Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior

Trabajo Fin de Grado

DESARROLLO DE UN SOFTWARE R PARA EL DESCUBRIMIENTO DE SUBGRUPOS

Alumno: Rubén Crespo Rayo

Tutor: Prof. Cristóbal J. Carmona del Jesus
Dpto: Departamento de Informática

Febrero, 2023

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Departamento de Informática

Don CRISTÓBAL J. CARMONA DEL JESUS y Don ANGEL MIGUEL GARCÍA VICO, tutores del Proyecto de Fin de Carrera titulado: DESARROLLO DE UN PAQUETE PARA EL SOFTWARE R PARA EL DESCUBRIMIENTO DE SUBGRUPOS, que presenta RUBÉN CRESPO RAYO, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Febrero de 2023

El alumno:

Firmado por CRESPO RAYO RUBEN - ***9533** el día 24/02/2023 con un certificado emitido por AC

RUBÉN CRESPO RAYO

El tutor:

Firmado por CARMONA DEL JESUS CRISTOBAL JOSE - 75102603J el día 24/02/2023 con un

CRISTÓBAL J. CARMONA DEL JESUS

Índice

Índice de tablas

Índice de figuras

Índice de ecuaciones

Capítulo 1. Ciencia de datos

1.1. Introducción y machine learning

1.1.1. Introducción

En los años recientes, se ha estado percibiendo un aumento muy significativo de la cantidad de datos e información disponibles en todas las áreas de la sociedad. La transformación digital está ocasionando un aumento exponencial de dichos datos y de las fuentes que los generan.

Y es que el volumen de información generada crece día a día y las previsiones sobre ese crecimiento también lo hacen. Para ejemplificar esto, veamos datos reales sobre éste pronóstico. El año pasado se generó alrededor de 79 zettabytes de información y se prevée que éste año (2022) se generen unos 97 zettabytes [1]. El volumen de datos previsto para 2025 se espera que llegue a 181 zettabytes, lo que significa el equivalente a 90 veces la información generada en 2010.

Figura 1. Crecimiento de la información previsto

La historia ha demostrado que la información es el arma más poderosa que tiene la humanidad. Hoy en día, las empresas y los gobiernos de todo el mundo se gastan millones y millones de euros en obtener datos. Datos sobre personas, sus gustos, su día a día, sobre tendencias, sobre otras empresas y otros gobiernos, sobre una línea de producción… sobre absolutamente todo.

Sin embargo, de poco sirve tener tal cantidad de datos si no se es capaz de extraer la información relevante o interpretar dichos datos. Y es que hasta hace muy poco tiempo, todo este análisis de datos se ha estado realizando a mano. Pero gracias al abaratamiento de los costes de almacenaje y a la imposibilidad de realizar a mano todo este análisis, empezaron a desarrollarse herramientas de analítica de datos, tales como las herramientas OLTP (On-Line Transaction Processing) y OLAP (OnLine Analitical Processsing). Sin embargo, estas herramientas no son lo suficientemente potentes como para extraer conocimiento de manera automática o semi-automática.

De ello, que surge la necesidad del proceso del llamado Descubrimiento de Conocimiento en Bases de datos (o más conocido como, Knowledge Discovery in Databases o KDD). El KDD tiene como fin tomar un gran volumen datos y analizarlos, para extraer una base de conocimiento útil para un posterior toma de decisión. Proceso que incluye las etapas de extracción de datos, preprocesamiento de datos, etc. La minería de datos (Data Mining o DM), aunque la más importante, es sólo una de sus etapas y consiste en identificar patrones que sean válidos, novedosos, potencialmente útiles y capaz de proporcionar datos comprensibles [2].

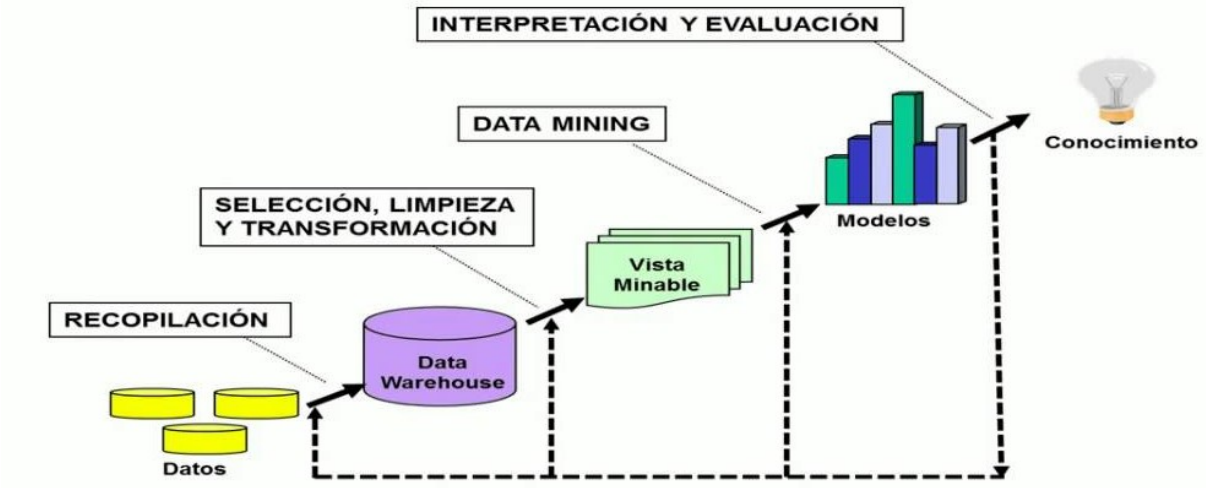


Figura 2: Procesos del KDD

Aunque actualmente, existen múltiples plataformas para la utilización de algoritmos de minería de datos, como por ejemplo WEKA [3], KEEL [4] o KNIME [5], entre otras. El software en que se centra este trabajo será R [7]. Y es que a día de hoy, R es la herramienta estadística más utilizada en ciencia de datos [8] y el número de usuarios va en aumento con cada año que pasa.

Sin embargo, a pesar de que dentro de sus repositorios ya se encuentra una librería que contiene los mismos algoritmos de descubrimiento de subgrupos que trataremos en este proyecto, éstos se encuentran programados en lenguaje R. El cual a pesar de ser el lenguaje nativo de éste software, no es la opción más eficiente para ésta tarea. Por tanto, utilizaremos un paquete llamado RCPP disponible en el repositorio CRAN [9] que permite emplear código C o C++ en proyectos R. Ésta es una alternativa más eficiente y sencilla, ya que uno de los principales motivos para usar este paquete es el hecho de que un algoritmo escrito en C o C++ es generalmente más rápido que el mismo escrito en R. Otro posible motivo para utilizar este paquete en este tipo de proyectos es la posibilidad de utilizar código ya existente en nuestros proyectos de R.

Los algoritmos de descubrimiento de subgrupos que implementaremos serán SDIGA [10], MESDIF [11], NMEEF-SD [12] y FuGePSD [55]. La característica fundamental de éstos algoritmos es que, entre los algoritmos que utilizan el modelo de aplicación de Descubrimiento de Subgrupos del que hablaremos más adelante, pertenecen a la clasificación de enfoque evolutivo. Este enfoque lo consiguen implementando un algoritmo genético como base, lo que les permite obtener soluciones normalmente no óptimas en espacios de búsqueda muy grandes, pero si muy buenas en relación al tiempo de ejecución empleado para descubrir los subgrupos. Destacamos que el tiempo de ejecución de un algoritmo que siga una estrategia de búsqueda exhaustiva se eleva de manera exponencial al número de atributos que tengamos. La gran ventaja que poseen frente al resto de algoritmos de la bibliografía especializada es que permiten trabajar con valores perdidos y hacen uso de lógica difusa para aumentar las interpretabilidad de los subgrupos obtenidos.

1.1.2. Machine learning

El Machine Learning es una disciplina científica que, junto a una base de datos, forma parte del proceso de la Minería de Datos que crea sistemas que aprenden automáticamente. Aprender, en este contexto, quiere decir identificar patrones complejos en millones de datos. Una máquina que realmente aprende es un algoritmo que revisa los datos y es capaz de predecir comportamientos futuros.

Como se ha dicho, el objetivo de la minería de datos se trata de extraer patrones (no triviales), normalmente desconocidos y potencialmente útiles para el usuario, a partir de grandes volúmenes de datos. Se trata de un área multidisciplinaria, capaz de extraer información útil tal como estudiar largas secuencias de ADN para el estudio de enfermedades o incluso tratar de analizar nuestras opiniones, preferencias y sentimientos expresados en las redes, entre otras aplicaciones.

Como una disciplina de la inteligencia artificial, el machine learning está inspirado por una de las principales propiedades de los sistemas inteligentes: el aprendizaje.

Los principales modelos de aplicación en la minería de datos son el modelo predictivo o supervisado y el modelo descriptivo o no-supervisado. Además de estos dos están el modelo de aprendizaje semi-supervisado (Chapelle et al., 2006), aprendizaje reforzado (Sutton y Barto, 1998), el modelo de transducción (Vapnik, 1998), el modelo de meta-aprendizaje (Vilalta y Drissi, 2002) entre otros.

1.2. Minería de datos predictiva

Ésta categoría de modelos son también llamados como supervisados, métodos asimétricos o directos.

La terminología del machine learning está basada en la terminología del lenguaje humano. En el aprendizaje supervisado humano, hay un profesor, tutor o supervisor el cual conoce de antemano las preguntas y las respuestas correctas y, dando las respuestas al aprendiz, le ayuda a aprender una nueva habilidad.

El machine learning supervisado emula al supervisor al proporcionar al aprendiz datos para entrenar. Los datos de entrenamiento consisten en pares de ejemplos de entrada - preguntas (usualmente en forma de vectores de valores atributo), salidas deseadas - respuestas (llamadas valores objetivo). Estos valores objetivo son habitualmente valores categóricos (en tal caso estamos hablando de una clasificación, donde cada posible categoría es una clase) o números. El objetivo del aprendizaje supervisado es obtener la capacidad de predecir el valor de las salidas para cada entrada válida, habiendo visto previamente un cierto número de ejemplos de entrenamiento (parejas de entradas y salidas). Para lograr esta capacidad de predicción, el aprendiz tiene que generalizar a partir de los datos presentados y desarrollar una función que asigne entradas a las salidas en función de los datos de entrada.

En resumen, la minería de datos predictiva se basan en entrenar un modelo o método por medio de diferentes datos para encontrar patrones con los que, como su nombre indica, intentar predecir un comportamiento futuro de una o más variables partiendo de estos mismos datos.

Los principales métodos de éste tipo son los modelos estadísticos clásicos, como los modelos de clasificación, de regresión lineal, series temporales; o los modelos

desarrollados en el ámbito de la máquina de aprendizaje, tales como las redes neuronales (como Back-Propagation [14]) y árboles de decisión (como C4.5 [15]).

1.3. Minería de datos descriptiva

Ésta categoría de modelos son también llamados como no-supervisado, modelos simétricos o indirectos. Se caracterizan por la ausencia de una etiqueta, clase o salida dentro de los mismos.

Comparado al aprendizaje supervisado, el aprendizaje no supervisado trata de aprender sin ningún supervisor. Al traducir esto a terminología del machine learning, el aprendizaje no supervisado trata de aprender de datos para los que no se proporciona valores objetivo. La meta del aprendiz es construir un representaciones de los datos de entrada. En cierto sentido, se puede considerar que el aprendizaje no supervisado encuentra patrones en los datos más allá de lo que se consideraría puro ruido no estructurado.

La minería de datos descriptiva tiene como objetivo encontrar patrones interesantes en los datos. La minería de datos descriptiva describe los datos de manera concisa y presenta características interesantes de los datos, generalmente sin tener un objetivo predefinido. El resultado de la minería de datos descriptiva es una descripción de un conjunto de datos de manera concisa y resumida, la presentación de las propiedades generales de los datos, así como la descripción de patrones locales en los datos.

Funciona analizando los datos y agrupándolos en categorías que no eran conocidos con anterioridad, pues los datos de estos grupos podrían estar relacionados mediante vínculos que a su vez eran desconocidos anteriormente. De ésta manera, todas las variables disponibles son tratadas en el mismo nivel y no hay hipótesis de causalidad.

Estos modelos se dedican a realizar resúmenes y agrupamiento de datos (como KMeans [16]), descubrir relaciones entre los datos actuales obteniendo reglas de asociación (como Apriori [17]), así como realizar análisis estadísticos de dispersión y estudios de distribución de datos.

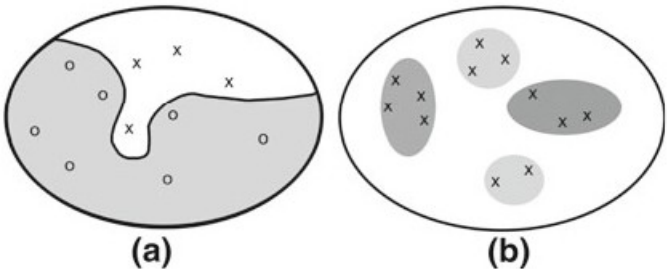


Figura 3: Modelos de diferentes técnicas de KDD [20]

En la *Figura 3*, la principal diferencia entre la inducción predictiva y descriptiva puede ser encontrada a simple vista. La *Figura 3.izquierda* representa un modelo complejo y preciso (clasificador) para inducción predictiva el cual divide perfectamente el espacio en dos regiones determinadas con respecto al tipo de objetos en el conjunto. Este modelo está basado en la precisión e interpretabilidad. Sin embargo, la *Figura 3.derecha* representa un modelo para inducción descriptiva el cual describe grupos de elementos (clusters) en el conjunto, sin una variable objetivo, y basado en el soporte y la confianza de los objetos. Como puede ser observado, el modelo de la izquierda (inducción predictiva) tiene una meta diferente con respecto al modelo de la derecha (inducción descriptiva). Por tanto, se emplean diferentes heurísticas y criterios de evaluación y aplicaciones en ambos tipos de aprendizaje.

1.4. Aprendizaje supervisado descriptivo

En los apartados anteriores hemos introducido, la minería de datos predictiva y descriptiva. Sin embargo, incluso si se supervisa el entorno de aprendizaje (se proporcionan pares de datos entrada-salida), el objetivo puede ser construir descripciones simbólicas destinadas a la interpretación humana. Este apartado aborda esta situación menos común, que es una combinación de aprendizaje supervisado y minería de datos descriptivos.

Entonces, podemos definir el aprendizaje supervisado descriptivo de la siguiente manera: Supongamos que existe un conjunto de datos, descrito por sus atributos y sus valores así como un atributo nominal seleccionado que es de interés (llamada atributo objetivo). El objetivo del aprendizaje supervisado descriptivo es inducir las reglas con el mismo formato que las reglas de clasificación para explicar la relación entre el atributo objetivo y los otros atributos en los datos.

En otras palabras, el aprendizaje supervisado descriptivo es un proceso de inducir un conjunto de reglas comprensibles en la forma de reglas de clasificación a partir

de datos etiquetados por clase. El formato de la regla de clasificación restringe la parte de las condiciones de las reglas para estar en forma conjunta y tener el atributo objetivo y uno de sus valores en la parte de las conclusiones de las reglas.

El propósito del aprendizaje supervisado descriptivo es permitir que el usuario, interesado en los datos, navegue a través de las reglas y, al hacerlo, obtenga información sobre el dominio de los datos. El objetivo final del aprendizaje supervisado descriptivo es permitir que el usuario comprenda los fenómenos subyacentes a los datos.

Podemos identificar tres modelos principales de aprendizaje supervisado descriptivo:

- **Minería de conjuntos de contraste** o *contrast mining set* es una forma de aprendizaje de reglas de asociación que busca identificar diferencias significativas entre grupos separados mediante la ingeniería inversa de los predictores clave que identifican para cada grupo en particular. Puede ser definido como "una conjunción de pares atributo-valor definida en grupos sin ningún atributo ocurriendo más de una vez". Ejemplos de búsqueda de diferencias:
    - Salud: ¿Qué diferencia hay en los síntomas entre enfermedades similares?
    - Marketing: ¿Qué diferencias hay entre los clientes que gastan menos dinero y los que gastan más en un determinado tipo de artículo?
    - Análisis de los datos del censo: ¿Cuál es la diferencia entre las personas que tienen títulos doctorado y personas con títulos de licenciatura?
- **Minería de patrones emergentes** o *emerging pattern mining*, puede ser definida como "patrones cuyas frecuencias en dos clases difieren en una gran proporción". La minería de patrones emergentes tienen el propósito de capturar tendencias emergentes en bases de datos con marca de tiempo, o capturar características diferenciadoras útiles entre clases de datos. La investigación posterior de patrones emergentes se ha centrado en gran medida en el uso de los patrones descubiertos para fines de clasificación.
- **Descubrimiento de subgrupos** o *subgroups discovery*, en el cual se centra este proyecto y en el cual entraremos en detalle más adelante.

La principal diferencia entre estas técnicas es que, mientras que los algoritmos de descubrimiento de subgrupos intentan describir distribuciones inusuales en el

espacio de búsqueda con respecto a un valor de la variable objetivo, los algoritmos de conjuntos de contraste y patrones emergentes buscan relaciones de los datos con respecto a los valores posibles de la variable objetivo. Los algoritmos de conjuntos de contraste intentan obtener altas diferencias de soporte entre los valores posibles y los algoritmos de patrones emergentes buscan patrones con diferentes frecuencias en dos clases de la variable objetivo. Estas dos últimas técnicas se basan en medidas de cobertura y precisión y el descubrimiento de subgrupos se enfoca en medidas de novedad e inusualidad.

1.5. Inteligencia computacional

1.5.1. Algoritmos evolutivos

Los algoritmos evolutivos son métodos de optimización y búsqueda de soluciones basados en los postulados de la evolución biológica. En ellos se mantiene un conjunto de entidades que representan posibles soluciones, las cuales se mezclan, y compiten entre sí, de tal manera que las más aptas son capaces de prevalecer a lo largo del tiempo, evolucionando hacia mejores soluciones cada vez.

En la comparación entre los algoritmos evolutivos y los métodos clásicos, los primeros son competitivos cuando el uso de estos últimos supone tiempos de ejecución prohibitivos, debido a una explosión combinatoria de posibles soluciones en el espacio de búsqueda (imposible realizar una búsqueda exhaustiva); igualmente, en aquellos casos en que los métodos clásicos son incapaces de dar una solución, los algoritmos evolutivos permiten obtener una solución que, aunque no sea la óptima, podría ser mejor que la obtenida hasta el momento para un problema dado o que la que podría producir un humano.

Por otro lado, si comparamos los algoritmos evolutivos con otras metaheurísticas, la principal ventaja de los primeros, comparados con aquellas metaheurísticas que hacen uso de una única solución, transformándola en pasos sucesivos (metaheurísticas basadas en trayectoria), es que la búsqueda de la solución en los algoritmos evolutivos está basada en población, lo que disminuye la posibilidad de éstos queden atrapados en un óptimo local. No obstante, también existen otro tipo de metaheurísticas, como las basadas en inteligencia de enjambre, que hacen uso de este tipo de búsqueda. Por el contrario, el inconveniente de las metaheurísticas que hacen uso de búsqueda basada en población es un mayor coste computacional. Sin embargo, esto último puede paliarse paralelizando la ejecución del algoritmo. Adicionalmente, la existencia de una gran variedad de algoritmos evolutivos facilita

el uso de la variante más adecuada en función de las características del problema a resolver.

Existe un amplio repertorio de problemas en donde los algoritmos evolutivos pueden ser competitivos:

- problemas en los que se quiere optimizar tanto los parámetros de un modelo como en aquellos otros donde se requiere la optimización del propio modelo y sus parámetros simultáneamente
- problemas de optimización tanto en espacios discretos/continuos
- problemas de optimización multiobjetivo
- problemas de optimización con restricciones

Los siguientes dos mecanismos forman la base de los sistemas evolutivos:

- Los operadores de variación (recombinación y mutación), que tienden a aumentar la variedad genética de la población y, por tanto, permiten crear la diversidad necesaria para poder alcanzar la solución a partir de una población inicial obtenida aleatoriamente. Se dice que este tipo de operadores favorecen la **exploración** del espacio de búsqueda
- El proceso de selección de supervivientes, que tiende a disminuir la variedad genética, pero actúa como una fuerza que empuja la convergencia hacia una solución.Se dice que este mecanismo favorece la **explotación** del espacio de búsqueda.

Otra propiedad importante los algoritmos evolutivos es su naturaleza estocástica, es decir, no son deterministas. Esto implica que el subsiguiente estado del sistema está determinado tanto por las acciones predecibles del proceso como por mecanismos aleatorios. Esta aleatoriedad se manifiesta principalmente en la toma de decisiones probabilista que subyace en la operativa de los operadores de variación. La principal consecuencia de esta propiedad se traduce en que la ejecución de un mismo algoritmo evolutivo, configurado con los mismos valores de parámetros, no tiene por qué dar resultados idénticos.

1.5.2. Lógica Difusa

Por lo general, la forma en que un experto humano representa el conocimiento es inherentemente cualitativa y vaga. Sin embargo, en alrededor de un 90% de los

casos de nuestra vida diaria estamos utilizando siempre adjetivos que cualifican situaciones, como «alto», «bajo», «caro», «barato», etc.

En este sentido, la lógica difusa o lógica borrosa permite la computación de conocimientos imprecisos y cualitativos, así como manejar la incertidumbre y tratar con naturalidad y en una medida razonable el razonamiento humano con etiquetas del lenguaje natural. Desde que fue propuesta en 1965 por Zadeh [38], se ha aplicado a múltiples áreas de investigación, fundamentalmente por su proximidad al razonamiento humano y porque proporciona una forma eficaz de captar la naturaleza aproximada e inexacta de la realidad.

En los procesos de inducción de reglas se incluye la Lógica Difusa de tal forma que los modelos extraídos son reglas difusas. En el tipo de reglas difusas más interpretables, las reglas difusas lingüísticas son por tanto las más apropiadas para la Minería de Datos. Y las variables continuas se definen así mismo como variables lingüísticas; es decir, variables que toman como valores posibles etiquetas lingüísticas, cuya semántica está representada por un conjunto borroso asociado [37]. La utilización de lógica difusa permite simplificar la dificultad de trabajar con variables de tipo numérico, aumentando la comprensibilidad de las reglas obtenidas.

En la lógica nítida, donde un elemento solo puede ser 0 absoluto o 1 absoluto, no es apropiada para describir la amplia mayoría de situaciones de la vida. Un ejemplo, el conjunto "hombres altos" es un conjunto al que pertenecerían los hombres con una estatura mayor que un cierto valor, que podemos establecer como 1,80 metros. Y todos los hombres con una altura inferior se consideran fuera de este conjunto. Así tendríamos que un hombre que mide 1,81 metros pertenecería a dicho conjunto, pero un hombre que mida 1,79 metros, no. Sin embargo, no es muy lógico decir que un hombre no es alto y otro no lo es cuando su altura difiere en unos meros 2 centímetros.

Por ello la lógica difusa establece rangos, y un valor de pertenencia, un número real entre 0 y 1, a cada rango. Así un hombre que mida 1,79 metros podría pertenecer al conjunto difuso (o etiqueta) "hombres altos" con un grado de 0,75. Mientras que un el hombre que mide 1,81 metros podría tener un grado de pertenencia de 0,80 a la vez que otro hombre que mide 1,50 metro y un grado de 0,1.

Visto desde esta perspectiva se puede considerar que la lógica clásica o nítida es un caso límite de la lógica difusa en el que se asigna un grado de pertenencia 1 a los hombres con una altura igual o mayor a 1,80 y un grado de pertenencia 0 a los que tienen una altura menor (figura 4).

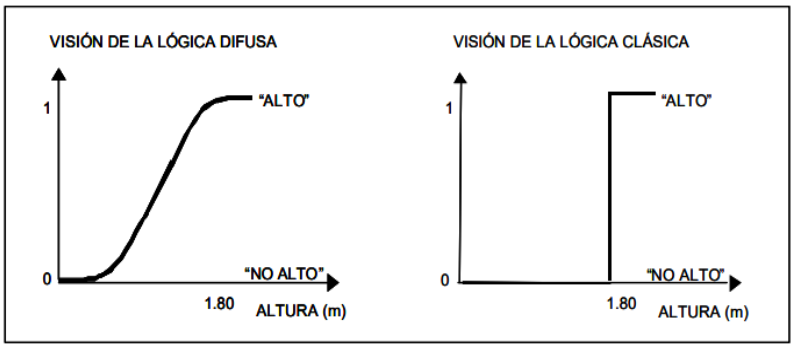


Figura 4. Visión de la lógica nítida frente a la difusa

Un punto a destacar de este ejemplo es que el grado de pertenencia a un conjunto difuso no tiene el mismo significado que el concepto de probabilidad. Si fuera según este último se referiría a "La probabilidad de que la persona sea Alta es de 0,41", es decir, se supone que la persona es alta (1) o no (0) teniendo un 41% de posibilidades de acertar en la afirmación. En el caso de la lógica difusa, nos referimos a que la persona es "más o menos" alta en otros términos, con un valor que corresponde al 0.41.

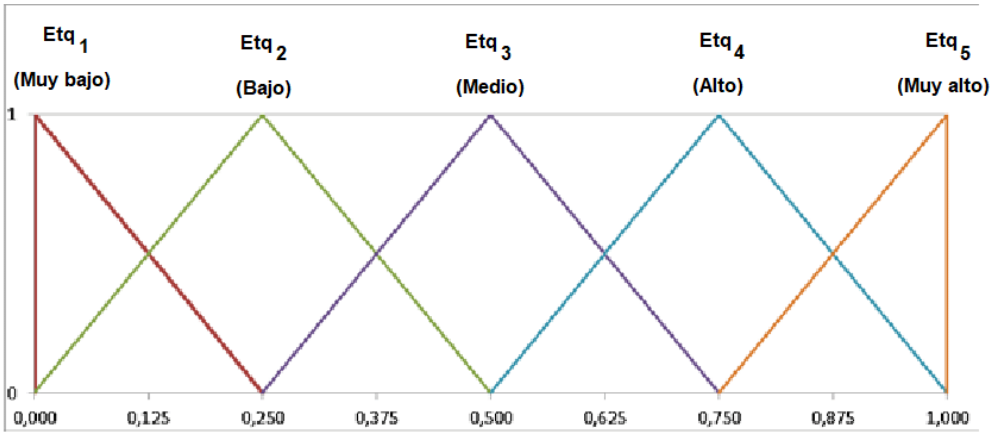


Figura 5: Representación de conjuntos difusos triangulares

Cuando no se dispone de conocimiento experto, se pueden utilizar particiones uniformes con funciones de pertenencia triangulares, como se muestra en la Fig. 5 para una variable con 5 etiquetas lingüísticas.

Al ser los conjuntos difusos una extensión de los conjuntos nítidos, las operaciones que se pueden realizar sobre estos son las mismas, es decir, se pueden intersecar (Figura 6 - Izquierda), unir (Figura 6 - Derecha) y negar conjuntos difusos.

