de Clases del Diseño

Un diagrama de clases del diseño, muestra las clases del diseño enfocadas a un lenguaje en específico, ya en este diagrama se manejan especificaciones más técnicas y detalladas. Son la primicia para entender lo que posteriormente se va a implementar. Es un modelo de objeto que describe la realización física de los casos de uso. A continuación, en las siguientes tres figuras, se muestran los diagramas de clases del diseño de forma simplificada de cada caso de uso.

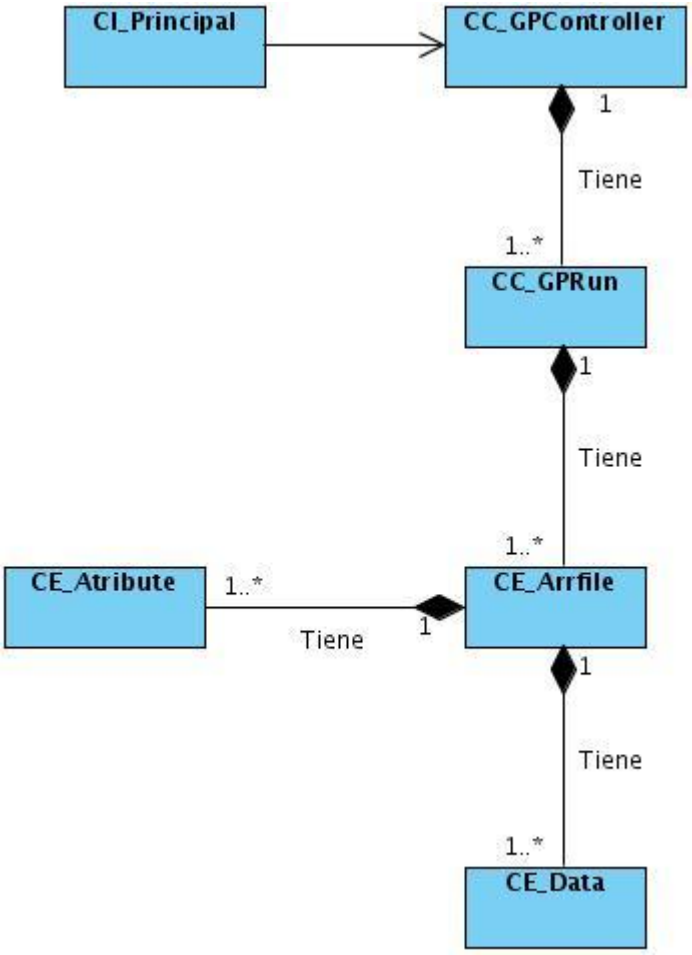

Fig. 18: Diagrama de Clases del Diseño del CU Gestionar fichero

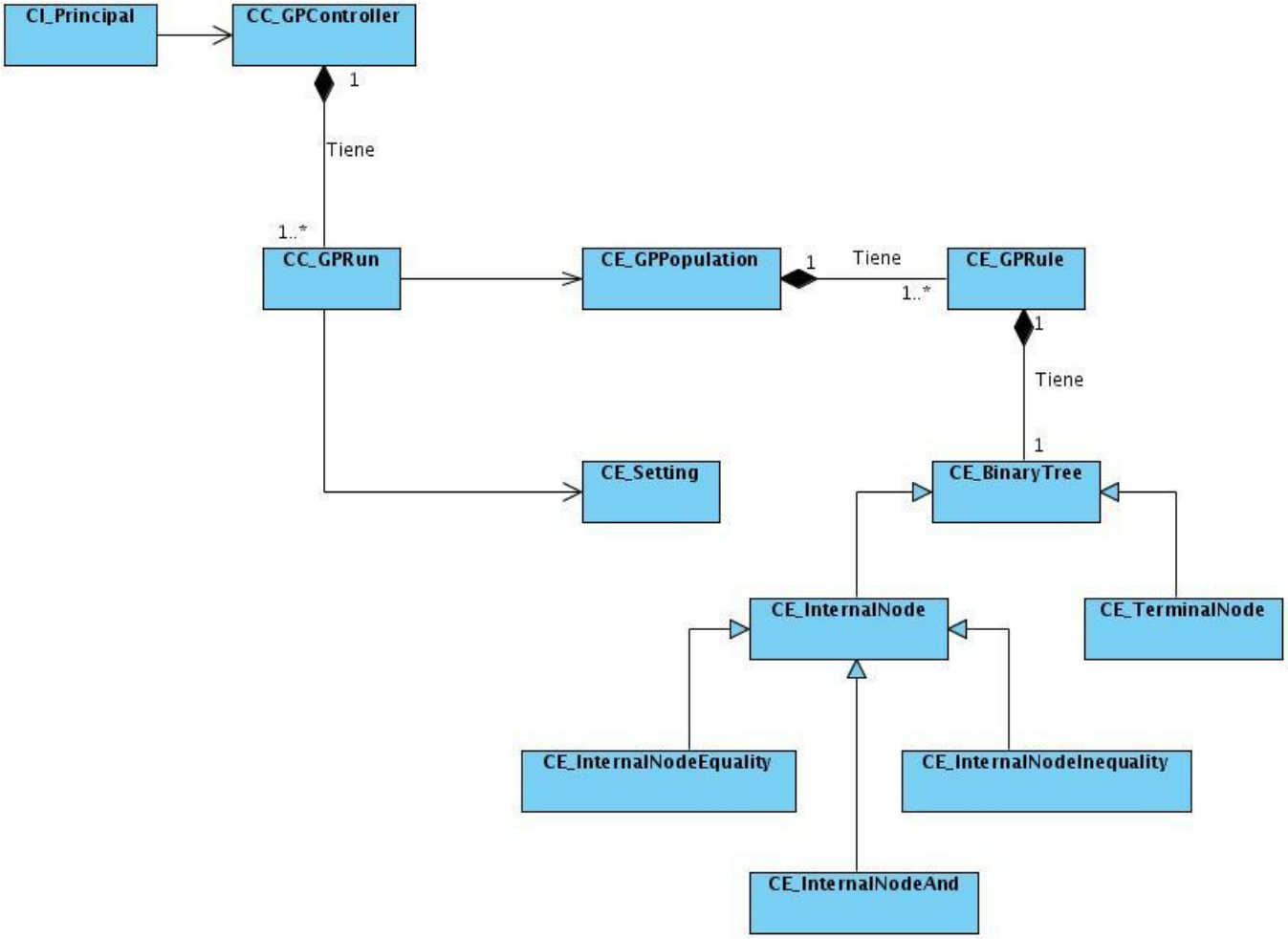

Fig. 19: Diagrama de Clases del Diseño del CU Obtener reglas de clasificación.