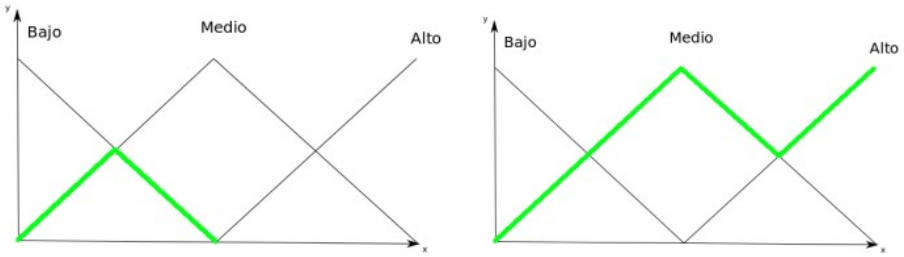


Figura 6. Intersección (Izqda.) y unión (dcha.) de conjuntos difusos

1.5.2.1. Reglas Lingüísticas

El objetivo del descubrimiento de subgrupos es extraer conocimiento sobre una variable de interés para el usuario, de una forma fácilmente interpretable.

Para la obtención de reglas, en caso de que una regla esté formada por una conjunción de variables, ésta calculará el grado de pertenencia con la intersección de las etiquetas de variables que la forman con una t-norma. Esta t-norma puede tener diferentes expresiones, pero las más utilizadas son la t-norma mínima y la t-norma producto.

Para una representación DNF (Disjunctive normal form), la cual veremos en mayor detalle más adelante (Sección 1.5.3.2), en una regla una misma variable estará conformada por la disyunción de una o más etiquetas de dicha variable. Para una regla DNF, el cálculo del grado de pertenencia de un ejemplo a una variable se realiza usando la unión difusa a través de la t-conorma, que normalmente es el valor máximo.

1.5.2.2. Grado de pertenencia a reglas de SGD

Dado un conjunto de datos conformado por un conjunto de ejemplos E = { e_1, e_2, … , e_n}. Cada ejemplo e_n = { V_1, V_2, … , V_n, $clase_i$} se representa como un conjunto de variables (que pueden ser numéricas o nominales) y valor para la clase objetivo. Y cada variable V_n = {v_1, v_2, …, v_m} está formada por un conjunto de valores v_m. Dicho valor puede ser nominal (de entre un número finito de valores) en caso de ser una variable nominal o una etiqueta difusa en caso de variables continuas.

Para poder calcular las medidas de calidad mencionadas en la Sección 2.3, es necesario averiguar, dada una regla, qué ejemplos están cubiertos y en que medida.

Para determinar si un ejemplo está cubierto por una regla se calculará el grado de pertenencia, un valor entre cero y uno, de dicho ejemplo a la regla. Por tanto, el grado de pertenencia de un ejemplo e_k a una regla R_i viene dado por la expresión APC (Antecedent Part Compatiblility):

$$APC(e, R_i) = T(TC(\mu^1_1(e_1), \dots \mu^1_n(e_k)), \dots, TC(\mu^k_1(e_1), \dots \mu^k_n(e_k)))$$

Ecuación 1: Antecedent Part Compatiblility

Donde:

- e_k indica el valor del ejemplo para la variable k.
- μ^k_n indica el grado de pertenencia a la etiqueta lingüística n de la variable k.
- TC es la t-conorma difusa seleccionada para representar el significado del operador OR -unión difusa-, en este caso es la t-conorma máxima.
- T es la t-norma difusa seleccionada para representar el significado del operador AND -intersección difusa-, en este caso es la t-norma mínima.

En caso de que la variable k sea nominal, si valor μ^k_n tiene el mismo valor que el que indique la regla será uno absoluto, en caso contrario será cero absoluto. En caso de ser numérica, μ^k_n se calculará según la función de pertenencia triangular:

$$\mu_{a,b,c}(x) = \begin{cases} 0, & si \quad x \leq a \\ \dfrac{x-a}{b-a}, & si \ a \leq x \leq b \\ \dfrac{x-a}{b-a}, & si \ a \leq x \leq b \\ 0, & si \quad c \leq x \end{cases}$$

Ecuación 2: Grado de pertenencia

Donde a, b, c definen el valor inferior, el punto intermedio y el valor superior del conjunto difuso, respectivamente.

Con esta definición, un ejemplo está cubierto por una regla si su valor APC con respecto a dicha regla es mayor a cero.

1.5.3. Esquema de representación

La representación genética de las soluciones es el aspecto más determinante de las características de cualquier propuesta de aprendizaje genético [42]. El enfoque “Cromosoma = Regla” (en el que cada individuo codifica una sola regla) es más adecuado en el descubrimiento de subgrupos porque el objetivo es encontrar un conjunto reducido de reglas en el que la calidad de cada regla se evalúa independientemente del resto. Este es el enfoque de codificación utilizado en la propuesta evolutiva que se describe a continuación.

Los algoritmos genéticos descubren una sola regla difusa cuyo consecuente está prefijado a uno de los posibles valores de la variable objetivo. Solo el antecedente está representado en el cromosoma y todos los individuos de la población están asociados con el mismo valor de la variable objetivo.

Puede operar con dos representaciones diferentes en función de nuestras necesidades: reglas canónicas (CAN) o reglas DNF.

1.5.3.1. Reglas canónicas (CAN)

Una regla canónica está formada una conjunción de valores R = {x_1, x_2, ..., x_k}. Donde k indica la variable a la que corresponde dicho valor. Al tratarse de una tarea de descubrimiento de subgrupos no es necesario incluir el consecuente en la representación.

Todos los individuos tienen una valor máximo fijo y en el caso de tener variables que no participen en la regla, estas contendrán un valor especial que será el número máximo de variables/etiquetas difusas posibles más uno.

Supongamos un caso, por ejemplo:

> Sea k = 5 variables de entrada, el número de etiquetas predefinido es 3. Las variables 1 y 2 son numéricas. Las variables 3, 4 y 5 son nominales. Las variables 3 y 4 tienen 3 posibles valores cada una, y la variable 5 tendrá 4 posibles valores. Sea la regla:
>
> SI X_1 = LL_1 & X_3 = $Valor_2$ & X_5 = $Valor_4$ ENTONCES $Clase_1$
>
> La representación del ejemplo se muestra en la siguiente figura:

X_1	X_2	X_3	X_4	X_5
1	4	2	4	4

Figura 7. Representación de una regla canónica

En este caso, significaría que ni la variable 2 ni la variable 4 intervienen en la regla. Sin embargo nota que a pesar de que $x_4 = x_5 = 4$, la variable 5 sí participa en la regla porque dicha variable nominal tiene cuatro valores posibles a diferencia de la variable 4.

1.5.3.2. Reglas DNF

Una regla difusa DNF, o forma normal disyuntiva, representa el conocimiento de forma flexible y compacta, permitiendo que cada variable tome más de un valor, y facilitando la extracción de reglas más generales. Además, las reglas difusas DNF lingüísticas nos permiten realizar cambios en la granularidad inicial en cada regla de manera descriptiva.

El antecedente completo se obtiene mediante la conjunción de cada una de las variables formadas por la disyunción de todos los valores de dicha variable. El siguiente es un ejemplo de regla difusa lingüística DNF:

SI Edad es *Medio* O *Alto* *Y* Estrés es *Alto* Y Género es *Varón* ENTONCES Calvicie es *Negativo*

Cabe señalar que, en las reglas DNF, cualquier subconjunto del conjunto completo de variables (con cualquier combinación de etiquetas lingüísticas relacionadas con el operador OR) puede formar parte del antecedente de la regla. De esta forma, un subgrupo es una descripción compacta e interpretable de patrones de interés en los datos.

En cuanto a la representación de las reglas DNF, toda la información relativa a una regla está contenida por varios cromosomas de longitud fija con representación binaria en el que, para cada variable (cromosoma), se almacena un bit para cada uno de los posibles valores de la variable; de esta forma, si el bit correspondiente contiene el valor 0 indica que el valor no se utiliza en la regla, y si el bit contiene el valor 1 indica que se incluye el valor correspondiente. Si una regla contiene todos los bits correspondientes a una característica con el valor 1, o todos contienen el valor 0, la característica se ignora y no forma parte de la regla.

El tamaño de un cromosoma dependerá de si la variable es nominal o numérica. Si la variable es nominal, el tamaño del cromosoma será igual al número de valores nominales que pueda tomar la variable, cada valor nominal será representada por un

gen. Si la variable es numérica, el tamaño del cromosoma será igual al número de etiquetas de dicha variable, cada etiqueta será representada por un gen.

El ejemplo anterior se podría representar como:

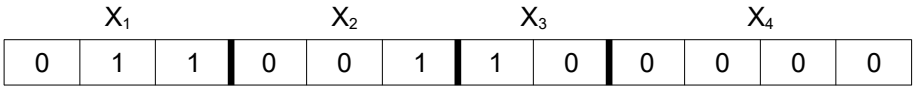


Figura 8. Codificación de una regla DNF

La regla anterior podría representarse de esta forma, siendo X_1 que representa la Edad, X_2 es el nivel de Estrés, X_3 es el Género y X_4 es la Raza de una persona (y puede tomar cuatro posibles valores).

Capítulo 2. Descubrimiento de subgrupos

2.1. Introducción

Como hemos mencionado en el capítulo anterior, el descubrimiento de subgrupos forma parte de los modelos de aprendizaje supervisado descriptivo.

El concepto inicial de descubrimiento de subgrupos fue originalmente presentado por W. Klösgen [21] y S. Wrobel [33], y posteriormente más formalmente definido, usando el nombre *Data Surveying*, por A. Siebes [48]. Wrobel definió el descubrimiento de subgrupos interesantes como [49]:

> *"En el descubrimiento de subgrupos, dada una llamada población de individuos y una propiedad de esos individuos en los que estamos interesados. La tarea del descubrimiento de subgrupos es por tanto, descubrir los subgrupos de la población que son estadísticamente 'más interesantes'."*

En otras palabras, la finalidad del descubrimiento de subgrupos es, la búsqueda de subgrupos tan grandes como sea posible y que tengan las características estadísticamente (distributivamente) más inusuales con respecto a la propiedad de interés.

Podemos describir a los subgrupos como conjunciones de rasgos (pares atributo-valor) que son característicos para una clase seleccionada de individuos (propiedad de interés). Una descripción de subgrupos puede ser vista como la parte condición de una regla [Descripcion_de_subgrupo → Clase]. Donde el consecuente, Clase, representa al $valor_{objetivo}$ para una única variable de interés ($Variable_{objetivo}$) en la tarea de descubrimiento de subgrupos. Y el antecedente, Descripcion_de_subgrupo, supone una condición cuyo cumplimiento describe una distribución estadística inusual con respecto a $Valor_{objetivo}$. Por tanto, el descubrimiento de subgrupos puede ser visto como un caso especial de una tarea de aprendizaje de reglas más general.

Como un modelo de aprendizaje supervisado descriptivo, en el modelo de Descubrimiento de Subgrupos las reglas que se generan no son utilizadas para predecir, sino para describir el comportamiento de los ejemplos del conjunto de datos en relación a un sólo valor de la variable objetivo. Por lo tanto, para poder cubrir todos los valores de la variable objetivo, es necesario ejecutar el algoritmo una vez para cada valor de ésta variable objetivo.

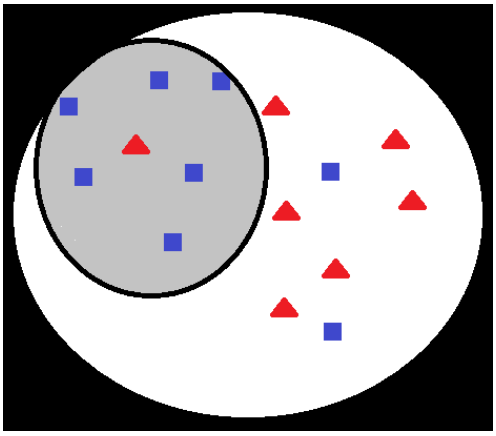


Figura 9: Representación de un subgrupo

En la figura de arriba se representa un ejemplo de una regla de descubrimiento de subgrupos. En esta regla, la variable objetivo puede tener dos posibles valores: Cuadrados y Triángulos. En el ejemplo, esta regla o subgrupo cubre un conjunto con un alto porcentaje de valores "Cuadrado". Pero como se puede ver el subgrupo no cubre todos los ejemplos con valor "Cuadrado", y de hecho, cubre ejemplos que no son correctos. Sin embargo la forma de la función es simple y muy interpretable, y de este modo el algoritmo ve su complejidad muy reducida.

Los algoritmos que se clasifican dentro del modelo de Descubrimiento de Subgrupos deben de tener las siguientes características:

- Tipo de variable objetivo: Las variables objetivo podrán ser de tipo binario (poseen solo dos valores, típicamente 0 o 1), nominales (o categóricos, donde la variable puede contener valores dentro de entre una lista de n valores alfanuméricos), o bien numéricas (la variable puede contener valores dentro de un rango numérico). En nuestro proyecto, nos centraremos solo en variables objetivo de tipo binario y nominal.
- Forma de representación: Para una fácil interpretabilidad, la forma en que representemos las reglas debe ser lo más sencilla y clara posible. Por ello generalmente o bien se utilizan reglas compuestas por conjunciones de pares atributo-valor, llamémoslas canónicas, o bien en forma normal disyuntiva (DNF). Además, el uso de la lógica difusa incrementa aún más su interpretabilidad.
- Medidas de calidad: Son las herramientas de medición en las que nos basamos para decidir en que magnitud es útil o "interesante" es un subgrupo.

Diferentes algoritmos utilizarán diferentes medidas de calidad, las cuales veremos más adelante.

- Estrategia de búsqueda: En una búsqueda en profundidad, el espacio de búsqueda aumenta de manera exponencial según el número de variables involucradas. Esto no es nada eficiente, por ello se necesitan técnicas (metaheurísticas) que reduzcan dicho espacio de búsqueda, a la vez que mantiene la capacidad de encontrar buenas soluciones. Para ello, nos centraremos en los algoritmos evolutivos de descubrimiento de subgrupos para nuestra librería.

Pongamos un ejemplo de un caso, sea D un conjunto de datos con tres variables:

Edad = {menos de 25, de 25 a 40, de 40 a 60, más de 60},

Género = {Hombre, Mujer},

Estado civil = {Soltero, Casado, Viudo, Divorciado, Otro}

y además definimos una variable de interés o variable objetivo:

Graduado en una universidad = {Si, No}

Algunas posibles reglas conteniendo descripciones de subgrupos son:

R_1 (SI Edad = 'menos de 25' Y Estado civil = 'Casado') $\rightarrow$ Graduado = 'Si'

R_2 (SI Género = 'Mujer' Y País = 'Viudo') $\rightarrow$ Graduado = 'No'

donde la regla R_1 representa un subgrupo de personas con estado civil soltero con menos de 25 años de edad, para las cuales la probabilidad de haberse graduado en una universidad es inusualmente alta con respecto al resto de la población. Y la regla R_2 representa que las mujeres con estado civil de viudedad tienen una mayor probabilidad de no haberse graduado de una universidad comparado al resto de la población.

2.2. Principales algoritmos de descubrimiento de subgrupos

En esta sección, se describirán los principales métodos de descubrimiento de subgrupos desarrollados. Los métodos existentes de descubrimiento de subgrupos

pueden ser clasificados ampliamente en tres grandes categorías según su estrategia de búsqueda de candidatos [57]. Estas tres categorías son: algoritmos basados en búsqueda por haz, algoritmos basados en búsqueda exhaustiva y algoritmos evolutivos.

La Tabla 1 muestra los principales algoritmos para el descubrimiento de subgrupos que veremos a continuación y compara algunas de sus características.

Estrategia de búsqueda	Método	Técnica de poda	Medidas de calidad
Por Haz	SubgroupMiner	Soporte mínimo	Prueba binomial
	SD	Soporte mínimo	Cociente de generalización
	CN2-SD	Restricción	Inusualidad
	RSD	Soporte mínimo	Inusualidad, Significancia o Cobertura
	DSSD	Cobertura mínima	Inusualidad, Chi cuadrado, Prueba media o WKL
Exhaustiva	EXPLORA	Sin poda	Generalidad, Redundancia o Simplicidad
	MIDOS	Soporte mínimo y estimación optimista	Novedad o Inusualidad distributiva
	APRIORI-SD	Soporte mínimo	Inusualidad
	SD-Map	Soporte mínimo y cobertura	Piatetsky-Shapiro, Inusualidad o Prueba binomial
	DpSubgroup	Estimación optimista	Piatetsky-Shapiro o Chi cuadrado
	MergeSD	Restricción	Piatetsky-Shapiro
Evolutiva	SDIGA	Sin poda	Confianza, Soporte, Inusualidad, Sensibilidad, Significancia o Interés
	MESDIF	Sin poda	Confianza, Soporte, Inusualidad, Sensibilidad, Significancia
	NMEEF-SD	Sin poda	Confianza, Soporte, Inusualidad, Sensibilidad, Significancia
	FuGePSD	Sin poda	Confianza, Soporte, Inusualidad, Sensibilidad
	CGBA-SD	Sin poda	Confianza, Soporte

Tabla 1: Clasificación de los algoritmos para el descubrimiento de subgrupos

2.2.1. Algoritmos basados en búsqueda por haz

Estos algoritmos generan un número fijo de candidatos (especificado por el parámetro de ancho de haz) durante cada iteración. En cada nivel, un operador de refinamiento genera candidatos para el siguiente nivel buscando entre los candidatos contenidos en el haz. Aunque la búsqueda por haz es más rápida en

comparación con otras estrategias de búsqueda, solo explora una parte del espacio de búsqueda y no garantiza una solución óptima. Hasta ahora se han desarrollado varios algoritmos basados en la búsqueda por haz. Los más populares son descritos abajo:

- **SubgroupMiner** [24] es una extensión de EXPLORA y MIDOS. Es un sistema de descubrimiento de subgrupos avanzado que usa reglas de decisión y búsqueda interactiva en el espacio de soluciones, permitiendo el uso de enormes bases de datos, hipótesis multi-relacionales y el descubrimiento de estructuras de subgrupos causales.
  Puede manejar atributos objetivo nominales y numéricos, pero para variables numéricas es necesario realizar una discretización previa. Este es el primer algoritmo que considera el uso de variables objetivo numéricas. Este algoritmo puede usa como medida de calidad el test binomial clásico para verificar si la distribución estadística del objetivo es significativamente diferente en el subgrupos extraído.

- **SD** [36] es un sistema de inducción de reglas guiado por conocimiento experto en lugar de simplemente usar una medida para buscar y seleccionar el mejor subgrupo. El algoritmo genera subgrupos de acuerdo con un ancho de haz fijo. En cada iteración de este algoritmo, la descripción del subgrupo se actualiza agregando nuevos valores de atributo. Los subgrupos descubiertos deben mantener una frecuencia mínima y deben ser relevantes para la aceptación, intentando buscar subgrupos que cubran el mayor número posible de ejemplos del valor objetivo y el mínimo número de ejemplos que no pertenezcan al valor objetivo.

- **CN2-SD** [19], este algoritmo es una versión modificada del algoritmo CN2 [26] para la tarea de descubrimiento de subgrupos. A diferencia de CN2 usa una modificación de la precisión relativa ponderada (inusualidad) como medida de calidad para hacer un ranking de subgrupos. CN2-SD implementa un esquema de cobertura diferente en comparación con el algoritmo estándar CN2. En CN2, cuando se encuentra una nueva regla, se eliminan todos los ejemplos cubiertos por la regla para evitar generar la misma regla nuevamente. En CN2-SD, los ejemplos cubiertos no se eliminan del conjunto de datos de entrenamiento; más bien, se almacena un recuento con cada ejemplo que indica la frecuencia con la que se ha cubierto el ejemplo hasta el momento. Si un ejemplo está cubierto por una regla, su peso disminuye y la calidad de la regla se mide considerando el nuevo peso de este ejemplo.

- **RSD** [52, 53], *Relational Subgroup Discovery*, este algoritmo es una actualización del algoritmo CN2-SD el cual permite el descubrimiento de subgrupos relacionales. Este método extrae subgrupos de una base de datos relacional eliminando características irrelevantes. Implementa cobertura ponderada modificada y heurística ponderada WRAcc (Precisión relativa ponderada) para extraer subgrupos. Emplea la inusualidad como medida de calidad para clasificar los subgrupos.

- **DSSD** [50]. DSSD es un algoritmo de descubrimiento de subgrupos no redundante basado en la búsqueda de haces. Implementa tres estrategias de selección heurística diferentes para gestionar la redundancia. DSSD asume que en un conjunto de subgrupos no redundantes, todos los pares de subgrupos deben ser diferentes en 1) descripción de subgrupos, 2) cobertura de subgrupos o 3) modelos excepcionales. DSSD genera una gran cantidad de j subgrupos en la fase inicial. En la segunda fase, estos j subgrupos se refinan incorporando la poda de dominancia que elimina la condición de subgrupo uno por uno si no disminuye la calidad de este subgrupo. En la fase final, se utiliza una estrategia de selección de subgrupos para seleccionar los k mejores subgrupos de los j subgrupos generados inicialmente. DSSD usa WRAcc, Chi cuadrado, prueba media o la divergencia ponderada KullbackLeibler para clasificar los subgrupos.

Los diferentes métodos utilizan diferentes tipos de heurística para extraer y evaluar subgrupos. SubgroupMiner usa la prueba binomial para clasificar los subgrupos, mientras que SD emplea el cociente de generalización para evaluarlos. Tanto CN2-SD como RSD implementan un esquema de cobertura modificado y WRAcc modificado para generar y clasificar subgrupos, aunque CN2-SD es un algoritmo de descubrimiento de subgrupos proposicional y RSD es un algoritmo de descubrimiento de subgrupos relacional. La incorporación del esquema de cobertura ponderado permite que estos métodos administren la redundancia hasta cierto punto y aumenten la diversidad en los subgrupos generados. DSSD implementa tres estrategias de selección de subgrupos para reducir el número de subgrupos redundantes.

2.2.2. Algoritmos basados en búsqueda exhaustiva

Estos algoritmos buscan todos los candidatos posibles y eliminan los subgrupos irrelevantes de acuerdo con algunas restricciones para reducir el espacio de hipótesis. Esta es una técnica de búsqueda muy utilizada por su facilidad de implementación; aún así, no es factible usarlo cuando el número de soluciones

candidatas crece rápidamente con el tamaño del problema. A continuación se describen algunos de los algoritmos de descubrimiento de subgrupos basados en la búsqueda exhaustiva ampliamente utilizados.

- **EXPLORA** [21] fue la primera aproximación desarrollada para descubrimiento de subgrupos. Implementa árboles de decisión para generar subgrupos. Las reglas se especifican definiendo un esquema descriptivo e implementando un método de verificación estadística. Para medir el interés de una regla utiliza medidas estadísticas, tales como evidencia, generalidad, redundancia o simplicidad, entre otras. El método de búsqueda aplicado puede ser heurístico o exhaustivo, pero sin realizar poda.

- **MIDOS** [33] aplica la aproximación de EXPLORA, siendo el primer algoritmo de descubrimiento de subgrupos que extrae subgrupos interesantes de bases de datos multi-relacionales. MIDOS usa una estimación optimista usando poda basada en soporte mínimo en variables objetivo binarias para reducir el espacio de búsqueda. El objetivo es descubrir subgrupos de la relación objetivo que tengan distribuciones estadísticas inusuales con respecto a la población completa. Las medidas de calidad utilizadas son una combinación de inusualidad y tamaño.
- **Apriori-SD** [54]: Es una extensión del algoritmo de aprendizaje de reglas de clasificación Apriori-C [28] que, a su vez, es una modificación del algoritmo de aprendizaje de reglas de asociación APRIORI para el descubrimiento de subgrupos. Cada subgrupo debe mantener un umbral mínimo de soporte y confianza para tener su lugar en el conjunto de resultados final. Los subgrupos descubiertos pasan por un post-procesamiento utilizando la precisión relativa ponderada (inusualidad) como medida de calidad. Todos los ejemplos positivos cubiertos por una regla no se eliminan del conjunto de datos de entrenamiento, sino que cada vez que se cubre un ejemplo, su peso disminuye. Un ejemplo se elimina solo cuando su peso cae por debajo de un umbral determinado o cuando un ejemplo se ha cubierto más de k veces.

- **SD-Map** [46]: SD-Map es un método exhaustivo de descubrimiento de subgrupos que es una extensión del método FP-growth [30] basado en minería de reglas de asociación. Utiliza un paso modificado en FP-growth que permite calcular la calidad de un subgrupo directamente sin referenciar a resultados previos. Selecciona solo aquellos subgrupos que mantienen un valor de soporte mínimo dentro del conjunto de datos. Implementa una búsqueda en profundidad para la generación de candidatos y también puede

manejar los valores faltantes para diferentes dominios. Utiliza varias medidas de calidad como Piatetsky-Shapiro [21] y la inusualidad.

- **DpSubgroup** [45]: Este algoritmo de descubrimiento de subgrupos implementa una estructura basada en árboles FP introducida por SD-Map para generar subgrupos. Utiliza una poda de estimación optimista estricta para eliminar subgrupos irrelevantes. Este algoritmo incorpora la estimación optimista para Piatetsky-Shapiro y la prueba Chi-cuadrado de Pearson. Se centra en manejar atributos objetivo nominales y binarios.

- **MergeSD** [44]: Este algoritmo emplea una estrategia de primera profundidad para buscar los subgrupos candidatos. Implementa la poda basada en restricciones en los subgrupos en intervalos superpuestos. Este algoritmo puede funcionar en los conjuntos de datos que tienen atributos numéricos mediante la poda de gran parte del espacio de búsqueda al explotar los límites entre las descripciones de subgrupos numéricos relacionados.

MIDOS es una adaptación del método EXPLORA que tiene como objetivo extraer subgrupos estadísticamente inusuales en bases de datos multirelacionales. APRIORI-SD, SD-Map, DpSubgroup y MergeSD son extensiones de diferentes algoritmos de asociación. APRIORI-SD es una adaptación para el descubrimiento de subgrupos del algoritmo de aprendizaje de reglas de clasificación APRIORI-C, que es una modificación del algoritmo de aprendizaje de reglas de asociación APRIORI, emplea una estrategia de prueba y generación de candidatos. SD-Map, DpSubgroup y MergeSD son las extensiones de los métodos basados en FP-growth que utilizan la estrategia divide y vencerás para buscar en el espacio de hipótesis. Todos ellos usan árboles de decisión para la representación. Solo Merge-SD Y SD-Map pueden manejar variables numéricas o continuas, en cuanto al resto, necesitan una discretización previa. Nota que el esquema de discretización utilizada afecta a los resultados obtenidos dependiendo del problema a ser resuelto.

2.2.3. Algoritmos evolutivos

La heurística empleada por este tipo de algoritmos está definida por una función de calidad que determina que individuos (reglas en este caso) serán seleccionados para formar parte de las nuevas poblaciones en procesos competitivos. Este tipo de heurística de búsqueda sigue el proceso de evolución natural como herencia, mutación, selección y cruce. A pesar de que generalmente estos algoritmos no obtienen una solución global óptima, son capaces de obtener una solución local

óptima lo suficientemente buena, alcanzando un compromiso entre eficacia y eficiencia.

A continuación se detallan los algoritmos evolutivos para descubrimiento de subgrupos presentados:

- **SDIGA** [10]: Utiliza un algoritmo genético como núcleo, hace uso de lógica difusa para tratar mejor con valores numéricos y permite utilizar diversas medidas para calcular la calidad de los individuos. Aplica un esquema iterativo para la extracción de reglas descriptivas difusas. Evalúa las reglas realizando una media ponderada de algunas medidas de calidad que pueden incluir confianza, soporte e inusualidad.

- **MESDIF** [11]: Emplea un algoritmo genético multiobjetivo para la extracción de reglas descriptivas difusas. Está basado en el enfoque SPEA2 [31] que aplica conceptos de elitismo para la selección de reglas. Permite también una gran diversidad de medidas de calidad para evaluar los subgrupos generados como confianza, soporte e inusualidad.

- **NMEEF-SD** [12]: Este algoritmo es un sistema evolutivo de inducción de reglas difusas multiobjetivo. Incorpora el uso de algunos operadores especiales para generar subgrupos de alta calidad y promover la diversidad. Este algoritmo se basa en un enfoque de clasificación no dominada NSGA-II [32] (Algoritmo genético de clasificación no dominada II) cuya estrategia de búsqueda se basa en la ordenación por dominancia y el uso de elitismo. Para la extracción y evaluación de subgrupos, este método puede emplear varias medidas de calidad, que pueden incluir confianza, soporte e inusualidad.

- **FuGePSD** [55]: Algoritmo de programación genético que emplea una aproximación genética de cooperación-competición donde la población compite entre sí para obtener una solución óptima. A lo largo del proceso evolutivo, la población tendrá un número variable de individuos, siendo estos de longitud variable a su vez.

- **CGBA-SD** [40]: Este algoritmo representa un enfoque de programación genética guiada por la gramática para el descubrimiento de subgrupos. De acuerdo con este método, cada regla se representa como un árbol de derivación que muestra una solución representada usando el lenguaje denotado por la gramática. El algoritmo consta de los mecanismos para adaptar la diversidad de la población mediante la autoadaptación de las probabilidades de recombinación y mutación.

Los algoritmos evolutivos desarrollados hasta el momento implementan una hibridación de algoritmo genético y sistema difuso, conocido como Sistema Difuso Evolutivo (Evolutive Fuzzy System o EFS). EFS proporciona herramientas nuevas y útiles para analizar patrones y descubrir conocimientos útiles.

Tal como se ha mencionado anteriormente, nuestra librería implementa los algoritmos evolutivos SDIGA, MESDIF, NMEEF-SD y FuGePSD, por ello los explicaremos más detalladamente más adelante.

2.3. Medidas de calidad para el descubrimiento de subgrupos

Uno de los aspectos más importantes de cualquier algoritmo de descubrimiento de subgrupos, y un factor determinante en la calidad del enfoque, es la medida de calidad a utilizar, tanto para seleccionar las reglas como para evaluar los resultados del proceso. Las medidas objetivas para la inducción descriptiva evalúan cada subgrupo individualmente, pero pueden complementarse con sus variantes para calcular la media del conjunto inducido de descripciones de subgrupos, lo que permite la comparación entre diferentes algoritmos de descubrimiento de subgrupos.

Existen diferentes estudios sobre medidas objetivas de calidad para el proceso de inducción descriptivo [36], [39], [41] pero es difícil llegar a un acuerdo sobre su uso. A continuación, se describen las medidas objetivas de calidad más utilizadas (no son todas) en la bibliografía especializada de descubrimiento de subgrupos y que utilizaremos en nuestra implementación, clasificadas por su meta principal tal como complejidad, generalidad, precisión, interés e híbridas:

2.3.1. Medidas de complejidad

Estas medidas de calidad se utilizan para cuantificar la calidad de las reglas individuales de acuerdo con los patrones individuales de interés cubiertos. Las medidas de calidad de este tipo se pueden observar a continuación:

- **Número de reglas** (N_{reg}): Determina el número de subgrupos generados. Puede ser calculado como el número medio de reglas obtenidas por clase.

- **Número de variables** (N_{var}): Indica el número medio de variables por regla generada.

2.3.2. Medidas de generalidad

Estas medidas de calidad muestran la precisión de los subgrupos y son ampliamente utilizadas en la extracción de reglas de asociación y clasificación. Dentro de este grupo se pueden encontrar:

- **Soporte** (Support): Mide el grado de cobertura que la regla ofrece a los ejemplos que pertenecen a la clase especificada en la regla consecuente. Considera el número de ejemplos que satisfacen ambas partes de la regla: antecedente y consecuente. Lavrac en [19] computa el soporte general como:

$$Support(R^i) = Support(Cond^i \rightarrow Target_{value}) = P(Cond^i \cup Target_{value}) = \frac{n(Target_{value}.Cond^i)}{n_s}$$

Ecuación 3: Soporte

  Donde $n(Target_{value} \cup Cond^i)$ es el número de ejemplos que satisfacen las condiciones del antecedente ($Cond^i$) y simultáneamente pertenecen al valor para la variable objetivo ($Target_{value}$) indicada en la parte del consecuente de la regla. n_s es el número total de ejemplos en nuestro conjunto de datos.

- **Cobertura** (Coverage) [36]: Mide el porcentaje de ejemplos cubiertos de media por una regla R^i del conjunto de reglas inducido en relación al total de ejemplos:

$$Cov(R^i) = Cov(Cond^i \rightarrow Target_{value}) = P(Cond^i) = \frac{n(Cond^i)}{n_s}$$

Ecuación 4: Cobertura

  Donde $n(Cond^i)$ es el número de ejemplos que verifican la condición $Cond^i$ descrita en los antecedentes (independientemente de la clase a la que pertenezca). Como antes n_s es el número total de ejemplos.

2.3.3. Medidas de precisión

Estas medidas de calidad muestran la precisión de los subgrupos y son ampliamente utilizadas en la extracción de reglas de asociación y clasificación. Dentro de este grupo se pueden encontrar:

- **Confianza** (Confidence): Determina la precisión de la regla, ya que refleja el grado en que los ejemplos dentro de la zona del espacio determinado por el antecedente verifican la información especificada en el consecuente de la regla.

  Mide la frecuencia relativa de ejemplos que satisfacen la regla completa entre aquellos que satisfacen solo el antecedente. Esto se puede calcular como [29]:

$$Conf(R^i) = Conf(Cond^i \rightarrow Target_{value}) = P(Target_{value} | Cond^i) = \frac{n(Target_{value}.Cond^i)}{n(Cond^i)}$$

Ecuación 5: Confianza

2.3.4. Medidas de interés

Estas medidas están destinadas a seleccionar y clasificar patrones según su interés potencial para el usuario. Dentro de este tipo de medidas se puede encontrar:

- **Significancia**: Indica cuan significativo o novedoso es un hallazgo, es medido por el índice de probabilidad de una regla:

$$Sign(R_i) = Sign(Cond^i \rightarrow Target_{value}) = 2 \cdot \sum_{j=1}^{n_c} n(Target_{value}.Cond^i) \cdot \log\left(\frac{n(Target_{value}.Cond^i)}{n(Target_{value}) \cdot p(Cond^i)}\right)$$

Ecuación 6: Significancia

  Donde n_c es el número de valores para la variable objetivo y $p(Cond^i)$, computado como $n(Cond^i)/n_s$, es usado como un factor de normalizado.

Hay otras medidas en esta categoría como la medida de Interés (Interest) o la de Novedad (Novelty).

2.3.5. Medidas híbridas

En este grupo se puede encontrar un gran número de medidas de calidad porque el descubrimiento de subgrupos intenta obtener un compromiso entre generalidad, interés y precisión en los resultados obtenidos. Entre las diferentes medidas de calidad que se pueden encontrar en esta categoría se encuentran la Sensitividad (Sensitivity), Falsa Alarma (False Alarm), Especificidad (Specificity) o Inusualidad (Unusualness), de las cuales utilizaremos:

- **Inusualidad o precisión relativa ponderada** (Unusualness o WRAcc) [22]: También llamado atipicidad, se define como la ponderación de la precisión relativa de la regla intentando establecer un equilibrio entre cobertura y precisión. Es una medida de calidad especialmente importante para el descubrimiento de subgrupos. De hecho en [58] propone la posibilidad de encontrar una visión unificada sobre el aprendizaje supervisado descriptivo a través de esta medida. Esta medida se calcula del siguiente modo:

$$WRAcc(R^i) = WRAcc(Cond^i \rightarrow Class_j) = Cobertura(R^i) \cdot (Confianza(R^i) - \frac{n(Target_{value})}{n_s})$$

$$WRAcc(R^i) = \frac{n(Cond^i)}{n_s} \cdot (\frac{n(Target_{value} \cdot Cond^i)}{n(Cond^i)} - \frac{n(Target_{value})}{n_s})$$

Ecuación 7: WRAcc

La atipicidad de una regla puede ser descrita como el balance entre la cobertura de una regla ($p(Cond^i)$) y su ganancia de precisión ($p(Target_{value}.Cond_i) - p(Target_{value})$). Debido a esta ganancia de precisión, es posible que WRAcc < 0 cuando Ri tiene una baja calidad, porque obtiene un mayor porcentaje de clases negativas que positivas. De este modo, una regla sólo puede ser considerada interesante cuando obtiene un WRAcc positivo.

Sin embargo, es conveniente estandarizar el valor del WRAcc a WRAcc normalizado (WRAccN). Con esto se consigue una mejora respecto a la homogeneización de esta medida de calidad.

$$LB_{WRAcc} = (1 - \frac{n(Target_{value})}{n_s}) \cdot (0 - \frac{n(Target_{value})}{n_s})$$

$$UB_{WRAcc} = \frac{n(Target_{value})}{n_s} \cdot (1 - \frac{n(Target_{value})}{n_s})$$

$$WRAccN = \frac{WRAcc(R^i) - LB_{WRAcc}}{UB_{WRAcc} - LB_{WRAcc}}$$

Ecuación 8: WRAccN

Donde LBWRAcc Y UBWRAcc representan el límite inferior y superior respectivamente. El WRAccN se normaliza al intervalo [0, 1] a través de la ecuación. De este modo, una regla sólo puede ser considerada interesante cuando obtiene un WRAccN superior a 0,5.

- **Sensibilidad o TPR**. Ésta medida mide la proporción de resultados positivas correctamente clasificadas. La sensibilidad tiene un componente basado en la generalidad. Se computa como:

$$Sens(R^i) = Sens(Cond^i \rightarrow Target_{value}) = \frac{n(Target_{value} \cdot Cond^i)}{n(Target_{value})} = \frac{TP}{TP+FN}$$

Ecuación 9: Sensibilidad

Ésta medida de calidad puede ser encontrada en la literatura como *Soporte basado en ejemplos de clase*, *Recall* o *TPRate*. Y su dominio es [0,1].

  - **TP (True Positive)**: Los verdaderos positivos se refiere aquellos resultados positivos correctamente clasificados.

  - **FN (False Negative)**: Los falsos negativos se refiere aquellos resultados negativos erróneamente clasificados.

		PREDICCIÓN	
		POSITIVO	NEGATIVO
CLASE REAL	POSITIVO	Verdadero Positivo (TP)	Falso Negativo (FN)
	NEGATIVO	Falso Positivo (FP)	Verdadero Negativo (TN)

Figura 10: Relación entre predicción y clase real

- **Especificidad o FPR**. Ésta medida mide la proporción de resultados negativos correctamente clasificados. Se computa como:

$$Espec(R^i) = Espec(Cond^i \rightarrow \overline{Target_{value}}) = \frac{n(\overline{Target_{value}} \cdot Cond^i)}{n(\overline{Target_{value}})} = \frac{TN}{TN+FP}$$

Ecuación 10: Especificidad

  - **TN (True Negative)**: Los verdaderos negativos se refiere aquellos resultados negativos correctamente clasificados.

- **FP (False Positive)**: Los falsos positivos se refiere aquellos resultados positivos erróneamente clasificados.

2.4. Estudio de librerías y plataformas existentes para el descubrimiento de subgrupos

Este apartado tiene que ver con la motivación del proyecto, y es que hasta ahora ha habido dos métodos para la tarea de descubrimiento de subgrupos para R:

- El primer método es un paquete denominado “rsubgroup” el cual implementa una interfaz de R para utilizar la herramienta de minería de datos VIKAMINE [6].
- El segundo método trata la tarea de descubrimiento de subgrupos de manera nativo por medio de una librería llamada SDEFSR.

La principal diferencia entre ambos métodos es que rsubgroup es una pasarela para utilizar la herramienta VIKAMINE desde R, mientras que la librería SDEFSR está implementada directamente en R, por lo que no tiene dependencias con lenguajes y herramientas externas.

En cuanto a los algoritmos implementados rsubgroup contiene los algoritmos Beam-Search, Bit-based Subgroup Discovery (BSD), SD-Map* y SD-Map. Mientras que SDEFSR implementa cuatro algoritmos, distintos a los que se encuentran en VIKAMINE, que son SDIGA, MESDIF, NMEEF-SD y FuGePSD.

Éste proyecto de hecho implementa tres de los cuatro algoritmos contenidos en SDEFSR. La principal diferencia entre ellos está en que los algoritmos de la librería SDEFSR están implementados en lenguaje R, mientras que nuestra implementación se realizará en lenguaje C++ a través del paquete RCPP, el cual describiremos en mayor detalle más adelante. Esto debería proporcionar una mayor eficiencia con respecto a la librería SDEFSR. En la sección 4.3.2.4 se muestra una comparación de la eficiencia entre ambos lenguajes.

Así mismo, estos algoritmos que se han realizado se encuentran también implementados en Java y están integrados en la herramienta de minería de datos de código abierto KEEL [9]. Por lo que tomaremos como referencia el código fuente de estos algoritmos para realizar el nuestro.

Capítulo 3. Algoritmos evolutivos para el descubrimiento de subgrupos

Como se ha mencionado anteriormente, los algoritmos de extracción de reglas evolutivos son: SDIGA, MESDIF, NMEEF-SD y FuGePSD. Los cuales son los algoritmos que se ha decidido que se implementen. En esta sección describiremos en detalle sus características.

3.1. Algoritmo SDIGA (Subgroup Discovery Iterative Genetic Algorithm)

SDIGA es un Sistema Difuso Evolutivo porque utiliza reglas difusas de representación del conocimiento y computación evolutiva como proceso de aprendizaje. Este algoritmo es capaz de trabajar con valores perdidos, con datos nominales, binarios y continuos. Es interesante remarcar que SDIGA busca reglas para cada valor de la variable objetivo, es decir, el consecuente no necesita ser representado en el cromosoma sino que es fijo. Además de que para dicha variable objetivo solo se admite valores nominales

Este algoritmo sigue el enfoque de aprendizaje de reglas iterativas donde la solución de cada iteración es el mejor individuo obtenido y la solución global está formada por los mejores individuos obtenidos en las diferentes ejecuciones. La representación de los individuos se realiza a través del enfoque 'Cromosoma = Regla' y el núcleo de SDIGA es un algoritmo evolutivo que utiliza un paso de post-procesamiento basado en una búsqueda local. Este algoritmo híbrido extrae una regla difusa simple e interpretable con un nivel adecuado de soporte y confianza. El modelo de algoritmo puede usar reglas canónicas difusas o DNF con un conjunto predefinido de etiquetas.

Este algoritmo se incluye en un proceso iterativo para la extracción de diferentes reglas. De esta forma, el algoritmo marca la cobertura de ejemplos para las reglas para evitar que se obtenga una nueva regla que cubra exactamente los mismos ejemplos en las siguientes ejecuciones. El algoritmo produce reglas mientras que las reglas generadas alcanzan un nivel mínimo de confianza y brindan información sobre áreas del espacio de búsqueda en las que hay ejemplos no descritos por las reglas generadas en iteraciones anteriores.

La regla se mejora en una fase de post-procesamiento a lo largo de un proceso de escalada, que modifica la regla para aumentar el grado de soporte.

El fitness es una función de agregación en la que el usuario determina la selección de medidas de calidad como cobertura, significancia, inusualidad, precisión, soporte nítido, soporte difuso, confianza y confianza difusa. El número de objetivos dentro de la función de agregación ponderada es entre uno y tres.

3.1.1. Funcionamiento del algoritmo genético de SDIGA

El funcionamiento del algoritmo es el siguiente:

INICIO
Conjunto de reglas (vacío)
REPETIR
  REPETIR
    Generar $P_{gen=0}$
    Evaluar P(0)
    REPETIR
      Incluir el mejor individuo en P_{gen+1}
      Completar P_{gen+1}: Cruce y Mutación de individuos in P_{gen}
      Evaluar P_{gen+1}
    HASTA Número de evaluaciones máximo sea alcanzado
    Obtener la mejor regla R
    Búsqueda Local (R)
    SI Confianza(R) >= Confianza Mínima **&** R representa nuevos ejemplos **ENTONCES**
      Conjunto de reglas ← R
      Marcar el conjunto de ejemplos cubiertos por R
    FIN_SI
  HASTA Confianza(R) <= Confianza Mínima **&** R no representa nuevos ejemplos
HASTA Valor objetivo no alcanzado
FIN

Figura 11. Esquema de funcionamiento de SDIGA

Como se indica en el esquema, este es un algoritmo iterativo en el que se ejecuta hasta que se cumple cierta condición de parada. Esta condición de parada se cumplirá si se da uno de los dos siguientes casos, que la regla generada por el algoritmo genético no alcance un nivel de confianza mínimo o que no se haya podido cubrir nuevos ejemplos no cubiertos por reglas anteriores (ejemplos nuevos).

Para cada iteración, el primer paso es generar una población inicial, que evaluamos y tomamos su mejor individuo. A continuación iremos generando nuevas poblaciones usando operadores genéticos. Tomando el mejor individuo anteriormente evaluado que será el primer individuo de la siguiente generación P_{gen+1}. A continuación realizaremos los operadores de cruce y mutación sobre la generación anterior, e incluiremos el resultado en la nueva generación P_{gen+1}. Por último, evaluamos esta nueva generación y asignamos un nuevo mejor individuo para la próxima iteración. Este proceso de generar poblaciones continúa hasta

alcanzar un número máximo de evaluaciones ya predefinido. El último mejor individuo será la mejor regla de esta iteración.

Una vez obtenida la mejor regla, para que esta regla sea más general y sin perjuicio para su confianza, se realiza una optimización local basada en ascensión de colinas. Si la regla alcanza una confianza mínima y representa nuevos ejemplos será incluida en el conjunto final de reglas. Esta optimización local es opcional y el usuario puede optar por no realizarla.

3.1.2. Esquema de representación

El esquema de representación usa reglas canónicas (sección 1.5.3.1) o reglas DNF (sección 1.5.3.2) en función de las necesidades del experto.

3.1.3. Operadores genéticos

Un operador genético es una función utilizada en los algoritmos genéticos para mantener la diversidad genética de una población. Para el paso de una generación a la siguiente se aplican una serie de operadores genéticos. En el caso de no trabajar con una población intermedia temporal también cobrarían relevancia los algoritmos de reemplazo.

La variación genética es necesaria para el proceso de evolución para una mayor diversidad. Los operadores genéticos utilizados en los algoritmos genéticos son análogos a aquellos que ocurren en el mundo natural: el operador de selección sería equivalente a la supervivencia del más apto en el mundo natural; la recombinación, o cruce, que equivale a la reproducción sexual y la mutación equivale a la mutación biológica.

Los Algoritmos Genéticos utilizan un modelo de reproducción de tipo estacionario modificado [27], con el objetivo de aumentar la diversidad de la población. En este modelo, la población original se modifica mediante la sustitución de los peores individuos por individuos resultantes del cruce y la mutación.

3.1.3.1. Operador de selección

En SDIGA, la función de selección crea una población intermedia con los individuos ordenados del mejor al peor.

3.1.3.2. Operador de cruce

El operador de cruce que se utiliza es un operador de cruce en dos puntos, en el cual se toman los dos mejores individuos de la población para ser cruzados, obteniendo dos nuevos individuos.

El proceso del cruce en 2 puntos funciona cortando los cromosomas de los padres en dos puntos, creando tres segmentos.

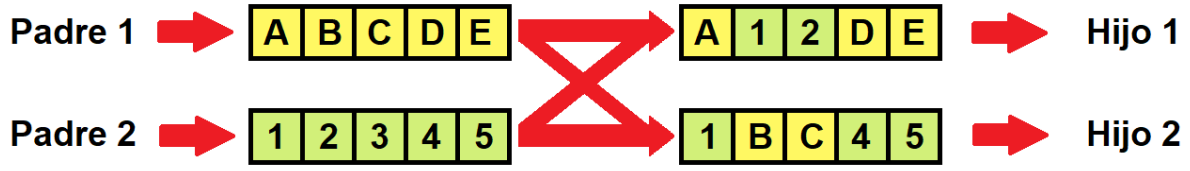


Figura 12: Representación del Cruce en 2 Puntos

Tal como se ve en la figura de arriba, para generar la descendencia, se escoge el segmento central de uno de los padres y los segmentos laterales del otro padre, resultando en dos hijos. Estos dos hijos sustituirán a los dos peores individuos de la población (recordemos que en el operador de selección la población ha sido ordenada de mejor a peor). Esta estrategia conlleva una alta presión selectiva y provoca una rápida convergencia a una solución que probablemente sea un óptimo local.

3.1.3.3. Operador de mutación

La mutación se lleva a cabo mediante un operador de mutación aleatorio sesgado aplicado al gen seleccionado. La mutación se aplica dada una la probabilidad de mutación no solo sobre los dos mejores individuos de la población sino sobre toda la población, tomando cromosomas aleatoriamente. Y una vez tomados estos dos cromosomas, escogiendo una variable aleatoria sobre la que se aplicará el operador.

Este operador se puede aplicar de dos formas diferentes, escogidas aleatoriamente y con la misma probabilidad, que son:

- **Eliminación**, la mutación provoca la eliminación de la variable a la que corresponde el gen. A nivel de código, esto se consigue asignándole el valor de no participación para reglas canónicas o todos sus valores a cero para reglas DNF.
- **Mutación**, se asigna un valor aleatorio a la variable para reglas canónicas o para cada uno de los valores de dicha variable para reglas DNF. El cambio de esta variable puede suponer un cambio en si la variable intervendrá o dejará de intervenir en la regla.

El uso de este operador permite promover la diversidad en la población.

Para obtener una mayor diversidad al aplicar este operador, se recomienda seleccionar un mayor tamaño de población al habitual en los modelos evolutivos con esquema estacionario.

3.1.3.4. Operador de reemplazo

El reemplazo se lleva a cabo, tomando los *n* peores individuos de la población inicial y reemplazándolos con los *n* mejores individuos de la población de descendientes. Siendo *n* igual al número de mutaciones realizadas por el operador correspondiente más dos.

3.1.4. Función de evaluación

El objetivo del proceso de descubrimiento de subgrupos es obtener reglas con alta confianza, comprensibles y generales. Así pues, la función de evaluación es una herramienta para medir como se ajusta cada regla a los objetivos propuestos. SDIGA es un algoritmo mono-objetivo, pero busca maximizar más de una medida de calidad. Para el caso de SDIGA, serían las medidas de Soporte, Confianza más un tercer objetivo adicional.

Para lograr esto, el método de suma ponderada para combinar un conjunto de objetivos en un solo objetivo es el enfoque más simple. Éste método también permite al experto ser quién, con respecto a la importancia de los objetivos para un problema específico en el proceso de generación de reglas, pondere el cálculo de los pesos para cada objetivo usando el criterio experto.

Por tanto, la función de evaluación será de la siguiente manera:

$$f(x)=\frac{Sop(x)*w_1+Conf(x)*w_2+Obj3(x)*w_3}{\sum_{i=1}^{3} w_i}$$

Ecuación 11: Función de evaluación SDIGA

Donde:

- **Obj3**: Es el tercer objetivo, aparte del Soporte y Confianza, a escoger libremente de entre los objetivos ya definidos en la Sección 2.3**.**
- **W_i**: es el peso del objetivo i.

- **Soporte Local (Sop)**: Mide el grado de cobertura que ofrece la regla a los ejemplos que pertenecen a la clase especificada en la regla consecuente. Se calcula de una manera diferente al Soporte (Ecuación 3) ya definido en la Sección 2.3 para promover que se obtengan diferentes reglas difusas en diferentes ejecuciones del algoritmo genético híbrido.

  Para ello, en el cómputo del soporte solo se consideran los ejemplos no marcados (es decir, los ejemplos no cubiertos por otras reglas difusas obtenidas previamente mediante las ejecuciones pasadas del algoritmo genético híbrido). Puede ser nítido o difuso según el usuario desee y se calcula de la siguiente manera:

  - **Soporte Local Nítido**: $Sop_n(x) = \frac{Ne^{'+'}(R_i)}{Ne_{NC}}$

    Ecuación 12: Soporte Local Nítido

  Así, el soporte local nítido se define como el cociente entre los ejemplos de este conjunto parcial cubiertos por la regla representada en el cromosoma (ejemplos correctamente cubiertos), no cubiertos por ninguna regla anterior y el número total de ejemplos de este conjunto parcial (ejemplos de la clase objetivo que quedan por cubrir). Y donde:

  - $Ne^{+}(R_i)$ es el número de nuevos ejemplos (que no hayan sido cubiertos por ninguna regla anterior) que cubren tanto el antecedente como el consecuente (ejemplos correctamente cubiertos) por la Regla *i* (R_i).
  - Ne_{NC} se refiere al número de ejemplos No Cubiertos (NC) todavía que son de la clase objetivo.

  - **Soporte Local Difuso**: $Sop_d(x) = \frac{\sum_k APC(E^k, R_i)}{Ne_{NC}}$

    Ecuación 13: Soporte Local Difuso

  El soporte local difuso se define como el cociente de la suma de los grados de pertenencia de los ejemplos no cubiertos por reglas anteriores y el número total de ejemplos que quedan por cubrir. Y donde:

  - Σ_k APC (E^k, R_i), donde APC = *Antecedent Part Compatibility* es el grado de compatibilidad entre un ejemplo y la parte antecedente de una regla difusa. Y lo calculamos como la suma de los grados de compatibilidad de los ejemplos que no han sido cubiertos por reglas anteriores.

- **Confianza (Conf)**: Determina la precisión de la regla, ya que refleja el grado en que los ejemplos dentro de la zona del espacio determinado por el antecedente verifican la información especificada en el consecuente de la regla como en [25].

  Esta medida también puede calcularse como nítido o difuso:

  - **Confianza nítida**: La confianza nítida se calcula tal y como ya ha sido definida en la Sección 2.3 (Ejemplo 3).