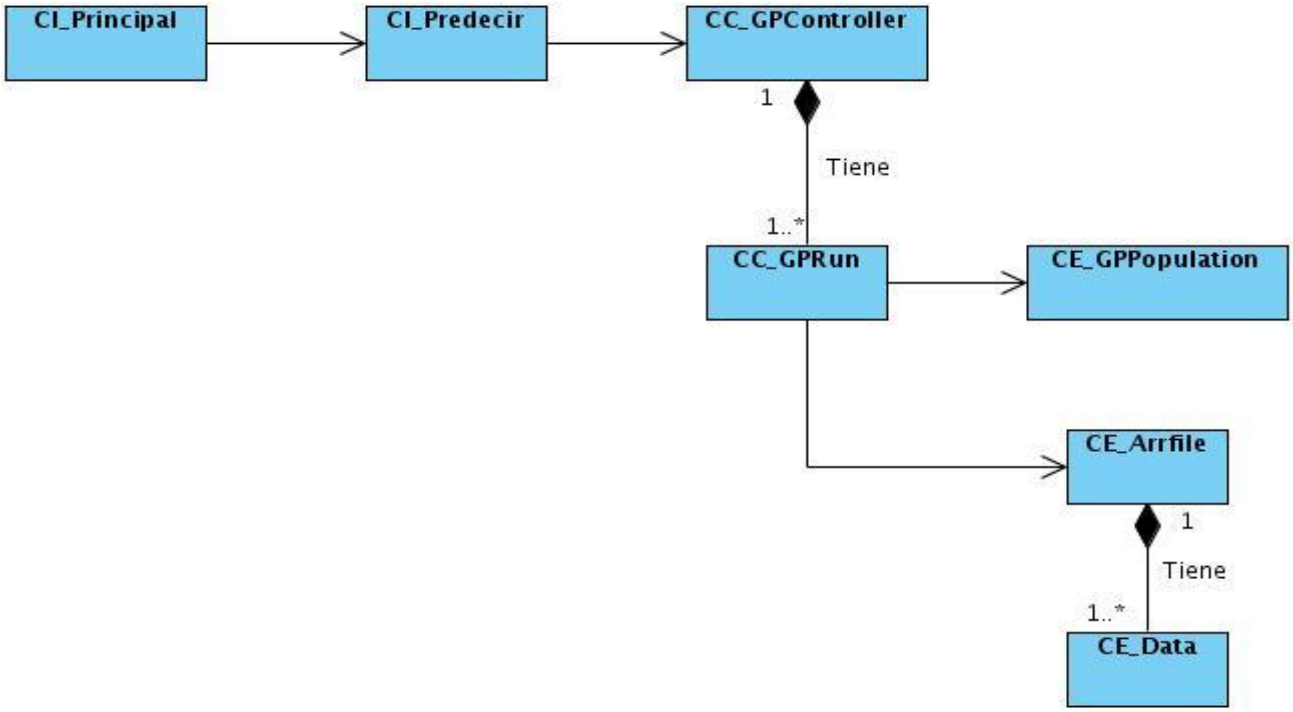

Fig. 20: Diagrama de Clases del Diseño del CU Realizar clasificación

2.4.3 Diagramas de secuencias

Como parte de la realización de los casos de uso se elaboran los diagramas de interacción (secuencia) correspondientes a cada escenario, los cuales muestran la secuencia de interacciones entre las clases mediante mensajes. De forma general reflejan la lógica que debe seguirse en cada solución específica. En las próximas cuatro figuras se reflejan los Diagrama de Secuencia correspondiente a cada escenario de cada caso de uso.