  - **Confianza difusa**:

$$Conf_d(x) = \frac{\sum_{E^{CC}} APC(E^k, R_i)}{\sum_{E^C} APC(E^k, R_i)}$$

Ecuación 14: Confianza difusa para SDIGA

  La confianza difusa se calcula como el ratio de la suma de los grados de compatibilidad de los ejemplos que cubren tanto antecedente como consecuente (ejemplos correctamente cubiertos) entre la suma de los grados de compatibilidad de los ejemplos que cubren el antecedente de la regla, es decir:

  - Σ_{ECC} APC (E^k, R_i) es la suma de los grados de compatibilidad de los ejemplos que cubren tanto antecedente como consecuente (ejemplos correctamente cubiertos) respectivamente.
  - Σ_{EC} APC (E^k, R_i) es la suma de los grados de compatibilidad de los ejemplos que cubren solamente el antecedente de la regla (ejemplos cubiertos).

3.1.5. Optimización Local

La función de Optimización Local es una función opcional (a decisión del usuario) que puede entrar en juego cuando el algoritmo ya ha obtenido la mejor regla de la iteración.

La fase de optimización local, que mejora la regla obtenida mediante un proceso de ascensión de colinas del primer mejor, modifica la regla para aumentar el grado de soporte. Para lograr esto, en cada iteración se selecciona una variable de manera que al eliminarla, se generaliza la regla y se aumenta el soporte de la regla resultante, mientras la confianza se mantiene igual o mayor. Finalmente, si y solo si supera la confianza mínima, la regla optimizada sustituirá a la original.

El diagrama de funcionamiento de la fase de optimización local es mostrado en la siguiente figura:

```
INICIO
    Mejor_regla ← R
    Mejor_soporte ← Soporte(R)
    Mejor ← Verdad
    REPETIR MIENTRAS Mejor
        Mejor ← Falso
        PARA (m=1 a n_v)
                R'_m ← Mejor_Regla sin considerar variable m
                SI (Soporte (R'_m) >= Soporte (R) AND Confianza (R'_m) >= Confianza (R))
                        Mejor ← Verdad
                        SI (Soporte (R'_m) > Mejor_soporte)
                                Mejor_soporte ← Soporte (R'_m)
                                Mejor_regla ← R'_m
                        FIN_SI
                FIN_SI
        FIN_PARA
    FIN_MIENTRAS
    SI (Confianza (Mejor_regla) >= Confianza_minima)
        DEVOLVER Mejor_regla
    SI_NO
        DEVOLVER R
FIN
```

Figura 13: Diagrama de funcionamiento de Optimización Local

3.2. Algoritmo MESDIF (Multiobjetive Evolutionary Subgroup Discovery Fuzzy rules)

Se trata de un Sistema Difuso Evolutivo Multi-objetivo, para ello se unifican todos los objetivos individuales como se indica en el punto "3.2.4. Función de evaluación", para intentar maximizarlos (o minimizarlos) sin que la mejora de un objetivo sea en perjuicio de otro distinto. El algoritmo MESDIF utiliza el enfoque SPEA2 [31], el cual aplica los conceptos de elitismo en la selección de reglas (utilizando una población secundaria o élite) y la búsqueda de soluciones óptimas en el frente de Pareto usando el concepto de dominancia, para rellenar la población de élite.

El concepto de dominancia se base en que un individuo A domina a otro B, si y solo si todos los valores de objetivo A_i son iguales o mejores que los valores de objetivo B_i. En el caso del algoritmo MESDIF, todos los valores de objetivo deberán serán mejores o iguales y, al menos uno de ellos, estrictamente mayor. Este sistema permite encontrar individuos no dominados, teniendo estos individuos prioridad para ser incluidos la población élite. Además, el algoritmo incorpora una Función de Truncado para el caso en que tengamos más individuos no dominados que el tamaño de la población élite. Así como una Función de Rellenado en caso de que tengamos el caso contrario, un número de individuos no dominados inferior al tamaño de la población de élite.

Para preservar la diversidad a nivel fenotípico el algoritmo utiliza una técnica de nichos que considera la proximidad en valores de los objetivos y un objetivo adicional basado en la novedad para promover reglas que den información sobre ejemplos no descritos por otras reglas de la población.

El proceso de inducción de reglas obtiene reglas con alta precisión predictiva y que son comprensibles e interesantes. En esta propuesta, el usuario puede elegir entre un amplio número de medidas de calidad (cobertura, significancia, atipicidad, precisión, soporte y confianza) para maximizar todos los objetivos definidos.

Uno de los aspectos más importantes de MESDIF es la obtención de resultados para todos los valores de la variable objetivo. Devuelve un número de reglas igual al tamaño de la población élite, cuyo tamaño es definido por el usuario, para cada valor de la variable objetivo. Ya que los individuos de ésta población de élite son los que, al final del proceso, serán las reglas que serán devueltas.

3.2.1. Funcionamiento del algoritmo genético MESDIF

El esquema general del algoritmo es el siguiente:

INICIO
Crear una población inicial $P_{gen=0}$
Crear una población de élite vacía $E_{gen=0} = \{ \}$
Generar una población inicial $P_{gen=0}$
REPETIR
Calcular fitness para P_{gen}
Unir los individuos no-dominados de P_{gen} y E_{gen} en E_{gen+1}.
Completar o Truncar la población de élite E_{gen+1} para alcanzar el tamaño tam_elite
Realizar selección por torneo binario con reemplazo sobre E_{gen+1} y aplicar
los operadores de cruce, mutación y reproducción, obteniendo E_{gen+1}.
HASTA que Número de evaluaciones alcanzado
Conjunto de Reglas ← Población E_{gen+1}
FIN

Figura 14. Esquema de funcionamiento de MESDIF

Como se ve en la Figura 14, el primer paso es generar una población principal inicial y una población élite vacía.

A continuación se evalúa la población principal obteniendo el fitness de cada individuo. Después se copian los individuos no dominados de las poblaciones principal y élite (vacía en la primera generación) en la siguiente generación de la población de élite que completaremos o truncaremos según sea necesario hasta alcanzar el tamaño prefijado de élite. Por último se aplican los operadores genéticos a esta nueva generación de élite.

Hasta que el número de evaluaciones alcance el valor máximo prefijado, se repetirán los proceso posteriores a la generación de las poblaciones. Para terminar se podrán incluir en el conjunto de reglas los individuos de la población de élite (individuos no-dominados).

3.2.2. Esquema de representación

El esquema de representación es el mismo que el utilizado en SDIGA, con reglas canónicas (sección 1.5.3.1) o reglas DNF (sección 1.5.3.2) en función de las necesidades del experto.

3.2.3. Generación de la población inicial

MESDIF utiliza una generación de población aleatoria sesgada. El primer paso es generar individuos para un porcentaje prefijado de la población, por defecto un 75%. Para estos individuos se generan cromosomas con un número máximo de variables participando en la regla. El número de variables dependerá de un porcentaje prefijado anteriormente, por defecto un 33%. Así hasta alcanzar el porcentaje de población anteriormente mencionado.

A continuación, el resto de la población se generará de forma completamente aleatoria, tal como en SDIGA. Con esta estrategia, se consigue una mayor generalidad en las reglas generadas. Esto significa que habrá un menor número de reglas con gran cantidad de variables participando en la regla, y por tanto tan específicas que apenas cubren una instancia o ninguna.

3.2.4. Función de evaluación

Como se ha mencionado antes, MESDIF es un algoritmo multi-objetivo, por ello se precisa un método para unificar todos los objetivos en una sola función. Ésa es la función de evaluación, en la que trata de obtener el valor fitness de los individuos, el cual se calcula a partir de los siguientes pasos:

1. Para cada individuo de la población se calcula el valor para todos los objetivos.

2. Los valores alcanzados por cada individuo tanto en la población principal como en la población de élite se utilizarán para calcular qué individuo domina a otro.

3. La fuerza de cada individuo se calcula como el número de individuos que domina cada uno.

4. El fitness inicial A_i de cada individuo i se determina como la suma de la fuerza de sus dominadores (tanto de la población principal como de la población élite). Por lo que tendrá un valor ideal de cero (individuos no-dominados), y para individuos dominados se habrá de minimizar.

5. El cálculo del fitness inicial (F_0) ofrece un mecanismo de nicho basado en el concepto de dominancia de Pareto, pero puede fallar cuando hay muchos de los individuos son no-dominados (ya que todos los individuos no-dominados alcanzan el valor máximo de la función). Para evitar esto, se incluye el

concepto adicional de la densidad para discriminar entre individuos con el mismo valor fitness inicial. La técnica de estimación de densidad utilizada en SPEA2 es una adaptación del método del k-ésimo vecino más cercano, donde la densidad en un punto es función decreciente de la distancia al k-ésimo punto más cercano. En esta propuesta usamos el inverso de la distancia al k-ésimo vecino más cercano como estimación de densidad.

$$D(i)=\frac{1}{\sigma^{k}+2}$$

Ecuación 15: Densidad

Donde σ^k es el valor del k-ésimo vecino más cercano.

6. El valor del fitness final de cada individuo es la suma de su valor de fitness inicial (F_0) y de su valor de densidad. Y obviamente, también es una función que habrá de minimizar.

$$Fitness(i)=D(i)+F_0(i)$$

Ecuación 16: Fitness

3.2.4.1. Definición de los objetivos del algoritmo MESDIF

En el proceso de inducción de reglas, MESDIF trata de obtener reglas con alta precisión predictiva, comprensibles e interesantes. Para ello se definen tres objetivos, y el algoritmo intenta maximizar todos los objetivos definidos.

- **Confianza** [10]: Determina la frecuencia relativa de ejemplos que satisfacen la regla completa entre aquellos que satisfacen solo el antecedente. En este proyecto usamos una adaptación de la expresión de precisión de Quinlan [18] para generar reglas de clasificación difusa [23]: la suma del grado de pertenencia de los ejemplos de esta clase (los ejemplos cubiertos por esta regla) a la zona determinada por el antecedente, dividido por la suma del grado de pertenencia de todos los ejemplos que verifican la parte antecedente de esta regla (independientemente de su clase) a la misma zona:

$$Conf(R^i)=\frac{\sum_{E^S\in E / E^S\in Class_j} APC(E^S,R^i)}{\sum_{E^S\in E} APC(E^S,R^i)}$$

Ecuación 17: Confianza para MESDIF

donde APC (Antecedent Part Compatibility) es el grado de compatibilidad entre un ejemplo y la parte antecedente de una regla difusa, es decir, el grado de pertenencia del ejemplo al subespacio difuso delimitado por la parte antecedente de la regla.

- **Soporte** [10]: Es la medida del grado de cobertura que la regla ofrece a los ejemplos de esa clase, calculado como el cociente entre el número de ejemplos pertenecientes a la clase que están cubiertos por la regla y el número total de ejemplos de la misma clase:

$$Sop(R^i) = \frac{n(Class_j . Cond^i)}{n(Class_j)}$$

Ecuación 18: Soporte para MESDIF

- **Soporte original**: Este objetivo es una medida del nivel de originalidad de la regla en comparación con el resto de reglas. Se calcula sumando, para cada ejemplo perteneciente al antecedente de la regla, el factor 1/k, donde k es el número de reglas de la población que describen información sobre ese ejemplo. Esta medida promueve la diversidad en la población a nivel fenotípico.

El último objetivo definido, el soporte original, es una restricción en las reglas para obtener un conjunto de reglas, el frente del Pareto, con un alto grado de cobertura, y está relacionado con la cooperación entre reglas; los demás objetivos tienen en cuenta el soporte y la confianza.

3.2.5 Función de truncado

Este algoritmo establece una longitud fija para la población élite, por tanto es posible que haya un número de individuos no-dominados mayor que el tamaño de la población élite. Por tanto es necesario una función que realice una criba entre estos individuos no-dominados, ésta sería la función de truncado.

La función de truncado permite eliminar las soluciones no-dominadas de la población élite si ésta supera el tamaño definido para dicha población. Para ello, es necesario conocer el valor fitness de cada individuo, así como saber qué individuos son dominados y no-dominados. También suponemos que hemos insertado todos los individuos no-dominados en la población élite. Una vez realizados los pasos preparatorios, podemos proceder con la función de truncado.

El primer paso será calcular la distancia entre todos los individuos no-dominados. Y hasta que el número de individuos quepa en la población élite, en cada iteración se elimina de la población de élite al individuo más cercano a los demás respecto de los valores de los objetivos.

Este método permite que las soluciones que añadamos a la población élite proporcionen una mayor diversidad.

3.2.6. Función de rellenado

Del mismo modo que en la función de truncado, es posible que haya menos individuos no-dominados que el tamaño de la población élite.

En este caso, dado que no hay suficientes individuos no-dominados, es necesario rellenar la población élite añadiendo individuos dominados. La función de relleno permite añadir individuos dominados de la población hasta alcanzar el tamaño exacto del conjunto, ordenando los individuos según sus valores de fitness (recordemos que el valor fitness cuanto más cerca de cero, mejor) e incluir en la población de élite aquellos con el fitness más bajo.

3.2.7. Operadores genéticos

Los operadores genéticos usados por el algoritmo MESDIF son los siguientes:

3.2.7.1. Operador de selección

El operador de selección usado por MESDIF consiste en una selección por torneo binario con reemplazo. La idea principal de este método consiste en realizar la selección en base a comparaciones directas entre individuos. Se selecciona al azar un número p de individuos (puesto que es torneo binario p=2). De entre los individuos seleccionados se selecciona el más apto para pasarlo a la siguiente generación.

Esta selección se realiza únicamente sobre los individuos de la población élite y se seleccionan individuos hasta alcanzar un número igual al tamaño de la población inicial.

Generalmente el tamaño de la población élite es mucho menor que el de la población inicial (desde cientos hasta miles de veces menor). Esto significa que tendremos muchos individuos repetidos, por lo que esta estrategia tiene una fuerte

presión selectiva, por tanto los peores individuos apenas tendrán oportunidades de reproducción.

3.2.7.2. Operador de cruce

El operador de cruce usado en el algoritmo MESDIF es el cruce en dos puntos, en otras palabras el mismo explicado en la sección 3.1.3.2.

3.2.7.2. Operador de mutación

El operador de mutación es un operador de mutación uniforme sesgado en el que la mitad de las mutaciones realizadas tienen como efecto eliminar la variable correspondiente, con el fin de aumentar la generalidad de las reglas. Este operador es el mismo que para SDIGA y se explica detalladamente en la sección 3.1.3.3.

3.2.7.4. Operador de reemplazo

El operador de reemplazo selecciona a los k peores individuos de la población y los reemplaza por los k primeros individuos de la población de descendientes. Siendo k el número de individuos resultantes de los operadores de cruce y mutación (descendientes).

Este operador, no comprueba si el valor fitness de los nuevos individuos es mejor o peor, siempre hace la sustitución en cualquier caso.

3.3. Algoritmo NMEEF-SD

El algoritmo NMEEF-SD es un Sistema Difuso Evolutivo con un enfoque multiobjetivo cuya estrategia de búsqueda se basa en el algoritmo NSGA-II [32], que se basa en un enfoque de ordenación rápida por dominancia y en el uso de elitismo.

Siendo el objetivo general de NMEEF-SD el obtener un conjunto de reglas, que sean los más generales y precisas posibles, el algoritmo incluye componentes que mejoran estas características. Concretamente, se mejora la diversidad en la población utilizando un nuevo operador, que si la población no evoluciona durante una cierto número de evaluaciones, realiza una re-inicialización basada en la cobertura. Además de la utilización de la técnica Crowding Distance en el operador de selección para promover la diversidad de los individuos seleccionados en la población de cada generación.

Finalmente, para asegurar la precisión, NMEEF-SD sólo devuelve como solución final aquellas reglas que alcanzan un umbral de confianza ya predeterminado.

El algoritmo NMEEF-SD permite establecer entre dos y tres medidas de calidad como objetivos del proceso evolutivo para obtener subgrupos relevantes, entre: cobertura, significancia, inusualidad, precisión, soporte y confianza.

3.3.1. Funcionamiento del algoritmo genético NMEEF-SD

El esquema general del algoritmo es el siguiente:

INICIO
REPETIR
Generar población inicial $P_{gen=0}$
REPETIR
Generar descendencia para Q_{gen} con los operadores genéticos a partir de P_{gen}
$R_{gen} \leftarrow (P_{gen}$ UNION $Q_{gen})$
Generar todos los frentes no dominados de R_{gen}
SI el frente del Pareto evoluciona ENTONCES
Completar P_{gen+1} con los frentes por orden de no-dominancia
SI_NO
Aplicar Re-Inicialización basado en cobertura sobre P_{gen+1}
FIN_SI
HASTA Número de evaluaciones máximo sea alcanzado
Conjunto de reglas ← frente del Pareto P_{gen}
HASTA Valor objetivo no alcanzado
FIN

Figura 15: Esquema de funcionamiento de NMEEF-SD

Como se indica en el esquema de la figura 15, el funcionamiento del algoritmo es el siguiente:

Comenzamos generando una población inicial $P_{gen=0}$ generada con inicialización sesgada al igual que en MESDIF. A continuación los siguientes pasos se reiterarán has alcanzar el número máximo de evaluaciones:

1. Realizamos sobre P_{gen} una selección por torneo binario, los individuos se seleccionan en función del frente (rango) al que pertenezcan y de su “crowding distance” y se les aplican los operadores genéticos de cruce y mutación. Los individuos resultantes de los operadores genéticos generarán la población descendencia Q_{gen}.

2. Se genera una población temporal R_{gen} formada por la unión de la población principal P_{gen} y la población descendencia Q_{gen}.

3. Esta población R_{gen} se ordenará según la siguiente forma: primero se colocan los individuos no dominados, seguidos de los individuos que están dominados por solo un individuo, después por los que están dominados por dos, y así hasta el final.

4. Comprobamos si el frente de Pareto evoluciona. Definimos evolucionar como el caso en que el frente de Pareto actual cubre al menos un nuevo ejemplo que no estaba cubierto por el frente de la iteración anterior.
5. Se genera la siguiente generación de la población. Esta población P_{gen+1} se puede obtener de dos formas:
   - Si el frente del Pareto evoluciona, completamos P_{gen+1} con los k primeros frentes completos hasta que la población se rellene. Si se da el caso de que el tamaño de un frente es mayor que el tamaño definido para la población principal (o sea, no caben todos los individuos), serán los individuos con mayor "crowding distance" los que serán copiados a P_{gen+1}.
   - Si el frente del Pareto no evoluciona tras un 5 % de las evaluaciones máximas, una nueva población P_{gen+1} será generada utilizando un operador de reinicialización de la población basada en cobertura, que se explicará mas adelante.

Una vez alcanzado el número máximo de evaluaciones, y si supera el umbral de confianza mínima, se devuelve el primer frente de Pareto como solución que será incluida en el conjunto de reglas para la variable objetivo.

3.3.2. Esquema de representación

Con respecto a la representación de las reglas, aunque si bien NMEEF-SD puede utilizar tanto las reglas canónicas (sección 1.5.3.1) como DNF (sección 1.5.3.2), de hecho actualmente la utilización de reglas DNF para NMEEF-SD no obtiene resultados favorables y por lo tanto se descarta la utilización de este tipo de representación.

3.3.3. Generación de la población inicial

El algoritmo NMEEF-SD incluye un método de inicialización sesgada de población exactamente igual al utilizado en el algoritmo MESDIF, ya explicado en la sección 3.2.3.

3.3.4. Operadores genéticos

Los operadores genéticos usados por el algoritmo NMEEF-SD son los siguientes:

3.3.4.1. Operador de selección

El operador de selección usado por NMEEF-SD consiste en una selección por torneo binario. La medida utilizada para decidir el ganador del torneo es rango, seleccionándose el individuo con menor rango como vencedor. Y en caso de que los dos oponentes tengan el mismo rango, se selecciona aquel con el mayor valor de crowding distance, el cual se explicará más adelante.

3.3.4.2. Operador de cruce

El operador de cruce usado en el algoritmo NMEEF-SD es el cruce en dos puntos, en otras palabras el mismo explicado en la sección 3.1.3.2.

3.3.4.3. Operador de mutación

El operador de mutación utilizado por el algoritmo NMEEF-SD es un operador de mutación uniforme sesgado. Este operador es el mismo que es utilizado para SDIGA y MESDIF. Se explica detalladamente en la sección 3.1.3.3.

3.3.5. Crowding Distance

NMEEF-SD utiliza esta medida de densidad en las siguientes ocasiones:

Primero, al realizar la selección de la población Q_{gen}. Como se ha mencionado en la sección 3.3.1, cuando se da el caso en que los dos oponentes del torneo binario tengan el mismo rango, para desempatar, se selecciona aquel con la mayor crowding distance. Ésta medida promueve la diversidad entre los individuos incluidos en la población principal para la próxima generación.

Segundo, después para rellenar la población P_{gen+1} para añadir individuos cuando se da el caso de que no caben todos los de un frente dado en la población.

La crowding distance se define como la distancia media de dos puntos, a cada lado de un punto i, según cada uno de los objetivos. La cantidad sirve como un estimado del perímetro del cuboide formado usando los vecinos más cercanos como vértices.

En otras palabras esta medida se basa en la obtención del mayor cuboide que encierra a un punto i, usando los vecinos más cercanos como vértices, sin que haya más puntos dentro. La crowding distance de la solución i-ésima con los puntos que están en el mismo frente (representados en la figura 16 como puntos negros) se calcula como la longitud de lado media del cuboide (mostrado como una caja discontinua) usando f_1 y f_2 como objetivos sobre los cuales calcular la distancia.

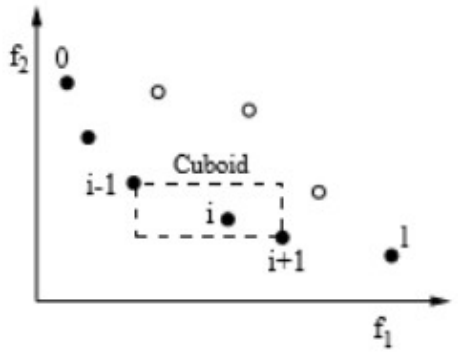


Figura 16. Representación del cuboide

La computación de la crowding distance, primero, requiere la ordenación de la población de acuerdo con cada valor de la función objetivo en orden ascendente de magnitud. Segundo, para cada función objetivo las soluciones límite (soluciones con los valores de función más grandes y más pequeños) se le asigna un valor de distancia infinito.

A todas las otras soluciones intermedias se les asigna un valor de distancia igual a la diferencia normalizada absoluta en los valores de función de dos soluciones adyacentes.

$$D_{[i]} = d_{[i]} / (O_{max} - O_{min})$$

Ecuación 19: Distancia para NMEEF-SD

Donde:

- $d_{[i]}$ es la distancia actual del individuo i.
- O_{min} se refiere al valor de cada objetivo para el primer individuo de la población ordenada.
- O_{max} se refiere al valor de cada objetivo para el último individuo de la población ordenada.

El valor crowding distance total es calculado como la suma de los valores de distancia individuales correspondientes a cada objetivo. Cada función objetivo es normalizada antes de calcular su crowding distance.

3.3.6. Operador de reinicialización

Como se ha explicado anteriormente, para generar la siguiente generación de la población para la siguiente iteración, se comprueba si el frente del Pareto evoluciona, esto es si el frente de Pareto actual cubre al menos un nuevo ejemplo que no estuviera cubierto por el frente de la iteración anterior.

Si pasadas un número de iteraciones igual al 5% del número de evaluaciones máximas y el frente aún no ha evolucionado, la siguiente población P_{gen+1} será generada aplicando un operador de reinicialización de la población basada en cobertura.

Este operador de reinicialización primero elimina los individuos repetidos del frente del Pareto. A continuación, genera nuevos individuos por el método de generación aleatoria de individuos por cobertura. Estos nuevos individuos sustituirán las posiciones de los individuos repetidos de la población, ésta será la siguiente generación P_{gen+1}.

Este método de generación aleatoria de individuos por cobertura funciona del siguiente modo: Lo primero, establece el número máximo de variables que intervendrán en el cromosoma, podemos predefinir este valor, por defecto se toma el

50% de las variables. El segundo paso será ir tomando ejemplos aleatoriamente hasta dar con uno que cubra la clase objetivo (que tenga la clase objetivo en el consecuente), si no se encuentra ninguno se toma un ejemplo aleatorio.

A continuación, para cada variable que interviene, se toma un índice de variable aleatorio que no esté ya inicializado en el cromosoma. Y se calcula el grado de pertenencia del ejemplo tomado para la variable tomada. Para ir terminando, se asigna al cromosoma la etiqueta con el mayor grado de pertenencia.

Por último, el resto de las variables se inicializan con el valor de no intervención en la regla.

3.4. Algoritmo FuGePSD

El algoritmo basado en la Programación Genética Difusa para el Descubrimiento de Subgrupos o también "Fuzzy Genetic Programming-based algorithm for Subgroup Discovery" (FuGePSD) es un Sistema Difuso Evolutivo que como su propio nombre dice, está basado en un algoritmo de programación genética capaz de extraer reglas difusas descriptivas para la tarea de desarrollo de subgrupos [56].

En primer lugar, es importante tener en cuenta la representación y el enfoque utilizados en el algoritmo. FuGePSD representa a los individuos a través de un enfoque “cromosoma = regla” que incluye tanto el antecedente como el consecuente de la regla. De esta forma, este algoritmo obtiene las reglas para todos los diferentes valores de la variable objetivo en tan solo una ejecución del algoritmo, en contraposición al resto de EFSs para Desarrollo de Subgrupos.

El algoritmo FuGePSD emplea el enfoque de cooperación-competición genética donde las reglas de la población cooperan y compiten entre sí para obtener la solución óptima. También es importante remarcar que los individuos son de longitud variable al igual que la población con un número variable de individuos a lo largo del proceso evolutivo. De este modo, FuGePSD es capaz de obtener reglas con distinto número de variables en la parte antecedente de la regla asociadas a la complejidad del subgrupo a describir.

El algoritmo FuGePSD hace uso de una serie de funciones como la función de competición de tokens y una función pantalla. La función de competición de tokens, tiene el fin de mejorar la diversidad del conjunto final de reglas, a la vez que reduce el número de reglas, eliminando reglas que describen información sobre ejemplos ya descritos por otras reglas. Mientras que la función pantalla realiza un cribado sobre la población con el fin de sólo obtener reglas que superen un umbral mínimo de Sensibilidad y Confianza.

Otro punto a mencionar es que el algoritmo FuGePSD siempre **devuelve al menos una regla**. Y es que si al final del proceso evolutivo, el conjunto de reglas final está vacío, FuGePSD devuelve la mejor regla con respecto a la confianza. Por tanto, como hemos dicho, este algoritmo siempre obtiene reglas.

3.4.1. Funcionamiento del algoritmo genético FuGePSD

El esquema general del algoritmo es como se muestra en la siguiente figura:

INICIO

```
    Generar PobPrincipal
    Evaluar PobPrincipal
    MejorPob ← PobPrincipal
    REPETIR
        Generar PobDescendiente a través de operadores genéticos
        Evaluar PobDescendiente
        PobUnida ← (PobPrincipal UNION PobDescendiente)
        PobPrincipal ← CompeticiónToken (PobUnida)
        SI PobPrincipal . Fitness > MejorPob . Fitness ENTONCES
            MejorPob ← PobPrincipal
        FIN_SI
    HASTA Número de evaluaciones máximo sea alcanzado
    Conjunto de reglas ← FunciónPantalla (MejorPob)
FIN
```

Figura 17: Esquema de funcionamiento de FuGePSD

El algoritmo inicia con la generación de una población principal (PobPrincipal) que es evaluada y será adaptada a lo largo de las diferentes generaciones por medio de operadores genéticos, generándose la población descendiente (PobDescendiente) y consecutivamente evaluándola. La población descendiente obtenida tendrá el mismo tamaño que la población principal.