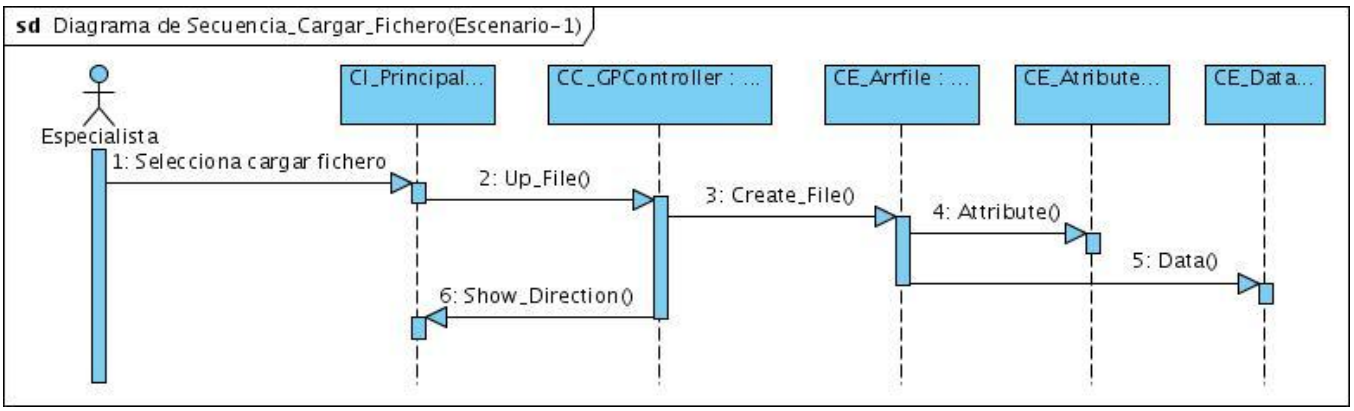

Fig. 21: Diagrama Secuencia del CU Gestionar fichero, escenario Cargar fichero

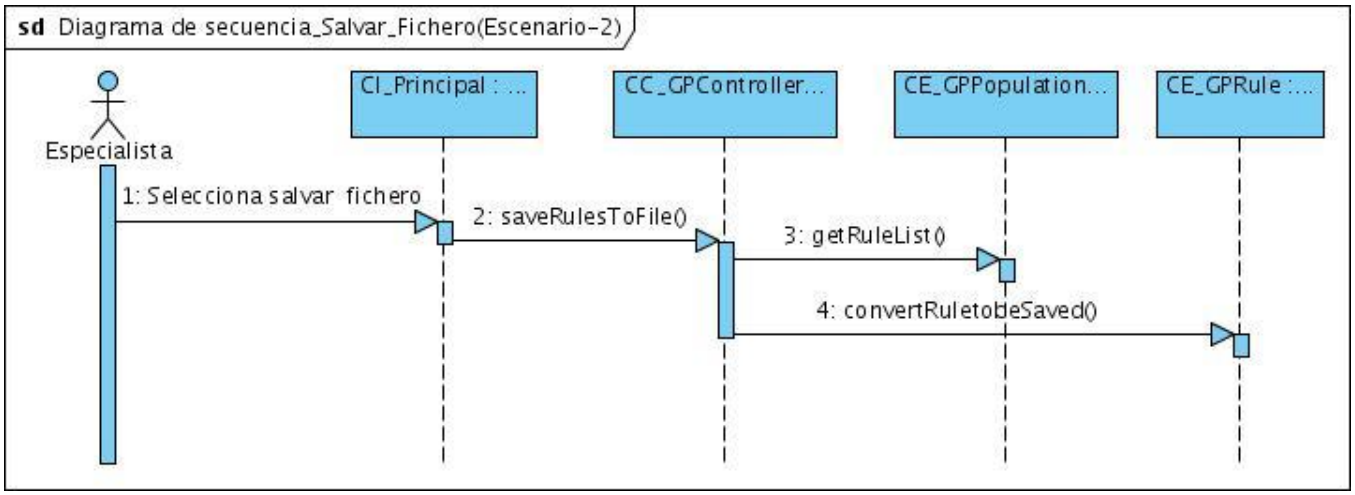

Fig. 22: Diagrama Secuencia del CU Gestionar fichero, escenario Salvar fichero

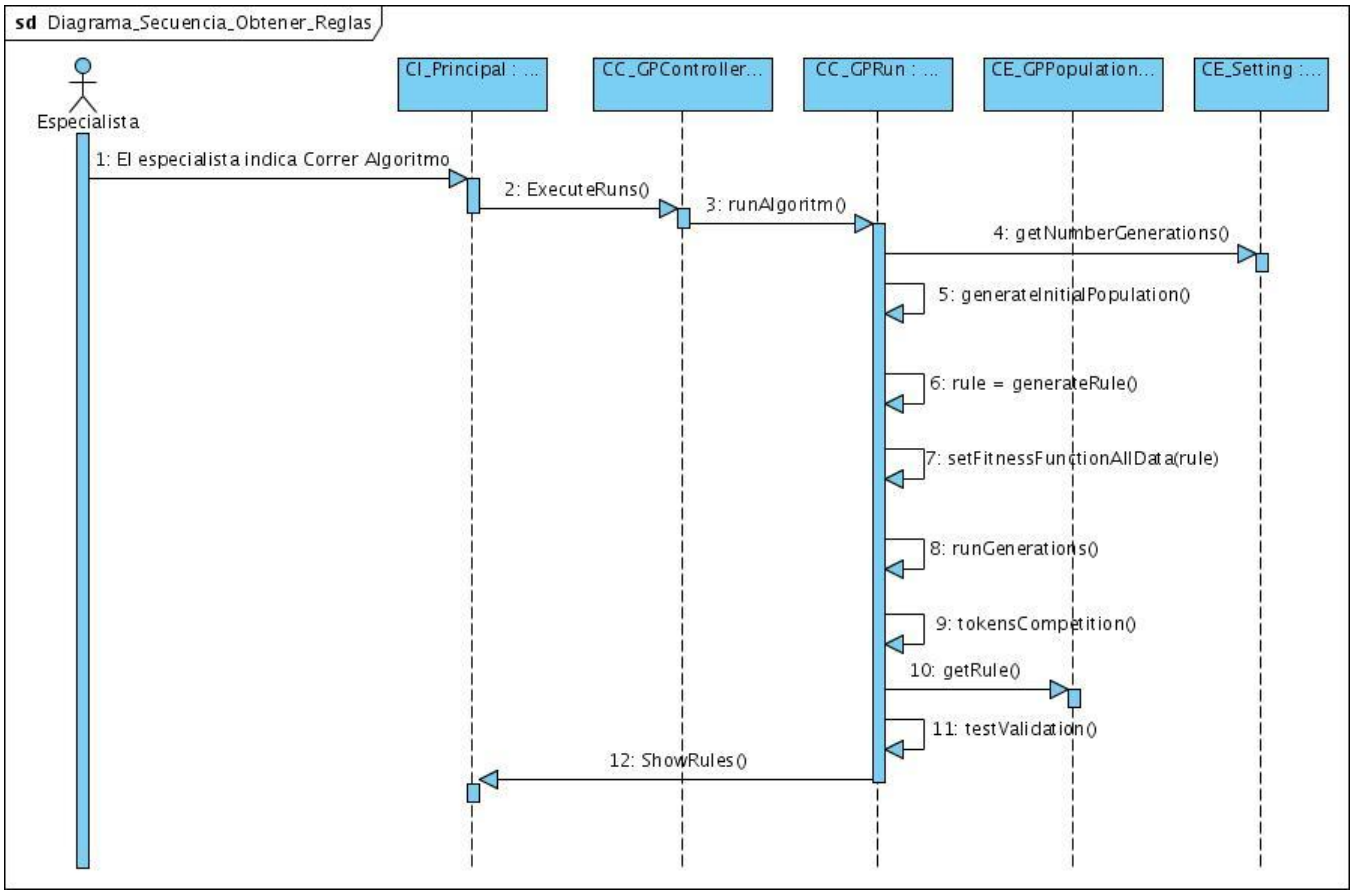

Fig. 23: Diagrama Secuencia del CU Obtener reglas de clasificación

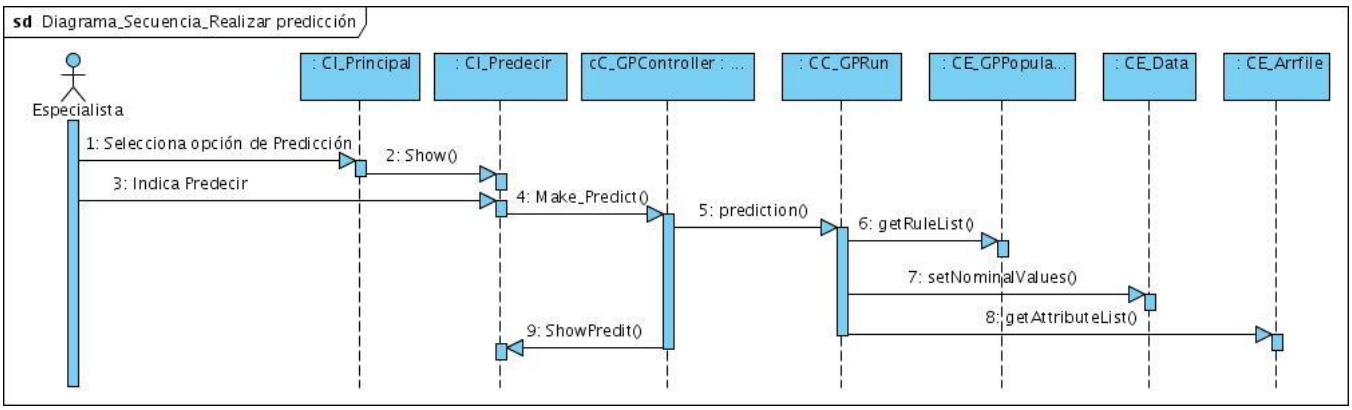

Fig. 24: Diagrama Secuencia del CU Realizar clasificación

2.5 Implementación

Dentro de esta disciplina se elabora el modelo de implementación que representa la composición física de la implementación en términos de subsistemas de implementación y elementos (Directorios y archivos, incluyendo código abierto, datos y archivos ejecutables).

2.5.1 Diagrama de Componentes

El diagrama de componentes muestra la estructura de los mismos así como las relaciones entre ellos. En este trabajo se realizó el diagrama de componentes del sistema que se muestra a continuación:

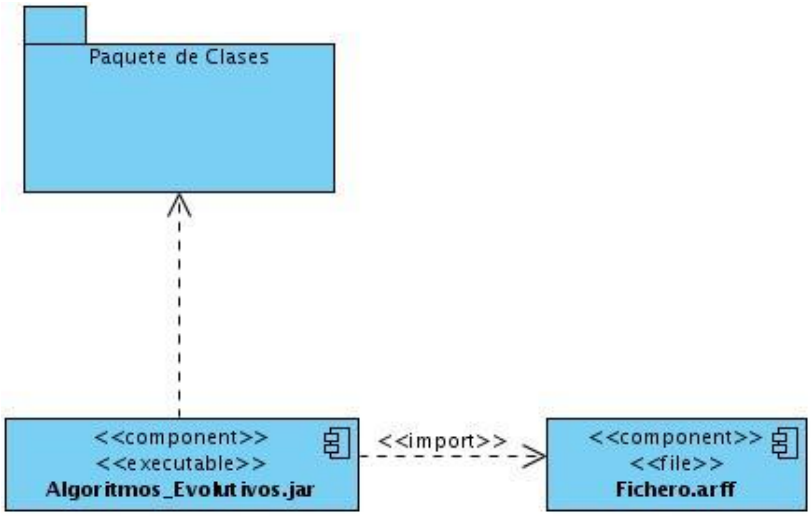

Fig. 25: Diagrama de componentes

En la figura anterior están presentes los tres componentes identificados:

- Paquete de Clases: Contiene todas las clases implementadas para la ejecución del algoritmo elaborado.
- Algoritmos_Evolutivos.jar: Es el componente ejecutable que depende de las clases implementadas para su ejecución.
- Fichero.arff: Es un fichero que contiene los datos a los cuales se les aplicará el algoritmo para obtener de esta manera las reglas de clasificación. El mismo es importado por el componente Algoritmos_Evolutivos.jar para ejecutar la funcionalidad anteriormente explicada.

2.6 Pruebas

Las pruebas son actividades en las cuales un sistema o componente es ejecutado bajo condiciones o requerimientos especificados, los resultados son observados y registrados.

La prueba de software es un elemento crítico para la garantía de la calidad del mismo y representa una revisión final de las especificaciones del diseño y de la codificación. [27]

Para lograr lo anteriormente planteado, fue elaborada y ejecutada la siguiente *estrategia de prueba* donde se especifican los niveles y tipos de pruebas a continuación:

Niveles de Prueba:

- Prueba de Unidad: Es la prueba enfocada a los elementos testeables más pequeños del software. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera. Durante el desarrollo de la herramienta se realizaron las pruebas de unidad de manera frecuente para verificar en todo momento que se obtuviera el resultado esperado.
- Prueba de sistema: Son las pruebas que se hacen cuando el software está funcionando como un todo. Es la actividad de prueba dirigida a verificar el programa final, después que todos los componentes de software y hardware han sido integrados.

Tipos de Pruebas:

- Función: Pruebas fijando su atención en la validación de las funciones, métodos, servicios, caso de uso. Para probar las diferentes funcionalidades de la herramienta se realizaron la mayor cantidad de casos de pruebas con el objetivo de mitigar los diferentes errores que pudiera haber tenido la misma.
- Volumen: Enfocada en verificar las habilidades de los programas para manejar grandes cantidades de datos, tanto como entrada y salida. Se realizaron varias pruebas, como por ejemplo en una de ellas, se cargó un fichero que contenía 12 960 datos, donde se ejecutó la herramienta de forma satisfactoria obteniendo como resultado 186 reglas en un tiempo de ejecución de 2 minutos.
- Stress: Enfocada a evaluar cómo el sistema responde bajo condiciones anormales. (extrema sobrecarga, insuficiente memoria, servicios y hardware no disponible, recursos compartidos no disponible). Esta prueba se efectuó ejecutando la herramienta en una computadora que contenía insuficiente espacio en disco duro, además de contener la misma una memoria RAM en el límite inferior especificada en los requerimientos no funcionales, de 256 Mbyte, donde se obtuvieron los mismos resultados en cuanto a efectividad del algoritmo, y similares en el tiempo de ejecución.
- Configuración: Enfocada a asegurar que funciona en diferentes configuraciones de hardware y software. Esta prueba es implementada también como prueba de rendimiento del sistema. En cuanto a configuraciones de Software, se puede asegurar que la herramienta es multiplataforma, ya que es posible ejecutarla tanto en el Sistema Operativo Linux, como en Windows, donde en este último el único requerimiento para que se logre es que tenga instalada la Maquina Virtual de Java.
- Instalación: Enfocada a asegurar la instalación en diferentes configuraciones de hardware y software bajo diferentes condiciones (insuficiente espacio en disco, etc.). Se ejecutó la herramienta tanto en Sistema Operativo Linux como Windows donde en ambas la computadora constaba con insuficiencia de espacio en disco duro así como poca memoria RAM.

Vale destacar que las pruebas efectuadas se realizaron de forma manual y los criterios de éxito y culminación de las pruebas que se tuvieron en cuenta fueron: tiempo de ejecución y porciento de efectividad del algoritmo.

Con el objetivo de demostrar que las funciones del software son operativas, que la entrada se aprueba de forma adecuada y que se produce un resultado correcto así como que se mantiene la integración de la información externa. Se decidió desarrollar el Método de Prueba de Caja Negra, las

cuales se realizan sobre la interfaz del software para comprobar el cumplimiento de los requisitos funcionales definidos. Estas pruebas permiten encontrar:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a ficheros.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Dentro de las técnicas de este método se efectuaron las siguientes:

- Técnica de la Partición de Equivalencia: esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- Técnica del Análisis de Valores Límites: esta técnica prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.

Aplicación de la Técnica de Partición de Equivalencia

La técnica de la Partición de Equivalencia es una de las más efectivas pues permite examinar los valores válidos e inválidos de las entradas existentes en el software.

Una partición equivalente es una técnica de prueba de Caja Negra que divide el dominio de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. El diseño de estos casos de prueba para la partición equivalente se basa en la evaluación de las clases de equivalencia.

Para el desarrollo de la estrategia de prueba planteada se definieron los siguientes casos de prueba correspondiente a las reglas definidas por la técnica de Partición de Equivalencia y el análisis de valores límites, donde además se muestra la respuesta del sistema y los resultados obtenidos para cada una de ellas.

Tabla 1: Casos de Prueba

Id	Esc	Tam. Pob.	No. Generac.	Prob. Reprod.	Prob. Cruzam.	Prob. Mutac.	Resp. Sistema	Result.
Escenario 3	CU2: Obtener Reglas de clasificación	200	100	**-0.1**	0.5	**1.2**	Muestra un mensaje indicando los errores cometidos	No se ejecuta el algoritmo.
		150	100	0.2	**2**	**-0.2**	Muestra un mensaje indicando los errores cometidos	No se ejecuta el algoritmo. No se ejecuta el algoritmo.
		80	90	**1.3**	**-2**	0.6	Muestra un mensaje indicando los errores cometidos	No se ejecuta el algoritmo.
		200	100	0.1	0.7	0.1	Se ejecutó el algoritmo satisfactoria mente mostrando los resultados	Se generaron 28 reglas con 70% de efectividad en 2.4 segundos.

2.7 Conclusiones

En este capítulo se especificaron seis requerimientos funcionales así como igual número de requerimientos no funcionales, englobando los mismos en tres casos de uso identificados inicializados todos por un actor, los que fueron representados en un Diagrama de Casos de Uso. Se elaboró el diseño en términos de diagramas de clases del diseño así como sus diagramas de interacción (secuencia) correspondientes a cada escenario evidenciando la utilización de patrones. Luego de realizada la implementación se realizaron las pruebas correspondientes a la herramienta obteniéndose resultados satisfactorios.

Capítulo 3
Resultados y Discusión

3.1 Introducción

En este capítulo se realizará un análisis de los resultados obtenidos con el desarrollo de este trabajo. Para esto se describirá en detalles el algoritmo que se propone y se nombrarán algunas ventajas significativas del mismo. También se exponen características y funcionalidades de la herramienta implementada. Además se presenta un análisis comparativo basado en pruebas experimentales realizadas a diferentes ficheros de muestra y con otros algoritmos, que validan el trabajo realizado.

3.2 Propuesta del Algoritmo de Clasificación

Este trabajo, como ya se ha venido tratando en apartados anteriores, propone un método para construir un clasificador basado en la evolución de reglas, sesgando la búsqueda hacia regiones de hipótesis comprensibles con alta calidad predictiva mediante Programación Genética. Para dar una visión más específica del algoritmo que se propone, se detalla a continuación todo el proceso que se llevó a cabo, sustentado por algunos códigos de implementación que facilitan su entendimiento.

En la siguiente figura se ilustra el método propuesto de forma general.

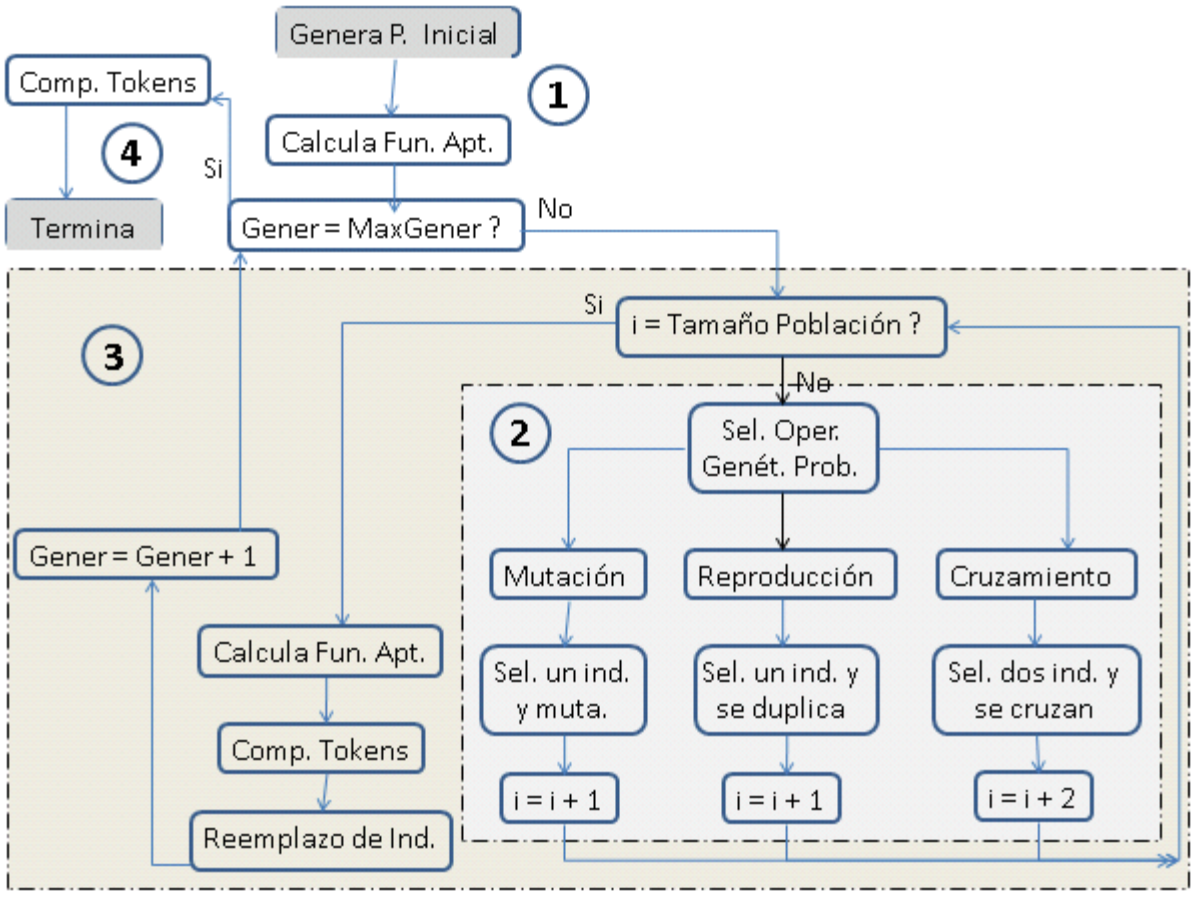

Fig. 26: Esquema que representa el método propuesto

1- Fase de inicialización:

Como se puede apreciar en la figura el algoritmo comienza generando una población inicial de individuos, o sea, reglas de clasificación, de forma aleatoria. Una regla, como se ha dicho anteriormente, está formada por un antecedente y un consecuente, y está representada mediante un árbol binario donde el subárbol izquierdo representa el antecedente y el subárbol derecho el consecuente.

Este antecedente de la regla se genera mediante el método de inicialización "Grow" [28], donde se va formando el árbol de forma tal que se vayan creando aleatoriamente nodos internos (no terminales) y nodos hojas (terminales) con los atributos y sus valores mientras se cumpla con la profundidad máxima establecida, todo esto garantizando que dicho árbol tenga una estructura válida para el antecedente de una regla.