Se aplicarán diferentes operadores genéticos, que serán detallados más adelante, para generar la población descendiente a partir de la población principal, de modo que todos y cada uno de los individuos generados en la población descendiente serán una modificación de un cierto individuo de la población principal.

A continuación, ambas poblaciones (principal y descendiente) se unen en una nueva población conjunta (PopUnida). Debido al uso de un enfoque cooperativo-competitivo, tanto los individuos como la población de la población conjunta se evalúan de manera separada, ya que es necesario evaluar a los individuos y las poblaciones a través de dos funciones de aptitud independientes. Por lo tanto, los individuos compiten entre sí con respecto a un fitness local y cooperan para obtener una población más adaptada al problema. Ambas funciones se denominarán función fitness y fitness global, respectivamente.

Entonces, se aplica la Función Competición de Tokens, para mejorar la diversidad entre los individuos a nivel fenotípico, emulando el comportamiento de un medio natural. Los individuos con buenos nichos intentarán explotarlo exclusivamente y evitarán que más individuos compartan sus recursos, a menos que uno más nuevo sea más fuerte que el desarrollado inicialmente. Por lo tanto, los otros individuos están obligados a buscar sus propios nichos. Estas propiedades proporcionan

diversidad en la población al algoritmo. Además, este operador reduce el número de reglas porque se eliminarán todos los individuos sin tokens.

Este proceso evolutivo se controla a través de un número de generaciones máximo previamente definido. Dada una mejor población (MejorPob) inicial igual a la población principal de la primera generación, al final de cada generación, si la población resultante de cada generación es mejor que la mejor población actual, sustituye a ésta como mejor población.

Finalmente, el algoritmo realiza una Función Pantalla (Screening Function), realizando un cribado sobre la mejor población del proceso evolutivo completo con el fin de obtener reglas solo con valores superiores a un umbral mínimo de Sensibilidad y Confianza. Generalmente, estos umbrales deben configurarse por encima del 60% porque los subgrupos obtenidos deben ser a la vez precisos y generales, y ambas medidas de calidad son ideales para cumplir estos objetivos. Esta función es capaz de controlar, a través de un parámetro externo, la necesidad de obtener reglas para todos los valores de la variable objetivo, obteniendo solo las mejores reglas. Si el conjunto de reglas final está vacío, FuGePSD devuelve la mejor regla con respecto a la confianza. De esta forma, el algoritmo siempre obtiene reglas.

3.4.2. Esquema de representación

El algoritmo FuGePSD utiliza una gramática libre de contexto que permite el aprendizaje de reglas difusas y la ausencia de algunas características de entrada, un proceso que da lugar a reglas compactas y simples. La Tabla 2 representa un ejemplo de gramática para una tarea de descubrimiento de subgrupos con dos valores (X_1; X_2), cinco etiquetas LL por valor ($LL_{1.1}$; $LL_{1.2}$; ... ; $LL_{1.5}$; $LL_{2.1}$; ... ; $LL_{2.5}$) y dos valores para la variable objetivo (V_{obj1} ; V_{obj2}) donde el símbolo '*?a*' en alguna de las reglas de producción de la gramática representa uno, y sólo uno, de los valores separados por comas entre corchetes.

Inicio → [Si], antecedente, [entonces], variable_objetivo, [·]
antecedente → descriptor1, [Y], descriptor2
descriptor1 → [algo]
descriptor1 → [X_1] = etiqueta
descriptor2 → [algo]
descriptor2 → [X_2] = etiqueta
etiqueta → miembro (?a, [LL_1, LL_2, LL_3, LL_4, LL_5]), [?a]
variable_objetivo → [valor_objetivo] = descriptor
descriptor → miembro (?a, V_{obj1}, V_{obj2}), [?a]

Tabla 2: Ejemplo de gramática de FuGePSD

Con respecto al uso de la lógica difusa, la cantidad de etiquetas (LL) pueden ser especificados por el usuario, o en su defecto, definidos mediante una partición uniforme (si no se dispone de conocimiento experto). Aunque FuGePSD puede usar ambas representaciones, en este proyecto se emplean particiones uniformes con funciones triangulares como las mostradas en la Figura 5 (ver sección 1.5.2).

3.4.3. Generación de la población inicial

Las reglas de la gramática antes mencionadas son consideradas a la hora de generar la población inicial del algoritmo FuGePSD. Esta población se genera de forma completamente aleatoria, donde los individuos contienen un número aleatorio de descriptores de hasta el 50% de las variables del problema.

3.4.4. Operadores genéticos

Los operadores genéticos usados por el algoritmo FuGePSD son los siguientes:

3.4.4.1. Operador de selección

El operador de selección usado por FuGePSD consiste en una selección por torneo, pero a diferencia del operador de selección del algoritmo NMEEF-SD, éste no será un torneo binario.

Se selecciona aleatoriamente k individuos de la población principal, siendo k el número de competidores predefinidos para el torneo. La medida utilizada para decidir el ganador del torneo es el fitness, seleccionándose el individuo con mayor fitness como vencedor.

3.4.4.2. Operador de cruce

El operador de cruce utilizado por FuGePSD es el operador de cruce en dos puntos ya explicado en la sección 3.1.3.2. En este caso, uno de los dos progenitores será el individuo ya escogido por el operador de selección, y en cuanto al otro será escogido aleatoriamente de entre el resto de individuos de la población principal.

Como ya se dijo anteriormente, el operador de cruce en dos puntos genera dos descendientes, pero sólo uno de ellos, aleatoriamente escogido, será devuelto como descendiente.

3.4.4.3. Operador de mutación

El operador de mutación está destinado a mejorar la diversidad de la población descendiente. Este operador modifica una variable del individuo seleccionado por el operador de selección. La variable en cuestión es seleccionada aleatoriamente.

La modificación consiste en asignar a la variable un valor distinto del original, teniendo siempre en cuenta las restricciones de la gramática.

3.4.4.4. Operador de inserción

El operador de inserción tiene como finalidad incluir reglas más específicas en el modelo, con el objetivo de mejorar la precisión y confianza del conjunto de reglas del algoritmo.

Este operador inserta una nueva variable en el individuo (seleccionado por el operador de selección) con un valor generado de forma aleatoria.

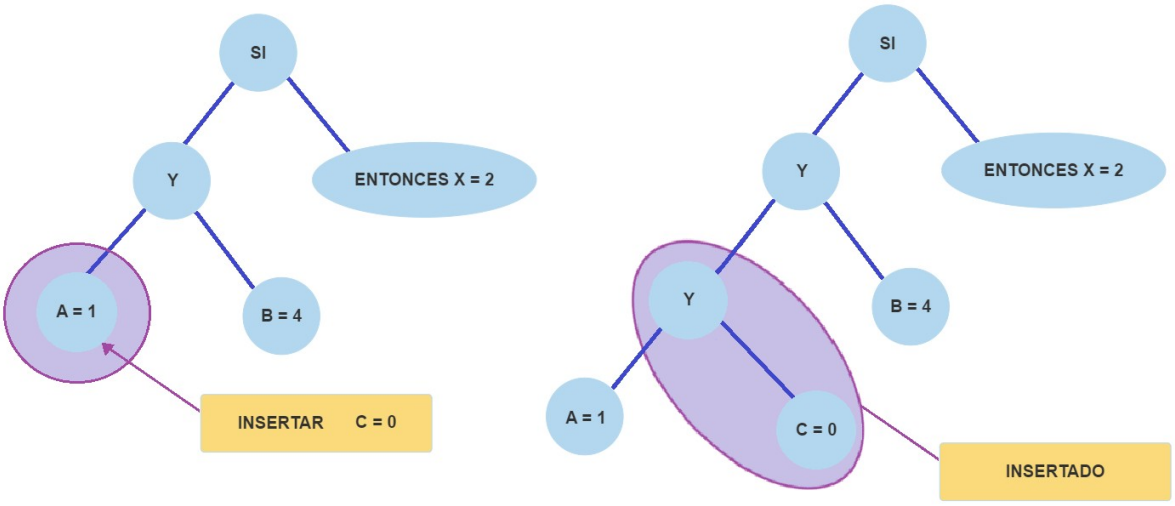


Figura 18: Representación del operador de inserción

El operador de inserción no es aplicado si el individuo ya tiene inicializadas todas las variables según la gramática. La Figura 18 izq. representa el estado preliminar de la regla, mientras que la figura 18 derecha representa la regla después de que el operador de inserción haya insertado la variable (C = 0) en la regla quedando ésta como: SI (A=1 Y C=0) Y B=4 ENTONCES X=2].

3.4.4.5. Operador de eliminación

El operador de eliminación (dropping operator) intenta generar individuos más generalizados que mejoren las medidas de soporte y sensibilidad. Además, dada la naturaleza probabilística de la programación genética, se pueden generar restricciones redundantes en el individuo. Por lo tanto, es necesario generalizar las reglas para representar el conocimiento de una forma más concisa.

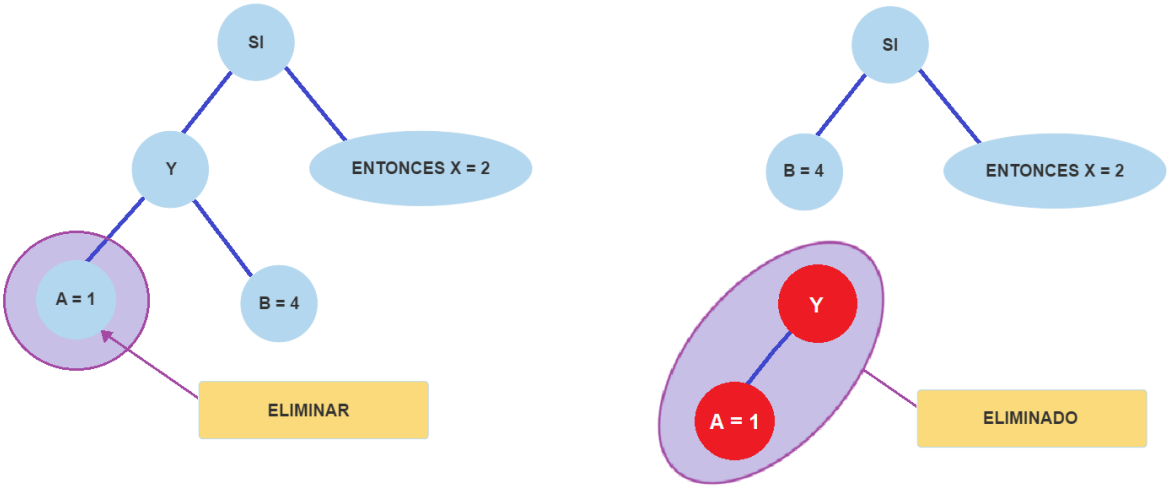


Figura 19: Representación del operador de eliminación

El operador selecciona aleatoriamente una variable en el individuo (seleccionado por el operador de selección) y la elimina. Esta variable dejará de ser considerada dentro de la regla, por lo que las selecciones pueden generalizarse. Este operador no se aplica si el individuo solo tiene una variable en el antecedente, siguiendo la gramática del algoritmo. La Figura 19 muestra el operador de eliminación del algoritmo, donde la variable X se elimina del individuo inicial.

3.4.5. Función de Competición por Tokens

El algoritmo FuGePSD hace uso de la Función de Competición por Tokens con el objetivo de mejorar la diversidad entre los individuos a nivel fenotípico. Este mecanismo nos permite obtener una serie de reglas, todas las cuales cubren al menos un ejemplo del conjunto de datos aún no cubierto por otras reglas más fuertes, y se reduce el número de reglas obtenidas, ya que los individuos que describen información sobre ejemplos descritos por otras reglas son eliminados.

Emulando el comportamiento de un medio natural, los individuos con buenos nichos intentarán explotarlo exclusivamente y evitarán que más individuos compartan sus

recursos, a menos que uno más nuevo sea más fuerte que el desarrollado inicialmente. Por lo tanto, los otros individuos están obligados a buscar sus propios nichos. Estas propiedades proporcionan diversidad en la población al algoritmo. Además, este operador reduce el número de reglas porque se eliminarán todos los individuos sin tokens.

Cada ejemplo de nuestro conjunto de datos es considerado como un token y todos los individuos de la población competirán por este ejemplo. Cuando un individuo coincide con un ejemplo proporcionado, una indicador mostrará que el ejemplo está tomado y, por lo tanto, otros individuos no pueden capturarlo. De esta manera, se mejora la diversidad con respecto al fenotipo, es decir, los individuos que superan la competición por tokens describen información sobre ejemplos del conjunto de datos que no están cubiertos por otras reglas. El objetivo de este operador es incrementar la descripción de la base de reglas sobre el conjunto de datos reduciendo, en la medida de lo posible, la redundancia.

El funcionamiento de este mecanismo comienza con la ordenación (de mayor a menor) de la población con respecto al fitness individual. Posteriormente, un individuo con el fitness más alto explotará sus nichos apoderándose de tantos tokens como pueda. Los otros individuos que ingresen a los mismos nichos verán disminuida su fuerza, ya que no pueden competir con los más fuertes. Esto se consigue introduciendo una penalización a la puntuación de fitness de cada individuo, una restricción basada en el número de fichas que cada individuo haya cogido:

$$FitnessPenalizado(R_i) = Atipicidad(R_i) * \frac{conteo(R_i)}{ideal(R_i)}$$

Ecuación 20: Fitness penalizado

donde conteo(R_i) es el número de tokens de la regla R_i actualmente tomados, e ideal (R_i) es el número total de tokens que puede tomar, en otras palabras es el número de ejemplos con que la regla coincide. Si una regla toma cero tokens, su aptitud se modifica a cero directamente. Al terminar la operación de este mecanismo, el tamaño de la población es reducida a los individuos, donde FitnessPenalizado es mayor que cero.

3.4.6. Funciones fitness

Como hemos mencionado anteriormente, el algoritmo FuGePSD sigue un enfoque de aprendizaje genético de tipo cooperación-competición, es decir, FuGePSD codifica una sola regla por individuo y compiten y cooperan simultáneamente. Esto hace necesario considerar no solo las características de las

reglas individuales sino también la cooperación entre reglas. Para hacerlo, FuGePSD requiere el uso de dos funciones fitness diferentes para optimizar los individuos de la población, y también otra función para optimizar a toda la población a través de una función fitness global que evalúa la precisión del conjunto de reglas como un medio de clasificación. Ambas funciones serán referidas como función fitness y función fitness global, respectivamente.

- **Función fitness**: Se calcula a través de una medida de calidad específica para la tarea de descubrimiento de subgrupos elegida por el experto. Específicamente, el algoritmo puede trabajar con las diferentes medidas de calidad ya presentadas en la sección 2.3: Inusualidad, Sensibilidad y Confianza difusa. El uso de una medida de calidad como objetivo en el proceso evolutivo suele permitir al algoritmo que los individuos con los mejores valores en esta medida de calidad sean elegidos y seleccionados para la evolución del proceso.

  En general, mediante el uso de la medida Inusualidad se logra un buen comportamiento para descubrimiento de subgrupos, ya que mide novedad e interés. Si el experto decide utilizar la medida Sensibilidad en el proceso evolutivo, los subgrupos obtenidos serán más generales con un número bajo de variables, mientras que con el uso de la confianza se puede obtener la obtención de subgrupos muy precisos pero menos compactos.

  La elección de la función de aptitud proporciona una gran mejora de la versatilidad del algoritmo, ya que los expertos pueden adaptar el proceso de extracción de conocimiento con respecto a la naturaleza del problema. La función de aptitud se elige a través de un parámetro externo del algoritmo.

- **Función Fitness Global**: Esta se calcula a través de un puntuación de fitness para obtener la mejor población a lo largo de todo el proceso evolutivo. De esta forma, es necesario calcular la precisión del conjunto de reglas utilizando la suma normalizada de las predicciones para cada regla. El fitness global se define de la siguiente manera:

$$FitnessGlobal = \frac{w_1 * TPM + w_2 * (1 - n_v) + w_3 * (1 - n_i)}{w_1 + w_2 + w_3}$$

Ecuación 21: Fitness Global

  donde n_i es el número medio de individuos para la población, n_v es el número de variables descriptivas y TPM o Tasa de Precisión Media es el valor medio

de la precisión de cada valor individual de la variable objetivo (calculado como):

$$TPM = \frac{1}{n_{Vobj}} * \sum_{i=1}^{n_{Vobj}} TPR_i$$

Se utilizan diferentes pesos (w1; w2 y w3) para lograr una mayor "compensación" entre precisión e interpretabilidad a fin de satisfacer todos los requisitos del descubrimiento de subgrupos (generalidad, precisión y que sea de interés). Estos pesos también se pueden modificar a través de parámetros externos para facilitar la adaptabilidad de los expertos a problemas reales y complejos. De esta forma, es recomendable utilizar valores superiores a 0,7 sobre 1 para w_1, y el 0,3 sobre 1 restante será dividido entre w_2 y w_3, ya que la idea principal es obtener un modelo preciso con un número bajo de reglas y un bajo número de variables para mayor generalidad en el conjunto de reglas.

Para el objetivo de generalidad, el uso de (1 - n_v) y (1 - n_i) da lugar a que se penalice si hay un número excesivo de reglas y variables en la puntuación de la población. Con respecto a la medida de precisión, se utiliza la Tasa de Precisión Media (TPM) ya que un peso equitativo para todas las clases del problema es independiente del número de ejemplos para los cuales cada uno es aplicado. El resultado de esta Tasa de Precisión Media es más adecuado para obtener una precisión homogénea para todos los valores de la variable objetivo.

3.4.7. Función de Pantalla

Para la extracción final de las mejores reglas, se aplica una función de criba. Esta función se puede observar en la figura de abajo. El cribado se realiza para la (mejor) toda la (mejor) población MejorPob donde el algoritmo obtiene las mejores reglas para el conjunto de datos, las cuales deben alcanzar unos valores determinados de Confianza y Sensibilidad.

```
INICIO
PARA (i=1 hasta MejorPob . Tam) HACER
    R ← getRegla (MejorPob, i)
    SI R.getConf ≥ CONFIZANZA Y R.getSens ≥ SENSIBILIDAD ENTONCES
        ConjuntoReglas ← R
    FIN_SI
FIN_PARA
SI AllTargetValues = TRUE ENTONCES
    PARA (i=1 hasta n_val_obj) HACER
        SI NumReglas (ConjuntoReglas, i) = ∅ ENTONCES
```

```
            R ← MejorPob . MejorRegla (ni)
            ConjuntoReglas ← R
        FIN_SI
    FIN_PARA
SI_NO
    SI NumReglas (ConjuntoReglas) = ∅ ENTONCES
        R ← MejorPob . MejorRegla()
        ConjuntoReglas ← R
    FIN_SI
FIN_SI
FIN
```

Figura 20: Esquema de funcionamiento de la función de pantalla

A continuación, la función contiene un parámetro externo (AllTargetValues) para indicar el tipo de reglas extraídas. Hay dos opciones para este parámetro, la primera, con valor VERDADERO indica la necesidad de obtener reglas para cada uno de los valores de la variable objetivo. Con este valor, se asegura que en el conjunto final de reglas se incluya al menos una regla para cada valor de la variable objetivo (la mejor regla para dicho valor objetivo). Por otro lado, con el valor FALSO, la función comprueba si el conjunto final de reglas está vacío, y si es así, incluye la mejor regla de entre toda la población completa (sólo una, y para un solo valor de la variable objetivo). De esta forma, el algoritmo siempre obtiene reglas.

La función MejorRegla(n_i) y MejorRegla() obtienen la regla con el mayor valor de confianza para un determinado valor de la variable objetivo y para la población completa, respectivamente. Es importante resaltar que la función de cribado se aplica sobre la mejor población, que es la población obtenida con el mejor fitness global a lo largo de toda la ejecución.

Capítulo 4. Análisis, diseño e implementación de la librería de algoritmos

En esta sección se describen las tareas relacionadas con la ingeniería del software, como el análisis de requisitos, el diseño de nuestra librería de algoritmos, el diagrama de clases y el diagrama de secuencia. También explicaremos la implementación de la misma así como las herramientas usadas para la misma.

4.1. Análisis de requerimientos

Comenzaremos haciendo un análisis de los requerimientos del sistema. Se define el análisis de requisitos como "el proceso del estudio de las necesidades de los usuarios para llegar a una definición de los requisitos del sistema, de hardware o de software, así como el proceso de estudio y refinamiento de dichos requisitos" (Estándar IEEE Std. 610 [IEEE 1990]).

El Requisito es pues "una condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado". Una vez terminada la implementación deberá verificarse que todos los requisitos que se especifiquen aquí se cumplan.

4.1.1. Requerimientos funcionales

Los requerimientos funcionales de un sistema, son aquellos que describen cualquier actividad que este deba realizar, en otras palabras, el comportamiento o función particular de un sistema o software cuando se cumplen ciertas condiciones. Por lo tanto, un requerimiento no debe de especificar el cómo se debe de realizar una tarea, si no qué tareas debe realizar el sistema. Los requerimientos funcionales de nuestro sistema serán los siguientes:

1. Permitir que el usuario pueda modificar los parámetros de cada algoritmo desde un fichero de texto.

2. El sistema debe ser capaz de mostrar por pantalla el resultado, así como en diferentes ficheros de salida clasificados según la función que se realice.

3. El sistema debe ser capaz de tratar correctamente con ficheros en el formato de datos de KEEL (.DAT), ARFF y CSV.

4. El sistema deberá crear correctamente las particiones difusas que le indique el usuario, estas serán todas iguales. Además, en la salida, si se utiliza una variable que utilice un conjunto difuso, el sistema deberá indicar claramente cuál es y los límites del mismo.

5. El sistema debe de generar mensajes informativos de lo que ocurre a cada momento y almacenarlos en un fichero. En caso de error, deberá informar del error y, si es posible, su posible solución.

4.1.2. Requerimientos no funcionales

Los requerimientos no funcionales definen las características o cualidades generales que se esperan de un sistema y establecen restricciones sobre el producto, el proceso de desarrollo de software y establecen restricciones externas que el software debe lograr. Los requerimientos no funcionales que detectamos para nuestro sistema son los siguientes:

1. El sistema deberá ser fácil de usar, por lo que una persona con poca experiencia de uso en R y con cierta experiencia en descubrimiento de subgrupos, o simplemente, conociendo el algoritmo a usar, debe ser capaz de usarlo sin problemas.
2. El sistema debe ser capaz de ejecutarse en cualquier sistema operativo donde se tenga R instalado.
3. El sistema debe tener un tiempo de respuesta aceptable, de unos segundos. Aunque este requerimiento depende mucho del tamaño del conjunto de datos, el número de evaluaciones y tamaño de población establecidos por el usuario.
4. El sistema debe ser fácilmente extensible de modo que si queremos mejorar o añadir una utilidad simplemente deberemos modificar mínimamente la librería para incluir el nuevo algoritmo.

4.1.3. Casos de uso

Con el fin de ejemplificar la interacción del usuario con el sistema en función de los requerimientos definidos anteriormente se ha diseñado un diagrama de casos de uso.

Un caso de uso de sistema es una secuencia de acciones que un sistema lleva a cabo que da lugar a un resultado de valor observable para un actor particular (alguien o algo fuera del sistema que interactúa con el sistema).

Los casos de uso son historias o casos de utilización de un sistema; no son exactamente los requerimientos ni las especificaciones funcionales, sino que ejemplifican e incluyen los requerimientos en las historias que narran.

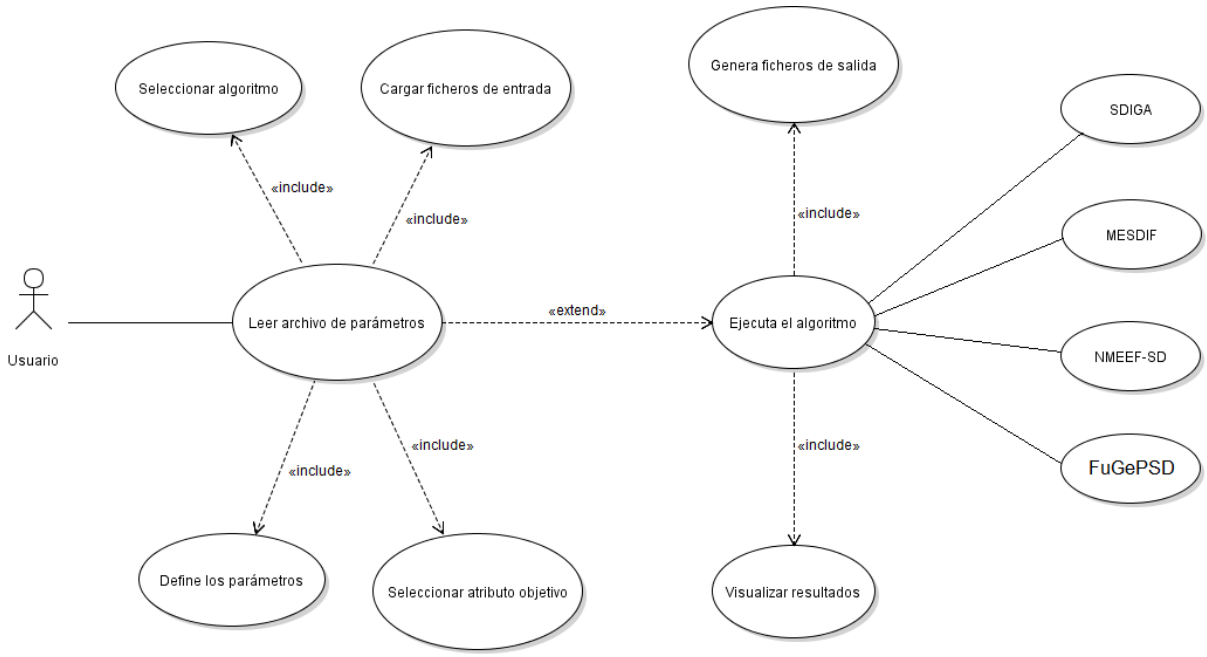


Figura 21: Diagrama de casos de uso

Para describir los casos de uso mostrados en la figura 21, usaremos la narrativa de casos de uso. La narrativa de casos de uso se puede ver como un documento narrativo que describe y especifica la secuencia de eventos de un actor que utiliza un sistema para completar un proceso, y cada una de las interacciones que el usuario tiene con el sistema.

Casos de uso	**Leer archivo de parámetros**
Actor primario	Usuario
Condición de entrada	El usuario indica la ruta del fichero de parámetros
Condición de salida	El sistema lee el fichero de parámetros y configura dichos parámetros
Flujo de eventos	
Paso	**Acción**
1	El usuario indica la ruta del fichero de parámetros.
2	El sistema lee los parámetros del fichero, seleccionando algoritmo, conjunto de datos de entrada para entrenamiento y prueba, selecciona atributo objetivo y configurando los parámetros necesarios para el algoritmo escogido.