Luego se realiza un proceso de búsqueda sobre el conjunto de datos de entrenamiento, en el cual se localiza la clase mayoritaria en dependencia de la cantidad de datos que logre emparejar dicho antecedente y se le atribuye esta clase al consecuente formándose así la regla (esto queda reflejado en el Algoritmo 3.1.1).

Algoritmo 3.1.1 Método que crea una regla.

```
public GPRule CreateRule(){
    GPRule rule = new GPRule();                                  //Inicializa la Regla
    BinaryTree<String> beforeRule = null;                        //Inicializa antecedente de la regla
    rule.setBinaryTree(new InternalNode(" THEN "));              //Crea la raíz del árbol
    arffFile.putFalseAllFlag();                                  //Se desmarcan los atributos
    arffFile.getAttributeList().get(classIndex).setFlag(true);   //Se marca el atributo clase para
                                                                 que no sea incluido en el antecedente de la regla
    beforeRule = generateBeforeRule(setting.getMaxDephtTree());  // Se genera el antecedente
    rule.getBinaryTree().SetSubArbolIzquierdo(beforeRule);       //Se le atribuye el antecedente
                                                                  a la regla
    setConsecuentToRule(rule);                                   //Se le asocia el consecuente en
                                                                  dependencia de los datos
    return rule;
}
```

Este proceso proporciona que en un inicio se generen reglas que sean capaces de cubrir al menos un dato.

Seguidamente esta regla pasa a ser evaluada mediante el cálculo de la función de aptitud, donde se utilizaron dos medidas de calidad. Por un lado está el soporte (S) [29] cuya forma de cálculo se muestra en la ecuación 3.1,

$$S = \frac{A}{T} \qquad (3.1)$$

donde A indica el número de instancias que se cumple el antecedente de la regla y T el total de instancias que se utilizó para entrenar el modelo. Por otro lado el porcentaje de aciertos (P), reflejándose su método de cálculo en la ecuación 3.2,

$$P = \frac{AC}{A} \qquad (3.2)$$

en la que AC es la cantidad de instancias que se cumple el antecedente y el consecuente de la regla. Esta medida da la probabilidad que tiene la regla de clasificar los datos correctamente.

Como se decidió utilizar una función multiobjetivo, es decir, un vector de varias medidas de calidad para lograr un balance entre ellas y hacer que se converja hacia el conjunto que está formado por las mejores soluciones (en términos de todos los objetivos individuales, no de cada uno por separado), se forma este vector con las medidas antes mencionadas y se le calcula su norma para de esta manera darle cumplimiento al cálculo de la función de aptitud, según la ecuación 3.3.

$$FA = \sqrt{S^2 + P^2} \qquad (3.3)$$

Es válido destacar que hay otras variantes de calcular una función multiobjetivo, como es el caso de la optimización de Pareto. [30]

Por último se adiciona la regla a la población repitiéndose este proceso hasta que se cumpla con el tamaño de la población predefinido antes de la corrida del algoritmo. De esta forma queda conformada la población inicial. La estructura del método que da solución a todo este proceso se muestra en el Algoritmo 3.1.2.

Algoritmo 3.1.2: Método que crea la población inicial formada por reglas.

```
public void CreatInitialPopulation(){
    int popSize = setting.getPopulationSize();
    /*Se generan tantas reglas como tamaño de la población se haya predefinido*/
    for (int i = 0; i < popSize; i++) {
    GPRule rule = CreateRule ();                    //Se crea la regla
    /*Si la regla creada tiene un consecuente es porque su antecedente capturó algún dato*/
    if(rule.getBinaryTree().GetSubArbolDerecho() != null){
    setFitnessFunctionAllData(rule);                // Se le calcula la función de aptitud
    population.addRule(rule);                       // Se adiciona a la población
       }
    /*De lo contrario se vuelve a crear otra regla que cubra al menos un dato*/
     else
          i--;
    }
}
```

2- Fase de evolución de los individuos:

Después de culminada la fase uno descrita anteriormente, se pasa a evolucionar en cada generación los individuos, para de esta forma converger a una población con los mejores individuos de acuerdo al problema que se está analizando, como una característica típica de los Algoritmos Evolutivos. Para desarrollar este proceso se comienza iterando tantas veces como cantidad de generaciones se haya predefinido en los parámetros de la corrida del algoritmo. En cada iteración se aplica un operador genético (cruzamiento, mutación o reproducción) seleccionado aleatoriamente, en dependencia de las probabilidades que posean.

Para la selección de los padres que originaran nuevos descendientes que serán incluidos en la nueva población, se utiliza el método de la ruleta, que fue descrito en la sesión 1.5.5 y está reflejado en el Algoritmo 3.1.3.

Algoritmo 3.1.3 Método que selecciona un individuo por el método de la ruleta.

```
public GPRule Selection(int popSize){
      double number = 0.0, value = 0.0;
      GPRule rule = null;
      number = random.nextDouble();       //Se genera un número aleatorio entre 0 y 1
      /* Se exploran todos los individuos hasta llegar al que coincida su probabilidad de adaptación
      con el número generado */
      for (int j = 0; j < popSize; j++) {
     value += population.getRule(j).getPortion();
     if (number <= value) {
     rule = population.getRule(j).Clone();
     break;
         }
      }
     return rule;
   }
```

La idea que se propone en el método es dar a cada individuo una probabilidad de ser seleccionado acorde a su función de aptitud y proporcional a su calidad dentro del espacio muestral de entrenamiento que se esté analizando (cuanto mejor es el valor de la función de aptitud mayor es la probabilidad de ser seleccionado y viceversa).

Después de ser aplicado el operador correspondiente se evalúan estos descendientes (de la misma forma que se describió en la fase uno), se adicionan a una nueva población y así se procede iterativamente hasta que esta última logre alcanzar el tamaño de la población predefinida.

3- Fase de evolución de las poblaciones:

En el proceso iterativo desde la primera generación hasta alcanzar la última prefijada, se realiza una estrategia de corte y reemplazo de los peores individuos por otros generados, que garantiza la diversidad de la población, así como su convergencia hacia la mejor solución. Se comienza ordenando la población de individuos por el valor de su función de aptitud, para que se logre garantizar que los

mejores individuos sean los primeros en competir por los datos a capturar y con respecto a la nueva población generada que los individuos más aptos sean los escogidos para reemplazar a los peores de la actual. Seguidamente se realiza el proceso de Competición de Tokens a la población actual ya ordenada, dicho proceso quedó descrito en la sesión 1.5.3 y está plasmado en el Algoritmo 3.1.4.

Algoritmo 3.1.4 Método que realiza el proceso de Competición de Tokens

```
public void tokensCompetition(){
    int countRules = population.getRuleList().size();
    int dataLenght = arffFile.getTrainningDataList().size();
    int count = 0, best = 0;
    double newAF = 0.0;
    ArrayList<String> list = new ArrayList<String>();
    Data data = null;

    //Se exploran todos los individuos
    for (int i = 0; i < countRules; i++) {
       list.clear();
       population.getRule(i).getBinaryTree().PreOrden(list);   /*Se convierte la regla(árbol) en
                                                                una cadena para una mejor
                                                                comparación con los datos*/
       count = best = 0;
       for (int k = 0; k < dataLenght; k++) {
          data = arffFile.getTrainningDataList().get(k);
     /*Si el antecedente de la regla empareja con el dato esta lo captura*/
          if(validateAntecedent(recorrido, data, true)){
             if(data.getMarked() != 1){
```

```
                count ++;
                 data.setMarked(1);
              }
              best++;
            }
          }
          if(best == 0)
            newAF = 0;
          else newAF = population.getRule(i).getAptitudFuntion() * (double)count / best;
          population.getRule(i).setModifiedAptitudFuntion(newAF); /* Se modifica el
                                                                    valor de la función de aptitud*/
        }
      /*Todos los individuos que llevan su función de aptitud a cero son eliminados de la población*/
        for (int i = 0; i < countRules; i++) {
    if(population.getRule(i).getModifiedAptitudFuntion() == 0){
            population.removeRuleAt(i);
            i--;  countRules--;  }
        }
        arffFile.putCeroAllData();                    //Se desmarcan los datos
      }
```

En este proceso se analizan cada uno de los individuos para ver a cuántos datos puede capturar y luego, los que logran emparejar con dicho individuo, son marcados para que otro individuo menos apto que él no pueda apoderarse de este recurso. Luego se le actualiza a cada individuo el valor de la función de aptitud según la ecuación (3.4),

$$FM = FO \times \frac{C}{M} \qquad (3.4)$$

donde FM es la función de adaptación modificada, FO es el valor de la función de aptitud obtenida por el individuo en el proceso de adaptación, C es el número de ejemplos que el individuo ha conseguido capturar y M es el número máximo de ejemplos que el individuo puede capturar. Finalmente todos los individuos cuyo valor de aptitud es cero son eliminados de la población, ya que esto indica que no

pudieron capturar ningún dato. Por tanto, estos k individuos son redundantes y son reemplazados por los k primeros individuos de la nueva población generada, lo que puede aportar un mayor grado de diversidad a la población y además proporcionar cambios adicionales para la generación de buenos individuos.

4- Fase de culminación:

Como conclusión se vuelve a aplicar el proceso de Competición de Tokens a la última población que se obtiene, para de esta forma culminar el algoritmo con un conjunto de reglas libres de redundancias, y con alto nivel de precisión y que puede tener un número menor de reglas que el tamaño de la población predefinida.

3.3 Descripción de la Herramienta

Debido a un previo estudio que se realizó en el Capítulo 1 de este trabajo, se ha desarrollado una herramienta específica con el objetivo de facilitar el proceso de descubrimiento de reglas de clasificación y darle una aplicación al algoritmo que se propone en este trabajo.

La herramienta implementada presenta una entrada de datos a procesar mediante ficheros con un formato específico (.arff). En la siguiente figura se muestra la estructura que presentan estos ficheros.

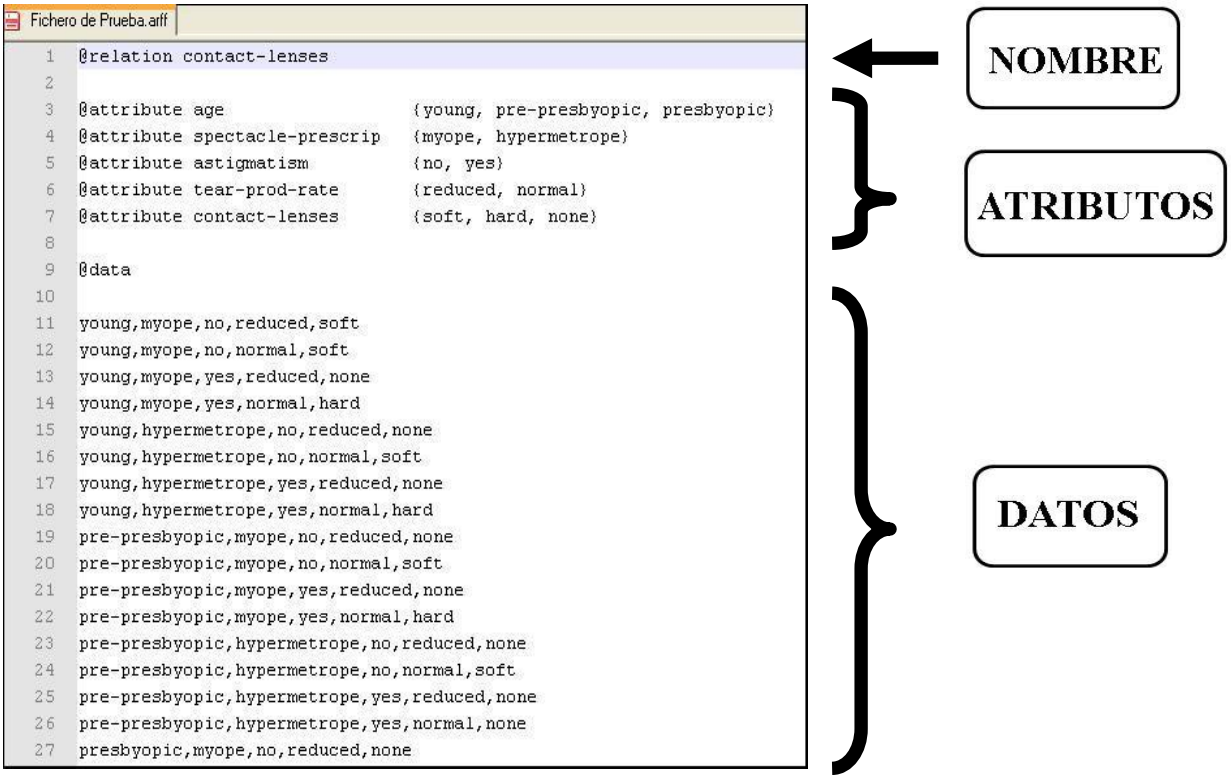

Fig. 27: Estructura del fichero

Como bien muestra la figura 25 el fichero está estructurado por el nombre del mismo, seguidamente los atributos con los posibles valores que puede tomar (en este caso los atributos son nominales), generalmente el último atributo es el que posee la clase a clasificar. Por último están los datos que se tienen para entrenar o probar el clasificador que se quiere obtener.

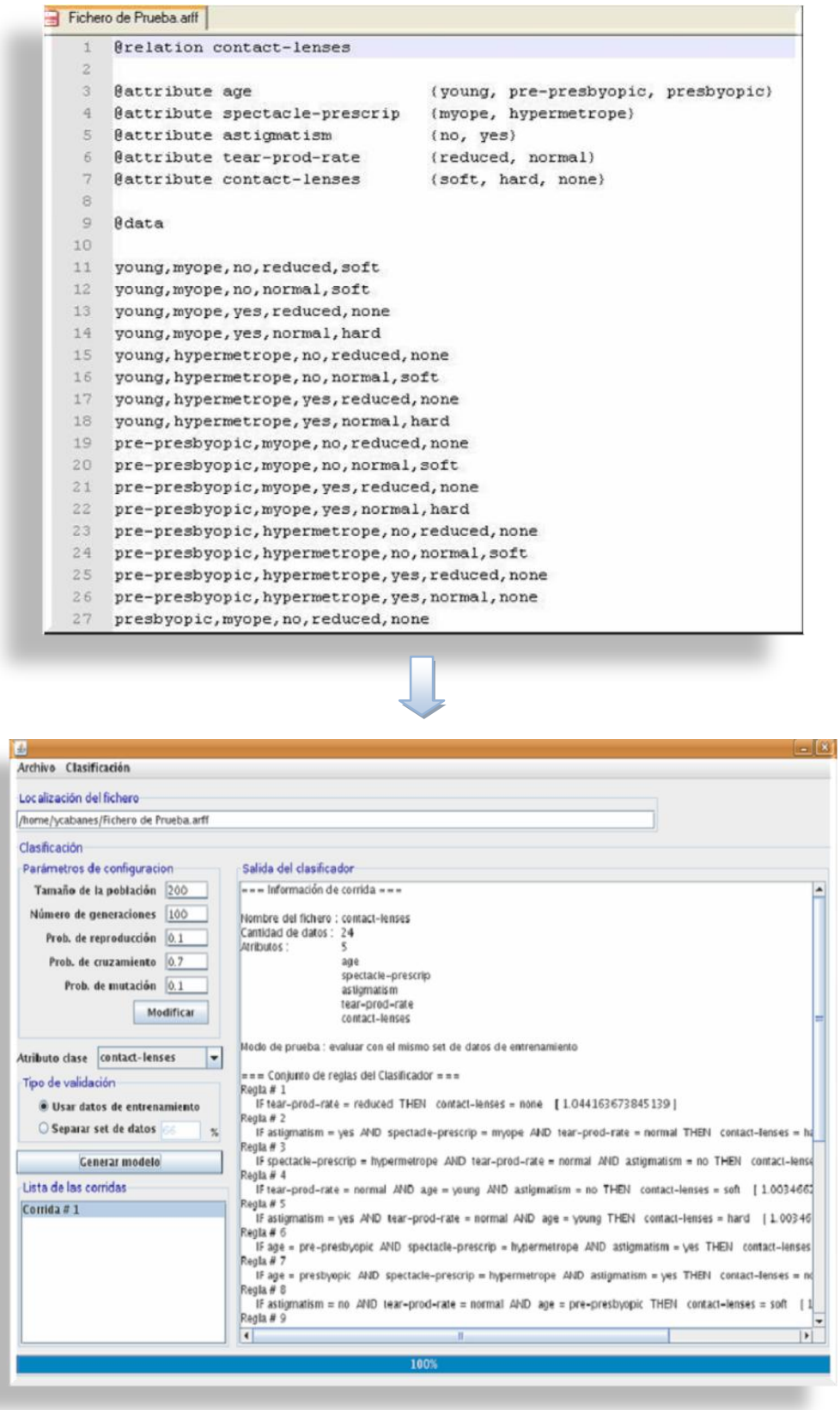

Fig. 28: Diseño General de la Aplicación

La herramienta presenta además una amigable e intuitiva interfaz visual, lo que permite que el usuario cuente con diferentes facilidades en su trabajo con la misma. A continuación se detallan estas características a las que se hace alusión.