3	El sistema carga los conjuntos de datos para entrenamiento y prueba, a partir de la dirección leída del fichero de parámetros.
4	Se muestra al usuario los parámetros leídos del fichero, y las variables leídas del conjunto de datos.
Excepciones	
Paso	**Acción**
1	a) La dirección proporcionada para el fichero de parámetros es incorrecta. En este caso se le notifica el error al usuario. Finaliza el caso de uso. b) Cualquiera de los parámetros necesarios para la ejecución tiene un formato incorrecto. En este caso se le notifica el error al usuario y cómo solucionarlo. Finaliza el caso de uso. c) La dirección proporcionada para los ficheros del conjunto de datos es incorrecta. En este caso se le notifica el error al usuario. Finaliza el caso de uso. d) El fichero del conjunto de datos tiene un formato incorrecto. En este caso se le notifica el error al usuario. Finaliza el caso de uso.

Tabla 3: Narrativa del caso de uso "Leer archivo de parámetros"

Casos de uso	**Ejecutar algoritmo**
Actor primario	Usuario
Condición de entrada	El usuario indica la ruta en el fichero de parámetros de entrada.
Condición de salida	El sistema encuentra los subgrupos que mejor se adaptan al conjunto de datos.
Flujo de eventos	
Paso	**Acción**
1	El usuario ejecuta el algoritmo indicando la ubicación del fichero de parámetros
2	El sistema devuelve los subgrupos y las medidas de calidad correspondientes, a la vez que genera los archivos de salida correspondientes.
Excepciones	
Paso	**Acción**
1	a) El sistema no encuentra ningún subgrupo que satisfaga las medidas de calidad mínimas. El sistema devuelve el mejor subgrupo obtenido en este caso. Finaliza el caso de uso.

Tabla 4: Narrativa del caso de uso "Ejecutar algoritmo"

4.2. Diseño

Una vez especificados los requisitos del sistema deberemos diseñar el sistema para cumplirlos. Esta fase es crítica para la correcta solución del sistema por lo que debe de llevarse a cabo con la máxima atención.

En nuestro caso, implementaremos nuestros algoritmos en el lenguaje de programación C++. En la sección 4.3.1 explicaremos con detalle las características del lenguaje y las herramientas utilizadas para la implementación del sistema.

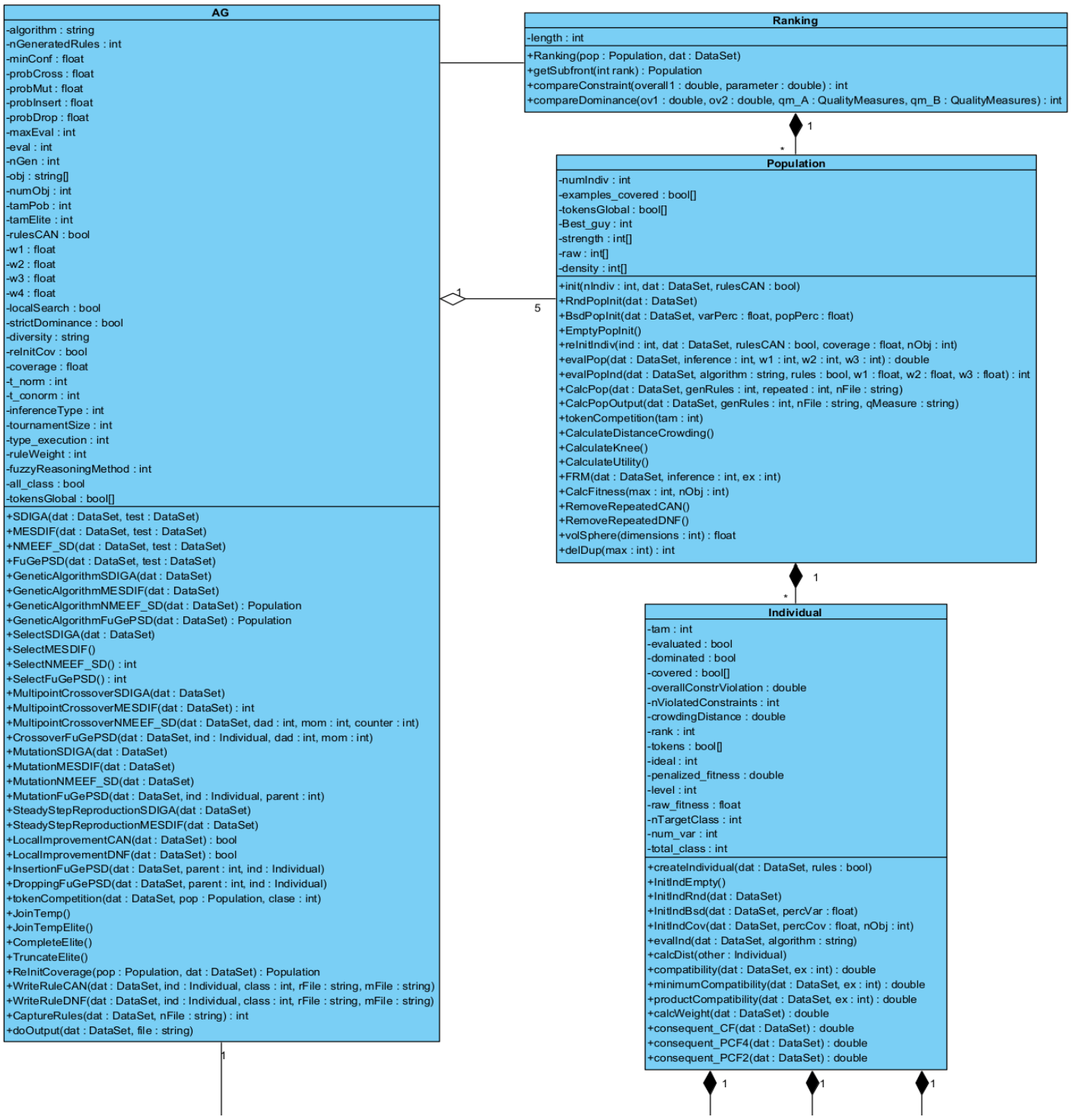
La implementación de los algoritmos se ha desarrollado utilizando como referencia el código fuente en Java de estos algoritmos, disponibles en la plataforma de código abierto KEEL [4]. Para la mejor comprensión de los diferentes elementos que forman parte de cada algoritmo, se incluyen a continuación el diagrama de clases y de secuencia de la implementación.

4.2.1. Diseño de clases del Sistema

En ingeniería del software, el diagrama de clases es un diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus propiedades, sus atributos, sus operaciones (o métodos) y las relaciones entre los objetos.

En este diagrama se representa la estructura y el comportamiento de cada uno de los objetos del sistema y sus relaciones con los demás objetos, pero no muestra información temporal, para ello se podría utilizar un diagrama de transición de estados.

Es importante destacar que, por esta misma razón, este diagrama no incluye la forma en la que se comportan a lo largo de la ejecución los distintos elementos, esa función puede ser representada a través de diagramas de comportamiento, como por ejemplo un diagrama de secuencia o un diagrama de casos de uso.


AG
-algorithm : string
-nGeneratedRules : int
-minConf : float
-probCross : float
-probMut : float
-probInsert : float
-probDrop : float
-maxEval : int
-eval : int
-nGen : int
-obj : string[]
-numObj : int
-tamPob : int
-tamElite : int
-rulesCAN : bool
-w1 : float
-w2 : float
-w3 : float
-w4 : float
-localSearch : bool
-strictDominance : bool
-diversity : string
-reInitCov : bool
-coverage : float
-t_norm : int
-t_conorm : int
-inferenceType : int
-tournamentSize : int
-type_execution : int
-ruleWeight : int
-fuzzyReasoningMethod : int
-all_class : bool
-tokensGlobal : bool[]
+SDIGA(dat : DataSet, test : DataSet)
+MESDIF(dat : DataSet, test : DataSet)
+NMEEF_SD(dat : DataSet, test : DataSet)
+FuGePSD(dat : DataSet, test : DataSet)
+GeneticAlgorithmSDIGA(dat : DataSet)
+GeneticAlgorithmMESDIF(dat : DataSet)
+GeneticAlgorithmNMEEF_SD(dat : DataSet) : Population
+GeneticAlgorithmFuGePSD(dat : DataSet) : Population
+SelectSDIGA(dat : DataSet)
+SelectMESDIF()
+SelectNMEEF_SD() : int
+SelectFuGePSD() : int
+MultipointCrossoverSDIGA(dat : DataSet)
+MultipointCrossoverMESDIF(dat : DataSet) : int
+MultipointCrossoverNMEEF_SD(dat : DataSet, dad : int, mom : int, counter : int)
+CrossoverFuGePSD(dat : DataSet, ind : Individual, dad : int, mom : int)
+MutationSDIGA(dat : DataSet)
+MutationMESDIF(dat : DataSet)
+MutationNMEEF_SD(dat : DataSet)
+MutationFuGePSD(dat : DataSet, ind : Individual, parent : int)
+SteadyStepReproductionSDIGA(dat : DataSet)
+SteadyStepReproductionMESDIF(dat : DataSet)
+LocalImprovementCAN(dat : DataSet) : bool
+LocalImprovementDNF(dat : DataSet) : bool
+InsertionFuGePSD(dat : DataSet, parent : int, ind : Individual)
+DroppingFuGePSD(dat : DataSet, parent : int, ind : Individual)
+tokenCompetition(dat : DataSet, pop : Population, clase : int)
+JoinTemp()
+JoinTempElite()
+CompleteElite()
+TruncateElite()
+ReInitCoverage(pop : Population, dat : DataSet) : Population
+WriteRuleCAN(dat : DataSet, ind : Individual, class : int, rFile : string, mFile : string)
+WriteRuleDNF(dat : DataSet, ind : Individual, class : int, rFile : string, mFile : string)
+CaptureRules(dat : DataSet, nFile : string) : int
+doOutput(dat : DataSet, file : string)
Ranking
-length : int
+Ranking(pop : Population, dat : DataSet)
+getSubfront(int rank) : Population
+compareConstraint(overall1 : double, parameter : double) : int
+compareDominance(ov1 : double, ov2 : double, qm_A : QualityMeasures, qm_B : QualityMeasures) : int
Population
-numIndiv : int
-examples_covered : bool[]
-tokensGlobal : bool[]
-Best_guy : int
-strength : int[]
-raw : int[]
-density : int[]
+init(nIndiv : int, dat : DataSet, rulesCAN : bool)
+RndPopInit(dat : DataSet)
+BsdPopInit(dat : DataSet, varPerc : float, popPerc : float)
+EmptyPopInit()
+reInitIndiv(ind : int, dat : DataSet, rulesCAN : bool, coverage : float, nObj : int)
+evalPop(dat : DataSet, inference : int, w1 : int, w2 : int, w3 : int) : double
+evalPopInd(dat : DataSet, algorithm : string, rules : bool, w1 : float, w2 : float, w3 : float) : int
+CalcPop(dat : DataSet, genRules : int, repeated : int, nFile : string)
+CalcPopOutput(dat : DataSet, genRules : int, nFile : string, qMeasure : string)
+tokenCompetition(tam : int)
+CalculateDistanceCrowding()
+CalculateKnee()
+CalculateUtility()
+FRM(dat : DataSet, inference : int, ex : int)
+CalcFitness(max : int, nObj : int)
+RemoveRepeatedCAN()
+RemoveRepeatedDNF()
+volSphere(dimensions : int) : float
+delDup(max : int) : int
Individual
-tam : int
-evaluated : bool
-dominated : bool
-covered : bool[]
-overallConstrViolation : double
-nViolatedConstraints : int
-crowdingDistance : double
-rank : int
-tokens : bool[]
-ideal : int
-penalized_fitness : double
-level : int
-raw_fitness : float
-nTargetClass : int
-num_var : int
-total_class : int
+createIndividual(dat : DataSet, rules : bool)
+InitIndEmpty()
+InitIndRnd(dat : DataSet)
+InitIndBsd(dat : DataSet, percVar : float)
+InitIndCov(dat : DataSet, percCov : float, nObj : int)
+evalInd(dat : DataSet, algorithm : string)
+calcDist(other : Individual)
+compatibility(dat : DataSet, ex : int) : double
+minimumCompatibility(dat : DataSet, ex : int) : double
+productCompatibility(dat : DataSet, ex : int) : double
+calcWeight(dat : DataSet) : double
+consequent_CF(dat : DataSet) : double
+consequent_PCF4(dat : DataSet) : double
+consequent_PCF2(dat : DataSet) : double
1
*
1
5
1
*
1
1
1
1

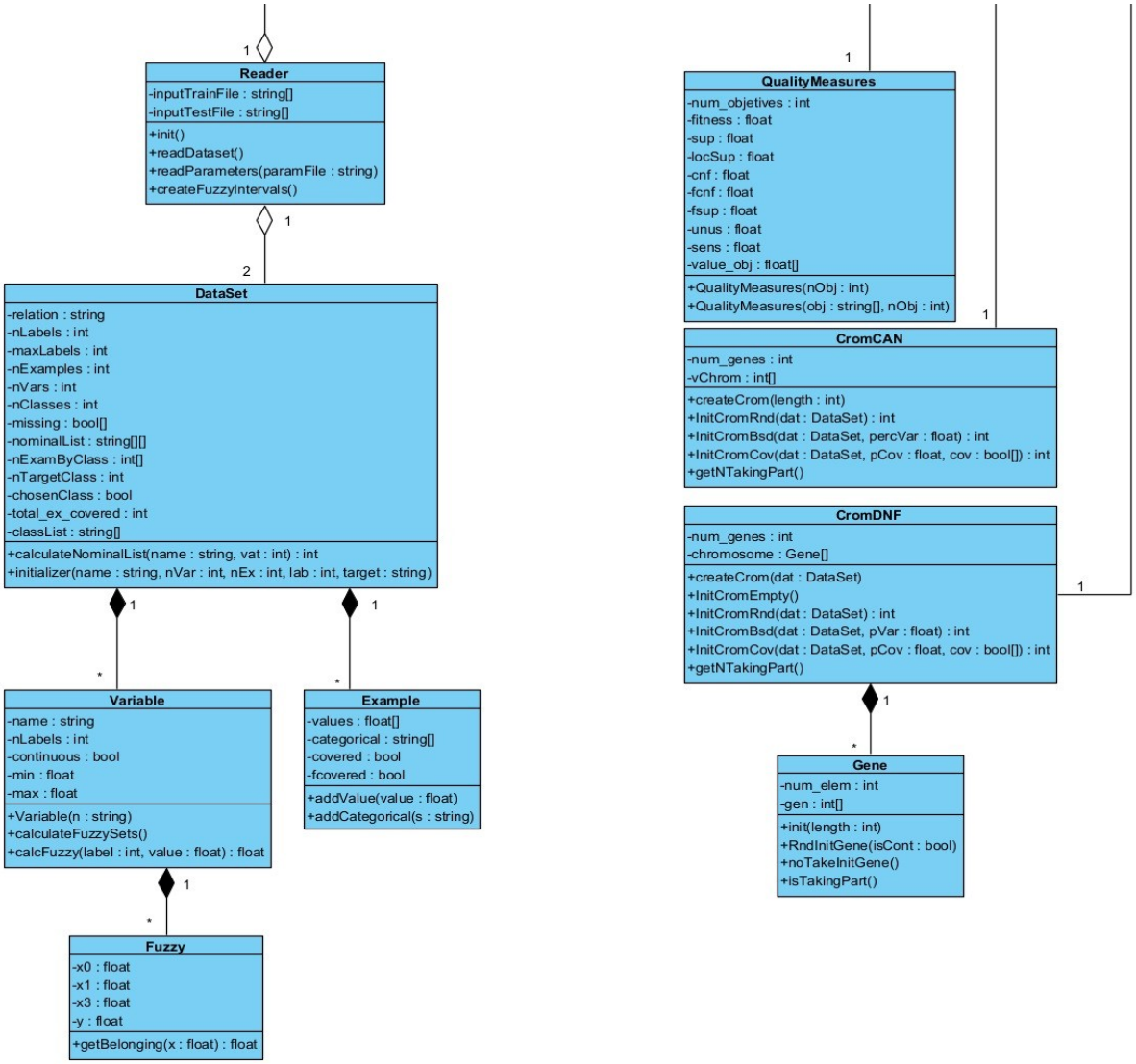
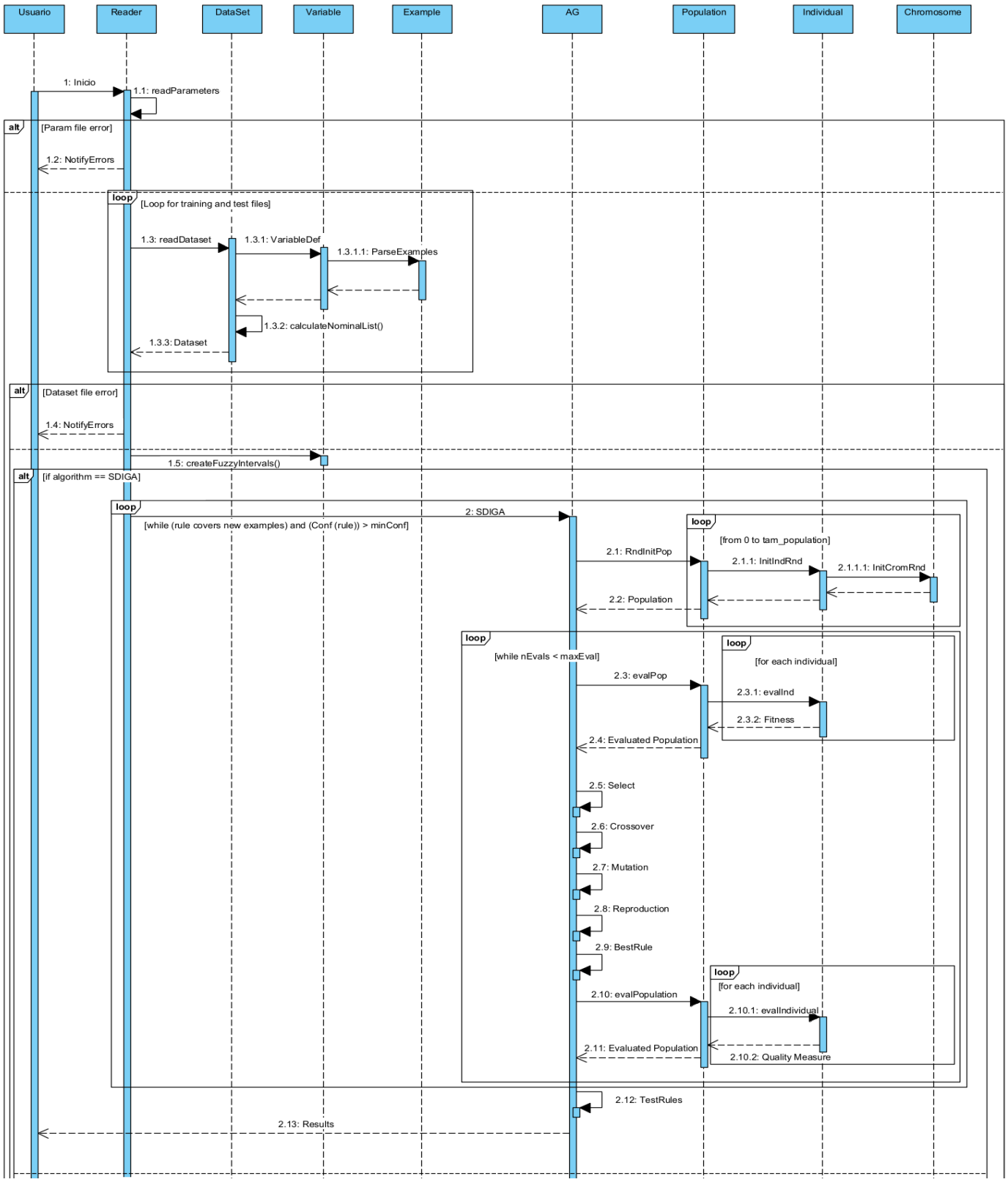
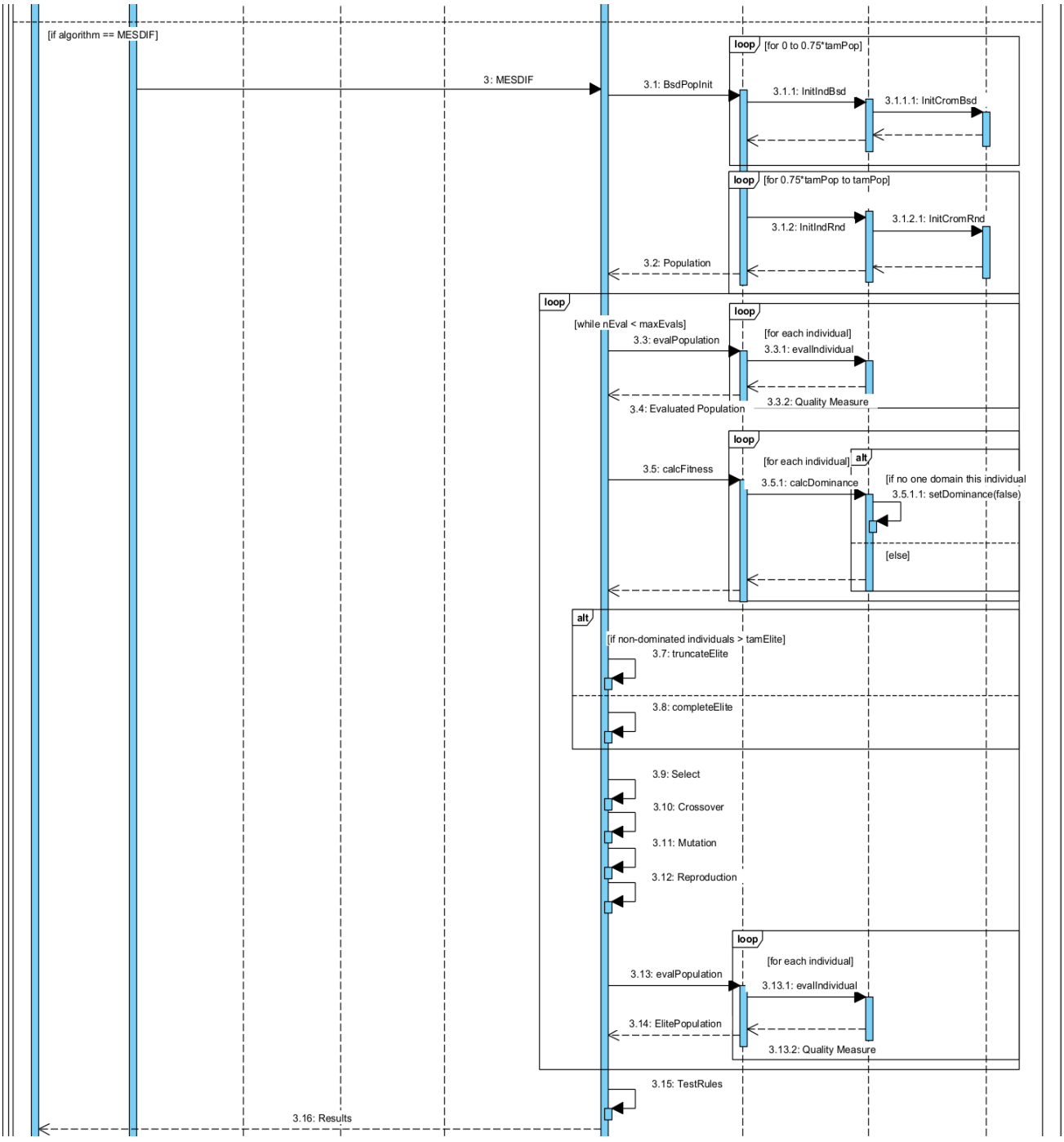


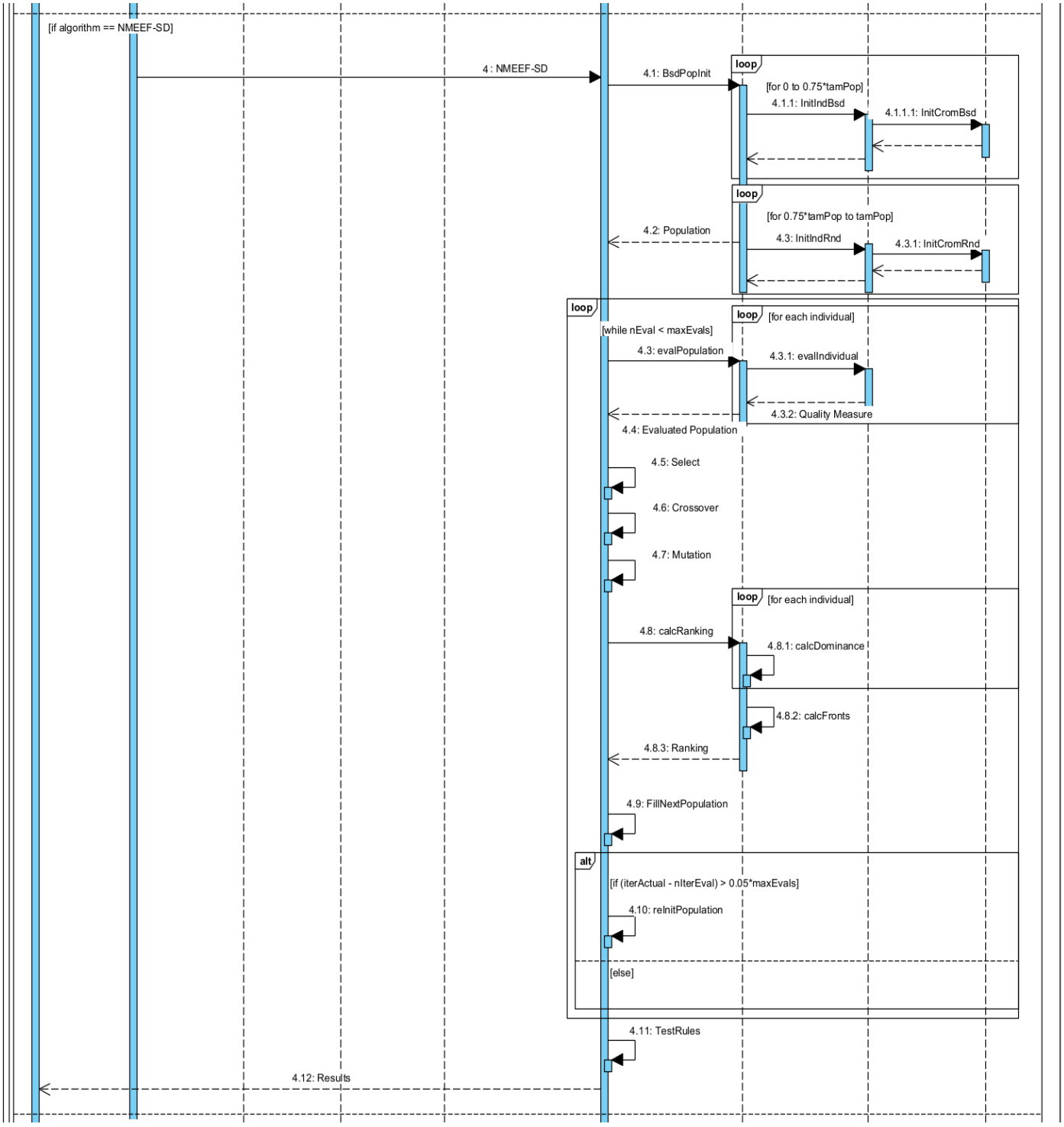
Figura 22: Diagrama de clases

4.2.2. Diseño de la secuencia de actividades del Sistema

El diagrama de secuencia es un tipo de diagrama usado para modelar interacción entre objetos en un sistema a través del tiempo. El diagrama de secuencia se podría describir como "el diagrama de clases en movimiento".