- Primeramente se puede decir que el modelo generado, o sea el conjunto de reglas de clasificación que se obtienen, es mostrado de una forma clara y precisa, acompañado de diferentes valores estadísticos que se calculan asociados al proceso de validación del modelo obtenido, como son: la cantidad de instancias correctamente clasificadas, las incorrectamente clasificadas así como las no clasificadas. Además se muestra la matriz de confusión, que es otra forma de evaluación de un modelo de clasificación.

- Por otra parte la herramienta brinda la posibilidad de realizar varias corridas, es decir, almacena los diferentes modelos que puedan ser generados, para que el usuario puede saber en todo momento los resultados de las corridas que haya realizado y da la posibilidad de salvarlos en ficheros.

- Como ya se habló en el Capítulo 1 las reglas de clasificación están compuestas por un antecedente y un consecuente donde este último posee una sola condición. Estas reglas son muy populares en minería de datos debido a que representan conocimiento a un alto nivel de abstracción y se puede utilizar directamente en procesos de toma de decisiones. Una vez que se ha generado un modelo, se tiene la posibilidad de dada las características de una nueva instancia decir a que clase pertenece, o sea, clasificarlo de acuerdo al modelo generado. Para esto primeramente se muestran, de manera dinámica, los atributos con sus respectivos valores para que estos sean seleccionados de acuerdo a las características que posea la instancia a clasificar, todo esto evita que se haga un proceso de validación de entrada de datos. Una vez obtenidos los valores de los atributos de la nueva instancia que se va a clasificar, se compara cada regla del modelo obtenido con estos nuevos valores de la instancia para ver cual empareja o está contenida y finalmente si se encuentra una o varias reglas que cumplan la condición anterior y presenten un único valor de clase, este se le asigna a la nueva instancia, de lo contrario se emite que no se puede clasificar dicha instancia.

3.4 Análisis comparativo de resultados obtenidos

Se han realizado diferentes pruebas orientadas a comparar y validar los resultados que produce el algoritmo que se propone en la tarea de descubrimiento de conocimiento, Clasificación. Para esto se determinó comprobar los resultados obtenidos, realizando dos tipos de corridas diferentes en relación con la cantidad de datos usados para entrenar y probar los modelos generados, por cada uno de los ficheros de muestra seleccionados de las Bases de Datos (Contact-Lenses, Wisconsin, Tic-Tac-Toe, Car) del UC Irvine Machine Learning Repository [31], cuyas características principales quedan resumidas en la tabla 2. También se realizó un análisis comparativo entre el algoritmo propuesto y otros implementados en una de las herramientas estudiadas que generan modelos formados por reglas de producción y basan su mecanismo de funcionamiento mediante enfoques evolutivos.

Tabla 2: Características de los ficheros utilizados

Nombre Fichero	No. de Atributos	No. De Clases	No. de Ejemplos
Contact-Lenses	5	3	24
Wisconsin	10	2	683
Tic-Tac-Toe	10	2	958
Car	7	4	1728

Para estos análisis se decidió tomar un tamaño de muestra de población inicial de 100, con un número de generaciones de 200 y las probabilidades de los operadores genéticos reproducción, cruzamiento y mutación fueron de 0.1, 0.7 y 0.1 respectivamente, quedando reflejados estos parámetros en la siguiente figura.

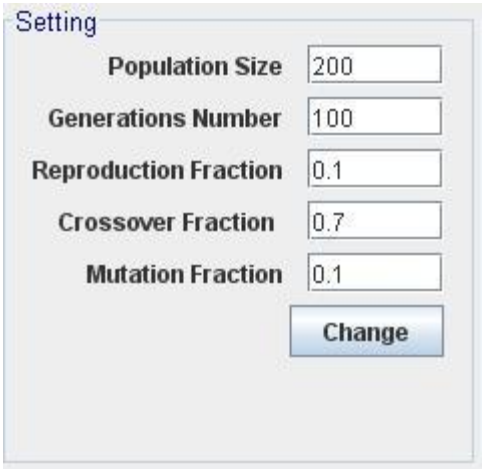

Fig. 29: Parámetros para la corrida del algoritmo

Los tipos de corridas utilizados son los siguientes:

1. Se utiliza el mismo conjunto de datos de entrenamiento y prueba.
2. Se particiona el conjunto de datos en dos, un 50% para el entrenamiento y el resto para prueba.

Al realizar las diferentes corridas para cada fichero analizado se obtuvieron los siguientes resultados:

Tabla 3: Resultados de las corridas con los ficheros seleccionados

Nombre del Fichero	Tipo de Corridas	Total Reglas	(%) Ejemplos Correctamente Clasificados	Tiempo (s)
Contact-Lenses	1	17	95.8	0.71
	2	10	33.3	0.46
Wisconsin	1	91	78.6	7.24
	2	56	78.1	4.09
Tic-Tac-Toe	1	200	64.9	11.39
	2	186	61.8	6.5
Car	1	55	70.0	16.8
	2	74	64.7	10.4

Como se puede ver en la tabla de resultados anterior, los porcientos de ejemplos correctamente clasificados, analizado para cada fichero, se comportan de forma similar en los dos tipos de corridas que se efectuaron, solo disminuye en un rango de 0.5% a 5.3%, lo que demuestra que la disposición

de los datos de entrenamiento y prueba de forma variable, no afecta en grandes porcientos la precisión de clasificación.

Por otra parte se determinó realizar una comparación con dos algoritmos de clasificación que generan modelos basados en reglas de producción de la herramienta KEEL (LogenPro y PGIRLA). El primero de estos algoritmos está implementado mediante Programación Genética y el segundo está basado en Algoritmos Genéticos, por lo que ambos presentan características evolutivas propias muy parecidas al que se trata en este trabajo, lo que facilita una mejor comparación.

Al realizar estas pruebas se obtuvieron los siguientes resultados expresados en porcientos de ejemplos correctamente clasificados:

Tabla 4: Resultados de las corridas con los algoritmos y ficheros seleccionados

Algoritmos	Contact-Lenses	Wisconsin	Tic-Tac-Toe	Car
Algoritmo propuesto	**95.8**	**78.6**	64.9	**70**
LogenPro	83.3	62.8	**69.2**	69.2
PGIRLA	58.3	78.4	56.2	56.2

Como se puede apreciar en las dos tablas anteriores, los resultados obtenidos con el algoritmo que se propone en cuanto a la medida de precisión de clasificación que se presenta, comparándolos con los que arrojan LogenPro y PGIRLA son muy similares y en tres casos arroja los mejores resultados, lo que da una medida de validez del algoritmo que se propone.

3.5 Conclusiones

En este capítulo se abordaron y discutieron los diferentes resultados obtenidos, o sea, se explicó de manera detallada el algoritmo propuesto y se plantearon las principales características de la herramienta implementada, así como su estructura y potencialidades. Además, se realizó un análisis comparativo con otros algoritmos que presentan características muy similares al que se trata en este trabajo, que mediante los resultados obtenidos se demostró la validez del algoritmo propuesto. Estos algoritmos con los que se comparó, principalmente LogenPro basado en Programación Genética, son algoritmos probados y validados, lo que permite dar una medida de los resultados de este trabajo.

CONCLUSIONES

- Se desarrolló la propuesta de un algoritmo para la generación de reglas de clasificación basado en Programación Genética.

- La identificación de los requerimientos propició el diseño e implementación de la herramienta para la generación de reglas de clasificación basada en el algoritmo propuesto, además la aplicación de una estrategia de prueba garantizó la calidad de la misma.

- Se realizó un análisis comparativo de los resultados obtenidos con otros algoritmos existentes que resuelven problemas similares, validando de esta manera la calidad del algoritmo implementado.

- Los resultados alcanzados en este trabajo constituyen un paso de avance más en el estudio de los Algoritmos Evolutivos, fundamentalmente la Programación Genética como método novedoso y potencialmente prometedor en la resolución de tareas de Minería de Datos.