Usuario
Reader
DataSet
Variable
Example
AG
Population
Individual
Chromosome
1: Inicio
1.1: readParameters
alt
[Param file error]
1.2: NotifyErrors
loop
[Loop for training and test files]
1.3: readDataset
1.3.1: VariableDef
1.3.1.1: ParseExamples
1.3.2: calculateNominalList()
1.3.3: Dataset
alt
[Dataset file error]
1.4: NotifyErrors
1.5: createFuzzyIntervals()
alt
[if algorithm == SDIGA]
loop
[while (rule covers new examples) and (Conf (rule)) > minConf]
2: SDIGA
loop
[from 0 to tam_population]
2.1: RndInitPop
2.1.1: InitIndRnd
2.1.1.1: InitCromRnd
2.2: Population
loop
[while nEvals < maxEval]
loop
[for each individual]
2.3: evalPop
2.3.1: evalInd
2.3.2: Fitness
2.4: Evaluated Population
2.5: Select
2.6: Crossover
2.7: Mutation
2.8: Reproduction
2.9: BestRule
loop
[for each individual]
2.10: evalPopulation
2.10.1: evalIndividual
2.10.2: Quality Measure
2.11: Evaluated Population
2.12: TestRules
2.13: Results


[if algorithm == MESDIF]
3: MESDIF
loop
[for 0 to 0.75*tamPop]
3.1: BsdPopInit
3.1.1: InitIndBsd
3.1.1.1: InitCromBsd
loop
[for 0.75*tamPop to tamPop]
3.1.2: InitIndRnd
3.1.2.1: InitCromRnd
3.2: Population
loop
[while nEval < maxEvals]
loop
[for each individual]
3.3: evalPopulation
3.3.1: evalIndividual
3.3.2: Quality Measure
3.4: Evaluated Population
loop
[for each individual]
alt
3.5: calcFitness
3.5.1: calcDominance
[if no one domain this individual
3.5.1.1: setDominance(false)
[else]
alt
[if non-dominated individuals > tamElite]
3.7: truncateElite
3.8: completeElite
3.9: Select
3.10: Crossover
3.11: Mutation
3.12: Reproduction
loop
[for each individual]
3.13: evalPopulation
3.13.1: evalIndividual
3.14: ElitePopulation
3.13.2: Quality Measure
3.15: TestRules
3.16: Results


[if algorithm == NMEEF-SD]
4 : NMEEF-SD
4.1: BsdPopInit
loop
[for 0 to 0.75*tamPop]
4.1.1: InitIndBsd
4.1.1.1: InitCromBsd
loop
[for 0.75*tamPop to tamPop]
4.2: Population
4.3: InitIndRnd
4.3.1: InitCromRnd
loop
loop [for each individual]
[while nEval < maxEvals]
4.3: evalPopulation
4.3.1: evalIndividual
4.3.2: Quality Measure
4.4: Evaluated Population
4.5: Select
4.6: Crossover
4.7: Mutation
loop [for each individual]
4.8: calcRanking
4.8.1: calcDominance
4.8.2: calcFronts
4.8.3: Ranking
4.9: FillNextPopulation
alt
[if (iterActual - nIterEval) > 0.05*maxEvals]
4.10: reInitPopulation
[else]
4.11: TestRules
4.12: Results

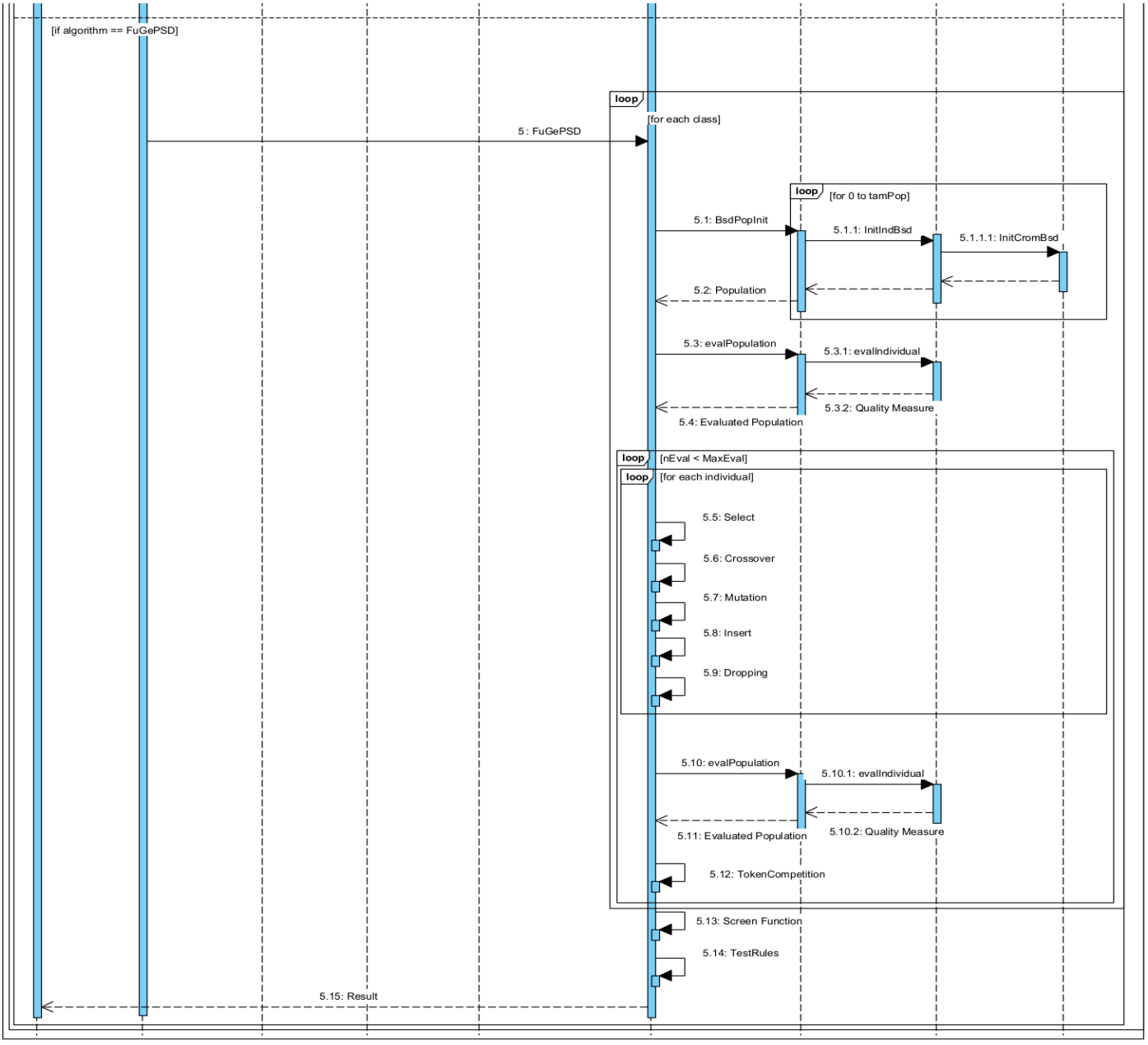


Figura 23: Diagrama de secuencia

4.3. Implementación y pruebas

En este apartado se describen las herramientas utilizadas para la implementación de los algoritmos, las características del lenguaje de programación utilizado y el desarrollo de las pruebas necesarias para validar el funcionamiento correcto de los diferentes algoritmos implementados.

4.3.1. Lenguaje de programación

El lenguaje de programación escogido será C++ montado sobre el paquete RCPP para R que incorpora dicho código C++ a R. La razón de su elección es por su ejecución más eficiente sobre R, como se demostrará más adelante.

C++ es un lenguaje de programación de alto nivel diseñado en 1979 por Bjarne Stroustrup. La intención de su creación fue extender al lenguaje de programación C, heredando su sintaxis, con mecanismos que permiten la manipulación de objetos.

Posteriormente se añadieron facilidades de programación genérica, que se sumaron a los paradigmas de programación estructurada y programación orientada a objetos.

C++ es un lenguaje de programación compilado, multiparadigma, principalmente de tipo imperativo y orientado a objetos, incluyendo también programación genérica y funcional, características estas últimas que comentaremos más adelante en el curso. Tiene un gran número de compiladores en diferentes plataformas y sistemas operativos, por lo que se le llama multiplataforma. C y C++ están muy extendidos. Casi cualquier programa o sistema están escritos o tienen alguna parte escrita en estos lenguajes (*desde un navegador web hasta el propio sistema operativo*).

Una de sus principales características es el alto rendimiento que ofrece. Esto es debido a que puede hacer llamadas directas al sistema operativo, posee gran variedad de parámetros de optimización y se integra de forma directa con el lenguaje ensamblador.

Las principales desventajas de C++ es que se trata de un lenguaje muy amplio (*con muchos años y muchas líneas de código*), tiene que tener una compilación por plataforma y su depuración se complica debido a los errores que surgen. Además el manejo de librerías es más complicado que otros lenguajes como Java o .Net y su curva de aprendizaje muy alta.

4.3.2. Herramientas para la implementación. RCPP

El software R, a pesar de estar escrito en lenguaje C, utiliza nativamente el lenguaje R. Para poder compilar y ejecutar el programa es necesario utilizar un paquete adicional del repositorio de R llamado RCPP. El paquete RCPP proporciona funciones de R, así como clases de C++ que ofrecen una integración perfecta de R y C++. Muchos tipos de datos y objetos de R se pueden mapear de un lado a los equivalentes de C++, lo que facilita tanto la escritura de código nuevo como una integración más sencilla de librerías de terceros.

4.3.2.1. Introducción

R puede ser perfectamente rápido para algunas cosas, pero a veces demasiado lento para otras. En general, R no es el lenguaje más rápido o más eficiente en memoria que existe, pero es muy fácil de usar, de compartir y produce resultados muy bonitos. C++ es muy rápido, pero no tan fácil de usar, y no fue diseñado pensando en generar resultados bonitos. Sin embargo, cuando juntamos los dos de manera inteligente, podemos ir rápido y escribir código que sea relativamente fácil de usar.

Aquí es donde RCPP entra en juego. Es un paquete R que hace que la incorporación de código C++ con código R sea bastante sencilla. Al fin y al cabo, R está escrito en C (pariente cercano de C++), por lo que C++ se integra bien. La buena gente de RStudio también ha incorporado mucho soporte para RCPP directamente en RStudio, y ha incluido un depurador bastante útil para arrancar. RcppArmadillo es un paquete complementario que le brinda acceso a toneladas de funciones útiles de álgebra lineal en C++.

El paquete Rcpp simplifica la integración del código C++ con R. Proporciona una jerarquía de clases C++ consistente, que mapea varios tipos de objetos R (vectores, matrices, funciones, entornos, . . .) a clases C++ dedicadas. El intercambio de objetos entre R y C++ se gestiona mediante conceptos sencillos, flexibles y ampliables que incluyen una amplio soporte para la biblioteca de plantillas estándar de C++ (C++ Standard Template Library). El código C++ puede compilarse, vincularse y cargarse sobre la marcha, o agregarse a través de paquetes. Se proporciona un manejo flexible de errores y excepciones. Rcpp reduce sustancialmente la barrera para los programadores que desean combinar código C++ con R.

4.3.2.2. Historia de RCPP

Rcpp apareció por primera vez en 2005 como una contribución (de Dominick Samperi) al paquete RQuantLib y se convirtió en un paquete CRAN a principios de 2006. Varios lanzamientos (todos de Samperi) siguieron en rápida sucesión bajo el nombre Rcpp. A continuación, se cambió el nombre del paquete a RcppTemplate; varios lanzamientos más siguieron durante 2006 bajo el nuevo nombre. Sin embargo, no se realizaron más lanzamientos durante 2007, 2008 o la mayor parte de 2009. Luego de algunas actualizaciones a fines de 2009, fue retirado del CRAN.

Dado el uso continuo del paquete, Dirk Eddelbuettel decidió revitalizarlo. Los nuevos lanzamientos, con el nombre inicial Rcpp, comenzaron en noviembre de 2008. Estos incluían un proceso mejorado de compilación y distribución, documentación adicional y nuevas funciones, al tiempo que conservaban la interfaz de "classic Rcpp" existente. Si bien no se describe aquí, esta API se proporcionará en un futuro previsible a través del paquete RcppClassic.

Como reflejo de la evolución de los estándares de codificación de C++, Dirk Eddelbuettel y Romain François (con contribuciones clave de Doug Bates, John Chambers y JJ Allaire) comenzaron un importante rediseño del código base en 2009 que agregó numerosas características nuevas. Esta nueva API es su enfoque actual, con la intención de ampliar y dar soporte a la API en el desarrollo futuro del paquete Rcpp.

4.3.2.3. Características de RCPP

Cuando trabajas con R, probablemente ya hayas trabajado con el paquete RCPP sin saberlo. En la página CRAN del paquete RCPP, verá sus dependencias inversas y sugerencias. Es decir, todos los paquetes que usan RCPP. La lista es enorme y sigue creciendo.

El uso de C++ a través de RCPP es especialmente útil cuando:

- se trabaja con bucles, ya que no se pueden vectorizar facilmente utilizando funciones como *apply* en su lugar, porque las iteraciones subsecuentes dependen de las anteriores.
- se trabaja con algoritmos iterativos en los que ciertas evaluaciones de funciones deben calcularse con mucha frecuencia.
- se trabaja con funciones recursivas, o problemas que involucran llamar a funciones millones de veces. La sobrecarga de llamar a una función en C++ es mucho menor que en R.

- se trabaja en problemas que requieren estructuras de datos y algoritmos avanzados que R no proporciona. A través de la biblioteca de plantillas estándar (STL), C++ tiene implementaciones eficientes de muchas estructuras de datos importantes, desde maps ordenados hasta colas dobles.
- se trabaja en un algoritmo que requiere operaciones matemáticas exigentes.

4.3.2.4. Comparación de eficiencia entre R y RCPP

Para resumir, las llamadas a funciones se encuentran entre las partes de menor rendimiento del lenguaje R. Operar sobre objetos y el lenguaje en sí mismo nos brinda funciones muy poderosas, pero el estado requerido y la verificación de pila para las llamadas a funciones es uno de los precios que pagamos.

Y como la secuencia de Fibonacci tiene una definición tan simple, el programa R simple se puede traducir fácilmente, lo que nos brinda un buen ejemplo del poder de C ++, particularmente para la evaluación de funciones.

Además mostraremos los dos enfoques para resolver dicho problema, enfoque recursivo y enfoque iterativo.

```r
fibR_it = function (n){
    x = rep(0, n)
    x[1] = 1
    x[2] = 2

    for(i in 3:n){
        x[i] = x[i-2] + x[i-1]
    }
    return(x)
}
```

```r
fibR_Rec <- function(n) {
    if((n == 0) | (n == 1))
        return (1)
    else
    return (fibR_Rec(n - 1)
          + fibR_Rec(n - 2))
}
```

```r
// [[Rcpp::export]]
IntegerVector fibCpp_it(int n) {
    IntegerVector x(n);
    x[0] = 1;
    x[1] = 2;
    for (int i = 2 ; i < n ; i++){
        x[i] = x[i-2] + x[i-1];
    }
    return(x);
}
```

```r
// [[Rcpp::export]]
int fibCpp_Rec(int n) {
    if((n==0) | (n==1))
        return 1;
    else
        return  fibCpp_Rec(n-1)
              + fibCpp_Rec(n-2);
}
```

Figura 24. Funciones usando los diferentes enfoques de POO de R y RCPP

Como se puede apreciar en la figura 25 abajo, las funciones en C++ tienen tiempos varias veces más pequeños que su versión en R. Especialmente el enfoque recursivo, que es cientos de veces más lento.

```
> microbenchmark(fibR_it(20),
+                fibR_Rec(20),
+                fibCpp_it(20),
+                fibCpp_Rec(20))
Unit: microseconds
           expr      min       lq      mean   median       uq      max neval cld
    fibR_it(20)      3.5     3.90     7.613     7.55    10.40    18.4   100  a
   fibR_Rec(20) 10725.5 11189.35 12135.111 11650.75 12250.20 16703.9   100   b
  fibCpp_it(20)     1.1     1.30     3.897     2.10     5.40    18.3   100  a
 fibCpp_Rec(20)    18.9    19.35    22.190    20.20    23.45    66.8   100  a
> |
```

Figura 25. Comparativa de tiempos de ejecución de funciones de R y RCPP

4.3.3. Entorno de desarrollo

Para desarrollar los algoritmos en el lenguaje de programación se ha utilizado Netbeans, principalmente porque ya tengo experiencia previa utilizando éste entorno de desarrollo.

En primer lugar, Netbeans es gratuito y de código abierto. A pesar de que Netbeans es un IDE especialmente dirigido para la creación de aplicaciones por medio del lenguaje Java, no obstante, es al mismo tiempo, multiplataforma, ya que Netbeans posee herramientas y framework de aplicaciones que incluye soporte para trabajar con otros lenguajes de programación, como el caso de PHP, Phyton o como en este caso C/C++.

El empaquetado en RCPP sin embargo, ha sido realizado sobre RStudio. RStudio es un entorno de desarrollo integrado (IDE) para el lenguaje de programación R. Incluye una consola, editor de sintaxis que apoya la ejecución de código, así como herramientas para el trazado, la depuración y la gestión del espacio de trabajo. Puede ejecutar código R directamente desde el editor de código fuente.

También permite crear y compilar paquetes para R de forma rápida y sencilla. Es claramente, el mejor entorno de desarrollo para R.

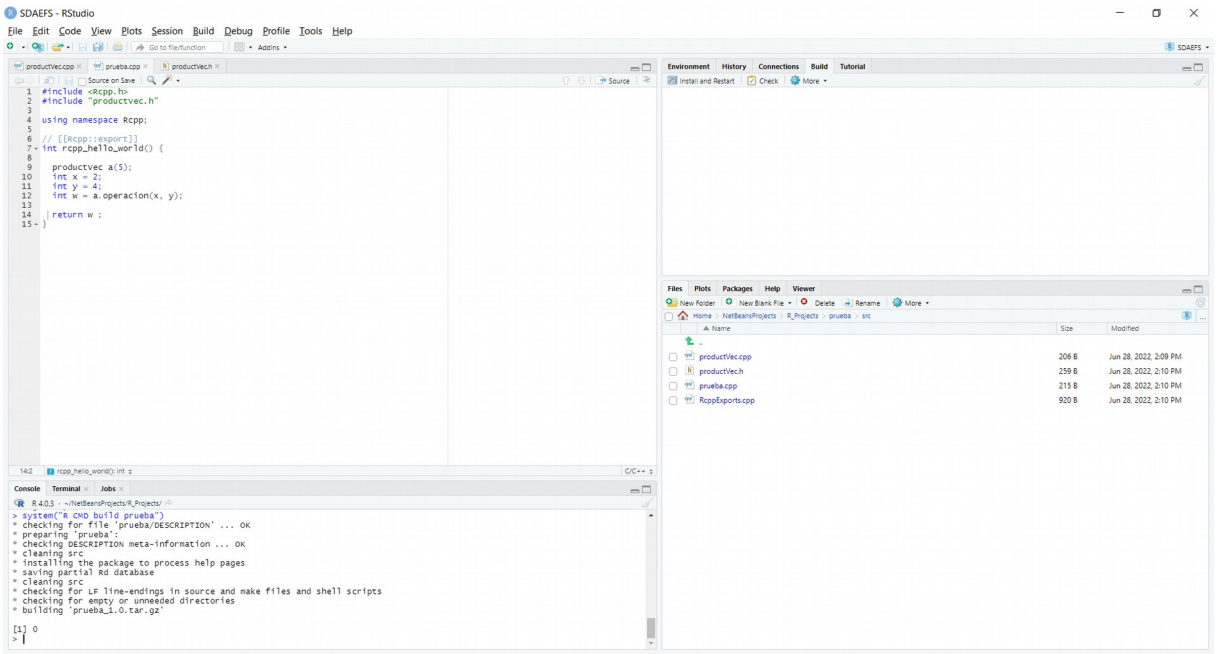


Figura 26. RStudio

4.3.4. Pruebas

Para la comprobación de la correcta funcionalidad de la implementación se ha utilizado como apoyo los algoritmos implementados en RCPP y R, con la finalidad de verificar que a igualdad de parámetros y datos, las salidas son en mayor o menor medida similares.

A continuación se recogen los resultados obtenidos con una serie de bases de datos de prueba. En la sección 5.1 se describen las características de estas bases de datos en la tabla 13.

a) Dataset iris

Reglas CAN			
	SDIGA - R	**SDIGA - RCPP**	**Coincidencia**
nº reglas	3	3	100 %
nº variables	3,333333	3	89,99 %
unusualness	0,182222	0,133827	73,44 %
fconfidence	0,831080	0,828471	99,68 %
sensitivity	0,86	1	116,27 %
time	59,6033 s	5,8956 s	9,89 %

Tabla 5: Resultados de prueba con el dataset Iris para SDIGA

Reglas CAN			
	MESDIF - R	**MESDIF - RCPP**	**Coincidencia**
nº reglas	9	9	100 %
nº variables	2,888889	2,821429	97,66 %
unusualness	0,081052	0,093708	115,61 %
fconfidence	0,579209	0,708085	122,25 %
sensitivity	0,582222	0,81	139,12 %
time	57,67 s	3,111 s	5,39 %

Tabla 6: Resultados de prueba con el dataset Iris para MESDIF

Reglas CAN			
	NMEEF-SD - R	**NMEEF-SD - RCPP**	**Coincidencia**
nº reglas	26	25,33	97,43 %
nº variables	3,805556	2,892039	75,99 %
unusualness	0,019976	0,127233	636,92 %
fconfidence	0,958736	0,808291	84,3 %
sensitivity	0,091921	0,885320	963,13 %
time	9,996 s	4,808 s	48,09 %

Tabla 7: Resultados de prueba con el dataset Iris para NMEEF-SD

Reglas CAN			
	FuGePSD - R	**FuGePSD - RCPP**	**Coincidencia**
nº reglas	8	3	37,5 %
nº variables	2,301347	3	130,35 %
unusualness	0,120533	0,880370	730,39 %
fconfidence	0,823009	0,983061	119,447 %
sensitivity	0,50	1	200 %
time	35,49 s	18,84 s	53,08 %

Tabla 8: Resultados de prueba con el dataset Iris para FuGePSD

b) Dataset ecoli

Reglas CAN			
	SDIGA - R	**SDIGA - RCPP**	**Coincidencia**
nº reglas	8	9	112,5 %
nº variables	3,625	4,888889	134,86 %
unusualness	0,010164	0,045147	444,16 %
fconfidence	0,263012	0,252191	95,88 %
TPR	0,96875	0,678497	70,03 %
time	311,413 s	32,323 s	10,37 %

Tabla 9: Resultados de prueba con el dataset ecoli para SDIGA

Reglas CAN			
	MESDIF - R	**MESDIF - RCPP**	**Coincidencia**
nº reglas	24	24	100 %
nº variables	3,930556	4,451732	113,25 %
unusualness	0,018666	0,013645	73,1 %
fconfidence	0,421579	0,130928	31,05 %
sensitivity	0,644315	0,671681	104,24 %
time	321,09 s	16,964 s	5,28 %

Tabla 10: Resultados de prueba con el dataset ecoli para MESDIF

Reglas CAN			
	NMEEF-SD - R	**NMEEF-SD - RCPP**	**Coincidencia**
nº reglas	74	10,33	13,96 %
nº variables	4,686311	3,451852	73,65 %
unusualness	0,023205	0,089577	386,02 %
fconfidence	0,864669	0,600349	69,43 %
sensitivity	0,306080	0,657651	214,86 %
time	38,11 s	13,118 s	34,42 %

Tabla 11: Resultados de prueba con el dataset ecoli para NMEEF-SD

Reglas CAN			
	FuGePSD - R	**FuGePSD - RCPP**	**Coincidencia**
nº reglas	8	12,333333	154,16 %
nº variables	4,166667	5,403810	129,69 %
unusualness	0,029787	0,469846	1577 %
fconfidence	0,402785	0,324378	80,53 %
sensitivity	0,5	0,456178	91,23 %
time	561,02 s	100,785 s	17,96 %

Tabla 12: Resultados de prueba con el dataset ecoli para FuGePSD

Como se puede apreciar, existen ciertas diferencias entre ambas implementaciones. Esto se debe principalmente a que la forma de generar un número aleatorio para seleccionar individuos, al elegir puntos de cruce, valores de mutación, etc. difiere de un lenguaje a otro. La mas mínima variación en la generación de los número aleatorios puede hacer que varíe mucho los resultados de uno a otro.

Los algoritmos evolutivos son algoritmos estocásticos y su resultado depende en gran medida de la secuencia de números aleatorios generada.

Capítulo 5. Experimentación y conclusiones finales

En este capítulo final de la memoria se exponen una serie de experimentaciones con bases de datos públicas, con el fin de analizar los resultados obtenidos y se concluye con el trabajo futuro para la mejora de este proyecto.

5.1. Características de la experimentación sobre bases de datos públicas

A continuación se detallarán una serie de experimentaciones llevadas a cabo con diferentes bases de datos públicas. Una vez verificada la correcta ejecución de los algoritmos, realizaremos pruebas sobre diferentes conjuntos de datos para posteriormente analizar los resultados. Para ello, tomaremos distintos conjuntos de datos públicos, ejecutaremos los algoritmos sobre ellas y extraeremos conclusiones los resultados obtenidos.

Los conjuntos de datos públicas los hemos obtenido del conjunto de datasets incluidos en el repositorio del software KEEL [4]. Las características de dichas bases de datos se recogen en la siguiente tabla:

Nombre	Nº ejemplos	Nº Variables de entrada	Tipo Variables (E / R / N)	Nº valores objetivo
australian	3450	14	5/3/6	2
banana	26500	2	0/2/0	2
car	8640	6	0/0/6	4
ecoli	1680	6	0/6/0	2
flare	5330	11	0/0/11	6
german	5000	20	0/7/13	6
haberman	1530	3	0/3/0	2
Iris	750	4	0/4/0	3
Monk-2	2160	6	0/6/0	2
Tic-Tac-Toe	4790	9	0/0/9	2

Tabla 13: Características de las bases de datos para la experimentación

Los tipos de variables se divides en (E)nteros, (R)eales o (N)ominales.