RECOMENDACIONES

- Estudiar la posibilidad de implementar variantes distribuidas o paralelas en aras de mejorar los tiempos de corrida del algoritmo.

- Realizar un análisis y desarrollo de nuevas métricas o medidas para evaluar la calidad de las reglas generadas, así como mantener el enfoque multiobjetivo que se plantea en este trabajo que promete muy buenos resultados.

- Convertir la herramienta a un plug-in que luego pueda ser utilizado por herramientas que implementen tareas de Minería de Datos.

- Incluir proceso de discretización de atributos continuos. Esto extenderá el alcance de la herramienta al permitir que los archivos de datos contengan atributos de este tipo.

REFERENCIAS BIBLIOGRÁFICAS

[1] Serrano, Giselle. Herramienta para la generación de reglas de asociación basada en un algoritmo de Programación Genética. UCI. Ciudad de la Habana, 2007. 61.

[2] Fernandez, Francisco. Mejoras en los Algoritmos Evolutivos: Algoritmos Evolutivos Paralelos. 2005

[3] Hernandez, José. Ramirez, Jose. Ferri Cesar. Introducción a la Minería de Datos. España. Editorial Pearson. 2004

[4] García, C. Ignacio. Representación del conocimiento. Universidad de Burgos. [Disponible en: http://pisuerga.inf.ubu.es/cgosorio/SExInArt/UD4/introKR.pdf].

[5] Romero, Cristóbal. Ventura Sebastián. Castro, Carlos. Aplicación de algoritmos evolutivos como técnica de minería de datos para La mejora de cursos Hipermedia adaptativos basados en Web.

[6] Koza J, Genetic Programming, on the programming of computers by means of natural selection. The MIT Press, 1992.

[7] Rocío, Rubén, Rossana, Rafael. Programación Genética. 2002. [Disponible en: http://www.depi.itch.edu.mx/apacheco/expo/html/ai14/gp.html].

[8] Torres, Juan Esteban, Martínez, José Jesús. TUTORIAL DE PROGRAMACIÓN GENÉTICA.

[9] Romero, C., Ventura, S., Hervas, C. Descubrimiento de Reglas de Predicción en Sistemas de e-Learning utilizando Programación Genética. 2004.

[10] Espejo, P. G., Hervás, C., Ventura, S., Romero, C. Elección de Operadores Lógicos para la Inducción de Conocimiento Comprensible, Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 2006, no. 29, pp. 19-30.

[11] Espejo, P. G., Hervás, C., Ventura, S., Romero, C. Elección de Operadores Lógicos para la Inducción de Conocimiento Comprensible, Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 2006, no. 29, pp. 19-30.

[12] Riquelme J, Fdez. Bejarano F, Glez. Morón P. y Toro M. Clasificación simbólica mediante programación genética. VII Conferencia de la Asociación Española para la Inteligencia Artificial Málaga, Noviembre 1997. pp. 571-580.

[13] Aguilar, J.L., Altamiranda, J. Un Algoritmo Basado en la Programación Genética para Minería de Datos, 2002 [Disponible en: http://www.fing.edu.uy/infouyclei2002/conflat/articulos/resumenes/136.html]

[14] Herramienta DBMiner [Disponible en: http://www.cs.sfu.ca/CC/459/han/tutorial/tutorial.html]

[15] Herramienta Weka [Disponible en: http://www.cs.waikato.ac.nz/ml/weka/]

[16] Herramienta Keel [Disponible en: http://sci2s.ugr.es/keel/members.php]

[17] Berlanga, Jose, Herrera, Francisco. Aprendizaje de reglas difusas mediante programación genética en problemas con alta dimensionalidad.

[18] Marczyk, Adam. Algoritmos genéticos y computación evolutiva. 2004 [Disponible en: http://the-geek.org/docs/algen/]

[19] Sitio Web de OpenUp [Disponible en: http://epf.eclipse.org/wikis/openup/index.htm]

[20] Manual del Visual Paradigm.

[21] Lemay, Laura; Perkins, Charles. Aprendiendo Java en 21 días. México, Prentice-Hall Hispanoamericana S.A., 1996. p. 525.

[22] NetBeans, Basic IDE Concepts [Disponible en: http://www.netbeans.org/kb/55/using-netbeans/project_setup.html#pgfId-1020595]

[23] Wikipedia [Disponible en: http://es.wikipedia.org/wiki/Arquitectura_de_software]

[24] Wikipedia [Disponible en: http://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o]

[25] Universidad Tecnica Particular de Loja [disponible en:http://www.utpl.edu.ec/blog/zamora/2008/02/13/arquitectura-por-capas/]

[26] Design Patterns. Elements of Reusable Object-Oriented Software - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides - Addison Wesley (GoF- Group of Four).

[27] Flujo de trabajo de Prueba. Departamento de Ingeniería y Gestión de Software, Universidad de Ciencias Informáticas. 2006-2007. Conferencia.

[28] Angel G Baños, Computación Evolutiva (CE), Programación Genética, Evolución Gramatical, Programación por Expresión Genética, Escuela de Ingeniería de Sistemas y Computación, Universidad del Valle, febrero 2008.

[29] César H. Martínez, Cristóbal R. y Sebastián V. Selección de medidas de evaluación de reglas obtenidas mediante programación genética basada en gramática. Universidad de Córdoba.

[30] Coello, Carlos A. Introducción a la Optimización Evolutiva Multiobjetivo [Disponible en: http://neo.lcc.uma.es/pdf-charlas/MOEA.pdf]

[31] Repositorio de Bases de Datos UCI [Disponible en: http://archive.ics.uci.edu/ml/]

BIBLIOGRAFÍA

(1)- Hernandez, José. Ramirez, Jose. Ferri Cesar. Introducción a la Minería de Datos. España. Editorial Pearson. 2004.

(2)- Guerra Alejandro, Aprendizaje automático: Clasificación. 2004, [Disponible en: www.mia.uv/aguerra]

(3)- Torres Juan Esteban, Martínez José Jesús. Tutorial de Programación Genética.

(4)- Aprendizaje Automático y Minería de Datos. 2006, Universidad Nacional de San Luis. Argentina.

(5)- García Salvador, Cano J. Ramón, Herrera Francisco. Un algoritmo memético para selección de prototipos: Una propuesta eficiente para problemas de tamaño medio.

(6)- Bejar Javier. Reglas de Clasificación. 1999.

(7)- Romero, Cristóbal. Ventura Sebastián. Castro, Carlos. Aplicación de algoritmos evolutivos como técnica de minería de datos para La mejora de cursos Hipermedia adaptativos basados en Web.

(8)- César H. Martínez, Cristóbal R. y Sebastián V. Selección de medidas de evaluación de reglas obtenidas mediante programación genética basada en gramática. Universidad de Córdoba.

(9)- Sánchez Luciano, Couso Inés, Corrales J. A. Combining GP operators wit SA search to evolve fuzzy rule based classifiers. 2001.

GLOSARIO DE TÉRMINOS

Algoritmo: Lista bien definida, ordenada y finita de operaciones para generar las reglas.

CASE: Computer Aided Software Engineering (Herramientas de ingeniería de software asistida por computadora).

Clasificación: Tarea de Minería de Datos.

Entrenamiento: Acción que se realiza para el aprendizaje del conjunto de reglas.

Especialista: Persona que interactúa con el sistema para obtener reglas de clasificación.

IDE: Un "Integrate Development Enviroment" es una herramienta de soporte al proceso de desarrollo de software que integra las funciones básicas de edición de código, compilación y ejecución de programas, entre otras.

Java: Es un lenguaje de programación, de alto nivel, orientado a objetos y desarrollado por Sun Microsystems.

Metodología: Define quién hace qué, cómo y cuándo.

Modelo: Representación abstracta de la realidad.

OpenUp/Basic: Metodología para el desarrollo de software.

Plug-ins: Aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica.

Predicción: Valor estimado que caracteriza una propiedad o fenómeno obtenido de un modelo.

Programación Genética: Es un concepto que surge a comienzos de la década del noventa que consiste en la evolución automática de programas usando ideas basadas en la selección natural.

Software: Término genérico que designa al conjunto de programas que posibilitan realizar una tarea específica en un ordenador.

UML: Lenguaje de Modelado Unificado (Unified Model Languaje), lenguaje gráfico que brinda un vocabulario y reglas para especificar, construir, visualizar y documentar los artefactos de un sistema utilizando el enfoque orientado a objetos.