La experimentación se llevará a cabo siguiendo el método de Validación Cruzada de k pliegues (k-fold Cross Validation). Antes de nada, la validación cruzada es la técnica estadística normalmente utilizada para analizar y diagnosticar si un modelo describe adecuadamente la variabilidad espacial de los datos y garantizar que los resultados son independientes de la partición entre datos de entrenamiento y prueba.

Concretamente la validación cruzada de k pliegues consiste en realizar k ejecuciones del algoritmo y calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones. Se utiliza en entornos donde el objetivo principal es la predicción y se quiere estimar la precisión de un modelo que se llevará a cabo a la práctica.

El método divide el conjunto original de ejemplos en k subconjuntos de igual tamaño y en cada una de las diferentes ejecuciones, tomaremos un subconjunto diferente para que sea el conjunto de prueba y el resto para el conjunto de entrenamiento. La Figura 27 que se muestra a continuación ejemplifica cómo se realizaría este método.

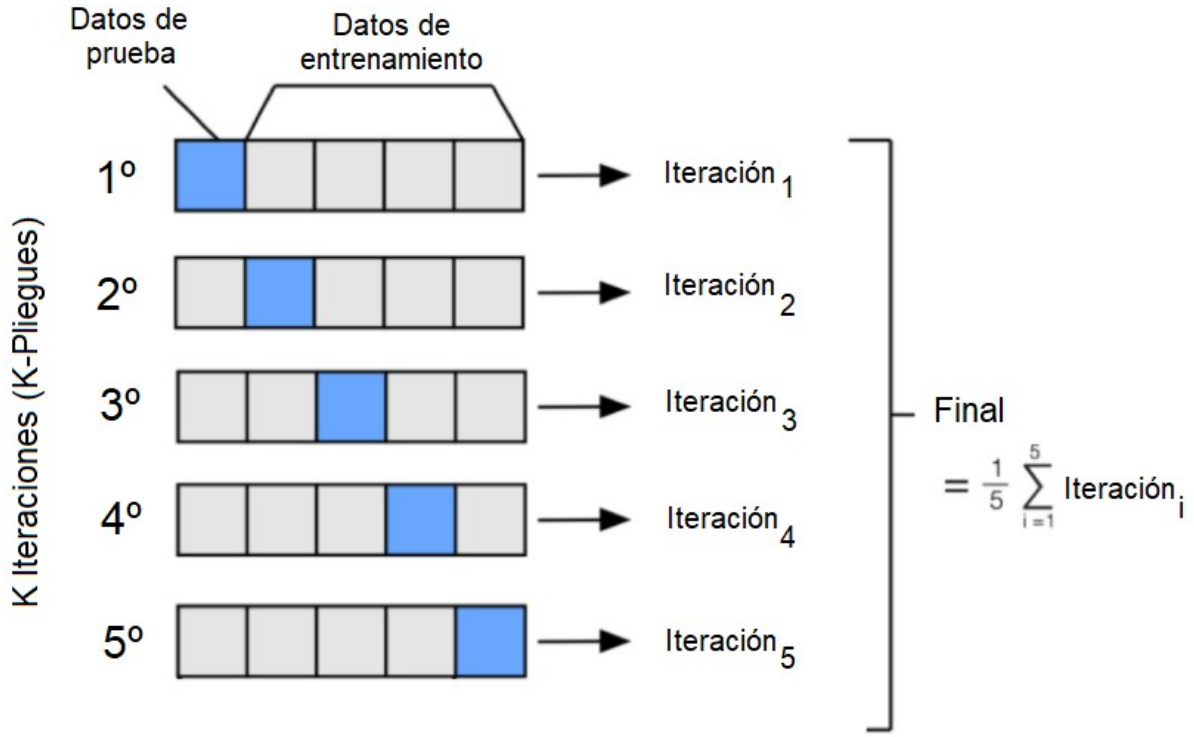


Figura 27: Esquema de validación cruzada en k pliegues

Para esta experimentación utilizaremos un valor de **k igual a 5**, y se utilizarán tres semillas aleatorias diferentes, de las que finalmente se utilizará la media de los

resultados obtenidos de cada una de las ejecuciones en que se han utilizado. Una semilla aleatoria es un número (o vector) utilizado para inicializar un generador de números pseudo-aleatorios.

Una secuencia de números de un generador de números pseudo-aleatorios está completamente determinada por la semilla: por lo tanto, si un generador de números pseudo-aleatorios es inicializado de nuevo con la misma semilla, producirá la misma secuencia de números, siguiendo una distribución de probabilidad de manera pseudo-aleatoria.

Por lo tanto nuestra experimentación constará de $5 \cdot 8 = 40$ ejecuciones para cada algoritmo (3 algoritmos) y para cada semilla (3 semillas), haciendo un total de 360 ejecuciones. Los parámetros utilizados por los algoritmos durante la experimentación se recogen en las siguientes tablas:

a) SDIGA:

Parámetro	Valor
seed	0, 1 y 1000
nLabels	3
nEval	10000
popLength	100
crossProb	0.6
mutProb	0.1
minConf	0.6
RulesRep	CAN
Obj1	CSUP
Obj2	CCNF
Obj3	Null
w1	0.7
w2	0.3
w3	0.0
lSearch	YES
targetClass	Null

Tabla 14: Parámetros utilizados para SDIGA

b) MESDIF:

Parámetro	Valor
seed	0, 1 y 1000
nLabels	3
nEval	10000
popLength	100
eliteLength	3
crossProb	0.6
mutProb	0.1
RulesRep	CAN
Obj1	CSUP
Obj2	CCNF
Obj3	Null
Obj4	Null
targetClass	Null

Tabla 15: Parámetros utilizados para MESDIF

c) NMEEF-SD:

Parámetro	Valor
seed	0, 1 y 1000
nLabels	3
nEval	1000
popLength	50
crossProb	0.6
mutProb	0.1
minConf	0.6
RulesRep	CAN
Obj1	CSUP
Obj2	CCNF
Obj3	Null
StrictDominance	yes
Diversity	CROWDING
ReInitCob	yes
percCov	0.5
targetClass	Null

Tabla 16: Parámetros utilizados para NMEEF-SD

d) FuGePSD:

Parámetro	Valor
seed	0, 1 y 1000
nLabels	3
nEval	1000
popLength	50
crossProb	0.6
mutProb	0.1
InsertProb	0,15
DropProb	0,15
minConf	0.6
Obj1	CSUP
Obj2	CCNF
Obj3	UNUS
W1	0,5
W2	0,3
W3	0,2
tournamentSize	3
inferenceType	ARITHMETIC_MEAN
RuleWeight	AVERAGE_PENALIZED_CERTAINTY_FACTOR
t-norm	PRODUCT
ALL_CLASS	YES

Tabla 17: Parámetros utilizados para FuGePSD

5.2. Resultados obtenidos y análisis

En la Tabla 18 se recogen los resultados obtenidos, realizando un análisis comparativo de los mismos a continuación.

Dataset	Algoritmo	NºReg.	NºVars	WRacc	FConf	CConf	TPR	FPR	Time	Time R
australian	SDIGA	2,00	**2,00**	0,178813	0,141603	0,141603	0,144928	**0,855072**	26,067333	103,32
	MESDIF	6,00	7,355556	0,028816	0,315521	0,350945	0,130757	0,175362	**7,357**	93,97
	NMEEF-SD	**18,33**	4,097756	0,036898	0,137270	0,177806	0,098428	0,249389	9,846333	**14,106**
	FuGePSD	2,333333	3,166667	**0,530516**	**0,455014**	**0,533128**	**0,593237**	0,547409	53,817333	2584,16
banana	SDIGA	2,00	**2,00**	0,000000	0,494379	0,500000	**0,999623**	**0,999623**	57,498333	1912,706
	MESDIF	**6,00**	2,111111	0,006048	0,494844	0,487088	0,692977	0,718029	32,423333	1127,573
	NMEEF-SD	2,67	2,333333	0,048119	**0,718009**	**0,691265**	0,642938	0,514183	**23,805333**	**116,19**
	FuGePSD	1,333333	**2,00**	**0,567955**	0,497722	0,550742	0,657200	0,530752	232,574	8517,623
car	SDIGA	5,00	**2,00**	0,057108	0,514699	0,514699	**0,499319**	**0,222553**	25,115333	131,043
	MESDIF	12,00	3,914141	0,016573	0,224690	0,224690	0,370937	0,121376	**17,582**	72,63
	NMEEF-SD	**28,00**	3,061905	0,008404	**0,805033**	**0,805033**	0,237628	0,202280	27,194	**15,486**
	FuGePSD	2,00	**2,00**	**0,549070**	0,614583	0,614583	0,409892	0,108217	150,46833	5781,826
ecoli	SDIGA	9,00	4,888889	0,045147	0,252191	0,238315	**0,678497**	0,261297	32,323333	311,4133
	MESDIF	**24,00**	4,451732	0,013645	0,130928	0,135119	0,671681	**0,342193**	16,964	321,09
	NMEEF-SD	10,33	**3,451852**	0,089577	**0,600349**	**0,577747**	0,657651	0,167309	**13,118667**	**38,11**
	FuGePSD	12,333333	5,403810	**0,469846**	0,324378	0,282006	0,456178	0,113901	100,78566	561,02
flare	SDIGA	6,00	**2,166667**	0,008889	0,126548	0,126548	0,493433	**0,605066**	31,445333	152,473
	MESDIF	**18,00**	5,979167	0,003500	0,152996	0,152996	0,280805	0,190650	**18,866**	89,776
	NMEEF-SD	6,00	3,822222	0,008178	**0,457496**	**0,457496**	0,024380	0,066809	28,133667	**22,043**
	FuGePSD	9,00	3,009921	**0,497829**	0,216508	0,216508	**0,732019**	0,328964	204,25066	4651,368
german	SDIGA	2,666667	**2,777778**	0,002117	0,663605	0,661530	0,501353	**0,741333**	36,539333	82,026
	MESDIF	6,00	9,855555	0,000217	0,138889	0,138889	0,034000	0,077889	14,029333	65,61
	NMEEF-SD	**13,33**	3,645833	0,001079	**0,753398**	**0,729439**	0,235859	0,144532	**12,585**	**12,55**
	FuGePSD	1,00	3,666667	**0,719143**	0,685151	0,672857	**0,542556**	0,433333	95,607333	1827,283
haberman	SDIGA	3,00	**2,00**	0,000737	0,591157	0,579181	**0,986817**	**0,984496**	6,334	169,143
	MESDIF	6,00	2,711111	0,011611	0,536557	0,507568	0,717139	0,648511	4,471333	68,273
	NMEEF-SD	**19,00**	2,687651	0,001732	**0,741502**	0,727195	0,629288	0,581615	**4,315667**	**10,86**
	FuGePSD	1,00	2,333333	**0,750109**	0,728832	**0,754676**	0,681481	0,625514	20,262333	137,01
iris	SDIGA	3,00	3,00	0,133827	0,828471	0,636058	**1,000000**	0,397778	5,895667	59,603
	MESDIF	9,00	**2,821429**	0,093708	0,708085	0,615741	0,810000	**0,469861**	**3,111**	57,67
	NMEEF-SD	**25,33**	2,892039	0,127233	0,808291	0,664603	0,885320	0,453276	4,808	**9,996**
	FuGePSD	3,00	3,00	**0,880370**	**0,983061**	**0,740088**	**1,000000**	0,179444	18,840333	35,49
monk-2	SDIGA	4,00	**2,00**	0,096065	0,777778	0,729167	**0,652778**	**0,263889**	20,285	174,276
	MESDIF	6,00	3,133333	0,050317	0,641049	0,641049	0,374571	0,190116	**6,544333**	86,556
	NMEEF-SD	**23,00**	3,290462	0,037288	0,790672	0,751392	0,280376	0,131764	6,773	**12,376**
	FuGePSD	2,00	2,333333	**0,648034**	**0,828704**	**0,796296**	0,462848	0,179051	26,716667	261,696
tic-tac-toe	SDIGA	4,00	**2,00**	0,048177	0,693829	0,693829	**0,519457**	**0,299630**	17,692333	51,793
	MESDIF	6,00	5,166667	0,004921	0,390193	0,390193	0,138367	0,128624	**5,126667**	27,1
	NMEEF-SD	**29,67**	3,699134	0,008604	**0,780262**	**0,780262**	0,165029	0,118860	11,322667	**7,766**
	FuGePSD	1,333333	2,333333	**0,658047**	0,722746	0,722746	0,328889	0,179643	62,092333	1128,745

Tabla 18: Resultados de las experimentaciones utilizando representación canónica

Observando detenidamente los resultados obtenidos, llegamos a las siguientes conclusiones, que dividiremos según su medida de calidad respectivamente:

- **Nº de reglas**: En general, la cantidad de reglas generadas por SDIGA y FuGePSD son notoriamente las menores entre los tres algoritmos (sin contar MESDIF). Aunque en los casos en que FuGePSD presentan una sola regla puede deberse a que, como se mencionó anteriormente, si las reglas generadas por FuGePSD no superan el umbral mínimo de Confianza o Sensibilidad (Función Pantalla), se escoge la mejor regla según la Confianza. Es decir, FuGePSD siempre devuelve al menos una regla. Sin embargo, el algoritmo NMEEF-SD suele ser el que más reglas genera.

  Por otro lado MESDIF debe ser mencionada a parte, ya que no es comparable con respecto al número de reglas. Eso se debe a que el número de reglas que obtiene este algoritmo depende completamente del valor de la población élite. Siendo siempre igual al tamaño de la población de élite por el número de valores de la variable objetivo.

- **Nº de variables**: El número de variables es una medida que influye mucho en otras medidas. Y es que, cuanto menor es el número de variables en la regla, más generales e interpretables son, sus antecedentes cubren más ejemplos, mejorando la cobertura y la sensibilidad (TPR). Además al reducir el espacio de búsqueda, se reduce a su vez el tiempo de ejecución.

  Por otro lado, cuando en una regla interviene un alto número de variables se vuelve más específica y aumenta la precisión y la confianza de la regla, en perjuicio de las medidas arriba mencionadas.

  En este sentido, SDIGA es el que tiene reglas con un menor número de variables e interpretables en promedio, seguido de cerca por FuGePSD.

- **Inusualidad**: En esta medida se destaca FuGePSD como merecido ganador, puesto que este algoritmo pone mucho énfasis en encontrar subgrupos novedosos e interesantes. De entre el resto de algoritmos, ninguno se destaca sobre otro.

- **Confianza**: Esta es una medida de la precisión de las reglas, por lo que cuanto más específicas sean, mejor. El algoritmo que, en promedio, obtiene los valores más altos de confianza es NMEEF-SD seguido por FuGePSD. Con respecto a NMEEF-SD, suele generar una alta cantidad de reglas, específicas y por tanto muy precisas. El caso contrario es MESDIF, con los

peores resultados respecto a confianza, que debido a la función de truncado, que genera reglas más generales y con mayor diversidad.

- **Sensibilidad o TPR (True Positive Rate)**: esta medida es la proporción de casos positivos que están bien detectados por la prueba. Siendo SDIGA, por la alta cobertura de sus reglas, el algoritmo que mejores resultados ha mostrado en esta medida. A este le sigue el algoritmo FuGePSD

- **Especificidad o FPR (False Positive Rate)**: esta medida es la proporción de casos negativos que están bien detectados por la prueba. Y como se puede ver en la tabla de arriba, SDIGA ha sido quién mejor ha clasificado los casos negativos.

- **Relación entre Sensibilidad y Especificidad**: Una prueba ideal rara vez pasa por alto lo que está buscando es decir, es sensible y rara vez la confunde con otra cosa es decir, es específica. Conforme a las probabilidades de sensibilidad y especificidad, se tienen las siguientes posibilidades:

  - Alta sensibilidad y alta especificidad: el modelo escogido maneja perfectamente la clase.

  - Alta sensibilidad y baja especificidad: El modelo escogido detecta bien la clase, pero también incluye muestras de otra clase.

  - Baja sensibilidad y alta especificidad: El modelo escogido no detecta bien la clase, pero cuando lo hace es altamente confiable.

  - Baja sensibilidad y baja especificidad: El modelo escogido no logra detectar bien la clase.

- **Tiempo de ejecución**: Como se puede ver en la tabla 18, el algoritmo con los tiempos de ejecución más cortos para RCPP es MESDIF, seguido por NMEEF-SD. Sin embargo, resalta más los largos tiempos de ejecución del algoritmo FuGePSD.

  En comparación, los tiempos de ejecución en NMEEF-SD para R son sin duda los más rápidos entre los cuatro algoritmos. Pero igualmente son mucho más lentos que los ejecutados sobre RCPP, esto se cumple especialemente para el algoritmo FuGePSD, donde los tiempos son tan altos que tardan horas en finalizar la ejecución, siendo muy poco eficiente.

Este último punto es el más relevante, al fin y al cabo la mejora de los tiempos ejecución fue propósito principal del desarrollo de este proyecto. Y como se puede ver, hemos cumplido sobradamente este objetivo habiendo mejorado en varias veces los tiempos de ejecución con los algoritmos implementados para RCPP.

5.3 Conclusiones finales y trabajo futuro

Habiendo expuesto ya todo el desarrollo del proyecto y la teoría asociada, debemos comprobar si se han cumplido los objetivos propuestos, y en qué medida.. Por último, se expondrán ideas de cómo se podría mejorar este proyecto en el futuro.

5.3.1. Conclusiones finales

Vistos los datos extraídos en las pruebas realizadas sobre bases de datos públicas podemos extraer conclusiones:

- Se ha realizado un estudio sobre la minería de datos, concretamente sobre la tarea de descubrimiento de subgrupos, las medidas de calidad utilizadas por estos modelos y sus técnicas. Profundizando en los algoritmos evolutivos, los cuales han sido el eje central de este proyecto.

- Se han pasado por el proceso de ingeniería del software para realizar el análisis y diseño de una librería de algoritmos para el software R utilizando el lenguaje de programación C++, en el cual se han implementado cuatro algoritmos de descubrimiento de subgrupos, por medio del paquete RCPP. Además, los algoritmos se han unificado en un único paquete de R para su fácil utilización e instalación.

- El sistema lee un fichero de parámetros al inicio del programa, a partir del cual se extraen los parámetro necesarios para cada algoritmo, así como la localización del conjunto de datos a analizar. Que a su vez es correctamente leído sin importar si su formato es cualquiera de los especificados en la sección de requerimientos. Y a partir de los conjuntos de datos leídos, se generan correctamente tantas particiones difusas como indique el usuario en el fichero de parámetros.

- El sistema genera mensajes informativos a lo largo del programa, y al final del mismo genera diferentes ficheros de salida clasificados según la función que se realice. En caso de error, éste es debidamente informado al usuario.

- Se han realizado diferentes experimentaciones, ejecutando los distintos algoritmos propuestos sobre bases de datos públicas. Expuestos los resultados, se ha realizado un análisis de los datos en relación con las principales medidas de calidad obtenidas, para intentar encontrar qué algoritmo es más adecuado para cual medida y ajustar los parámetros en la búsqueda de mejores resultados.

Con estos puntos podemos decir que se han cumplidos todos los objetivos definidos al inicio este proyecto.

Como conclusión personal diría que este proyecto me ha aportado una gran experiencia personal en la realización de un proyecto de tal envergadura, acostumbrado hasta ahora a los pequeños programas realizados en el Grado, y al reto que ha supuesto como tal. También este proyecto me ha servido para el afianzamiento de los conocimientos obtenidos en el Grado.

5.3.2. Trabajo Futuro

Las líneas de trabajo futuro que deja este proyecto abiertas son:

- Mayor optimización de los algoritmos existentes.
- Preparación del paquete para alojarlo en CRAN (The Comprehensive R Archive Network), el lugar donde se almacenan los paquetes que instalamos desde R. Actualmente el paquete solo está disponible de manera local o desde GitHub (dependiendo de un paquete para instalarlo desde esta plataforma) y para alojarlo en CRAN debe de seguir de forma estricta las características que se indican en [35] ya que son muy exigentes en cuanto a su cumplimiento.

,

Capítulo 6. Bibliografía

[1] <<¿Qué pasa en un minuto en Internet en 2021?>> 16/11/2021. [En línea]. Available: https://www.telefonica.com/es/sala-comunicacion/que-pasa-en-un-minuto-en-internet-en-2021/

[2] U. Fayyad, G. PIstetsky-Shapiro y P. Smyth, «From Data Mining to Knowledge Discovery in Databases,» 1996.

[3] WEKA <<¿Qué es Weka?>> 21/10/2020. [En línea]. Available: https://medium.com/@isaiaranda15/que-es-weka-926c05050d44

[4] KEEL <<Description>>. [En línea]. Available: https://sci2s.ugr.es/keel/description.php

[5] KNIME <<What is KNIME?>> 21/04/2022. [En línea]. Available: https://www.phdata.io/blog/getting-started-with-knime/

[6] VIKANIME. [En línea]. Available: http://www.vikamine.org/

[7] R <<What is R: Overview, its Applications and what is R used for?>> 29/10/2021. [En línea]. Available: https://www.simplilearn.com/what-is-r-article

[8] Ciencia de datos <<¿Qué es Data Science?>>. [En línea]. Available: https://www.tibco.com/es/reference-center/what-is-data-science

[9] CRAN <<Getting your R package on CRAN>>. [En línea]. Available: https://kbroman.org/pkg_primer/pages/cran.html

[10] M. del Jesus, P. González, F. Herrera y M. Mesonero, «Evolutionary fuzzy rule induction process for subgroup discovery: a case study in marketing,» IEEE Trans Fuzzy Syst., 2007.

[11] F. Berlanga, M. J. Del Jesus, P. González, F. Herrera y M. Mesonero, «Multiobjective Evolutionary Induction of Subgroup Discovery Fuzzy Rules: A Case Study in Marketing,» 2006.

[12] C. J. Carmona, P. González, M. J. del Jesús y F. Herrera, NMEEF-SD: Non-dominated Multi-objective Evolutionary algorithm for Extracting Fuzzy rules in Subgroup Discovery, 2010.

[13] EuropaPress, «Expertos en análisis de datos aseguran que la demanda de científicos en este área "será ingente en el futuro",» 25 08 2014. [En línea]. Available: http://www.europapress.es/andalucia/noticia-expertos-analisis-datos-asegurandemanda-cientificos-area-sera-ingente-futuro-20140825184716.html

[14] Backpropagation <<Understanding Backpropagation Algorithm>> 08/08/2019. [En línea]. Available: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

[15] J. Quinlan, C4.5: programs for machine learning, 2014.

[16] J. Hartigan y M. Wong, «Algorithm AS 136: A k-means clustering algorithm,» Applied statistics, 1979.

[17] R. Agrawal, T. Imieliski y A. Swami, «Mining association rules between sets of items in large databases,» Proceedings of the 1993 ACM SIGMOD international conference on management of data, pp. 207-216, 1993.

[18] Quinlan JR (1987) Generating Production Rules from Decision Trees. In: Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI'87). Milan, Italy, pp 304-307

[19] Lavrac N, Kavsec B, Flach P, Todorovski L (2004) Subgroup discovery with CN2-SD. Journal of Machine Learning Research 5: 153-188

[20] F. Herrera, M. J. del Jesus, P. González y C. J. Carmona, «An overview on subgroup discovery: foundations and applications,» 2010.

[21] Klösgen W (1996) Explora: A Multipattern and Multistrategy Discovery Assistant. In Fayyad U, Piatetsky-Shapiro G, Smyth P, Uthurusamy R (eds) Advances in Knowledge Discovery and Data Mining, AAAI Press, California, pp 249-271

[22] Lavrac N, Flach P, Zupan B (1999) Rule evaluation measures: A unifying view. In: Proceedings of the 9th International Workshop on Inductive Logic Programming (ILP'99). Bled, Slovenia, pp 174-185

[23] Cordón O, del Jesus MJ, Herrera F (1998) Genetic Learning of Fuzzy Rule-based Classification Systems Co-operating with Fuzzy Reasoning Methods. International Journal of Intelligent Systems 13 (10/11): 1025-1053

[24] Kloesgen W, May M (2002) Census data mining—an application. In: Proceedings of the 6th European conference on principles of data mining and knowledge discovery, pp 65-79

[25] Del Jesus MJ, González P, Herrera F, Mesonero M (2005) Evolutionary Induction of Descriptive Rules in a Market Problem. In Ruan D, Chen G, Kerre E, Wets G (eds) Intelligent Data Mining: Techniques and Applications. Springer Verlag, pp 267-292

[26] P. Clark y T. Niblett, «The CN2 induction algorithm,» MAchine Learning, vol. 3, nº 4, pp. 261-283, 1989.

[27] Bäck T, Fogel D, Michalewicz Z (1997) Handbook of Evolutionary Computation. Oxford University Press, Oxford

[28] V. Jovanoski y N. Lavrac, «Classification Rule Learning with APRIORI-C,» de Progress in Artificial Intelligence, Springer, 2002, pp. 44-51.

[29] Agrawal R, Mannila H, Srikant R, Toivonen H, Verkamo AI (1996) Fast discovery of association rules. In: Fayyad U, Piatetsky-Shapiro G, Smyth P, Uthurusamy R (eds) Advances in knowledge discovery and data mining. AAAI Press, Cambridge, pp 307-328

[30] J. Han, H. Pei y Y. Yin, «Mining Frequent Patterns without Candidate Generation,» de Proc. Conf. on the Management of Data, Dallas, TX, 2000.

[31] E. Zitzler, M. Laumanns y L. Thiele, SPEA2: Improving the Strength Pareto Evolutionary Algorithm, 2001.

[32] K. Deb, S. Agrawal, A. Pratap y T. Mayerivan, A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II, 2000.

[33] Wrobel S (1997) An algorithm for multi-relational discovery of subgroups. In: Proceedings of the 1st European symposium on principles of data mining and knowledge discovery, vol 1263. Springer, LNAI, pp 78-87

[34] C. Carmona, V. Ruíz-Rodado, A. Weber, M. Grootyeld, P. González y D. Elizondo, «A fuzzy genetic programming-based algorithm for subgroup discovery and the application to one problem of pathogenesis of acute sore throat conditions in humans,» Information Sciences, vol. 298, pp. 180-197, 20 3 2015.

[35] The R Core Team, Writing R Extensions, 2014 .

[36] Gamberger D, Lavrac N (2002) Expert-guided subgroup discovery: Methodology and application. Journal of Artificial Intelligence Research 17: 1–27

[37] Zadeh LA (1975) The concept of a linguistic variable and its applications to approximate reasoning, parts I, II, III. Information Sciences 8-9: 199–249, 301–357, 43–80

[38] Zadeh LA (1965) Fuzzy sets. Information Control 8: 338–353

[39] Klösgen W (2002) Subgroup Discovery. In Klösgen W, Zytkow J (eds) Handbook of Data Mining and Knowledge Discovery. Oxford University Press, New York, pp 354–364

[40] Gamberger, D, Lavrac, N. Generating actionable knowledge by expert-guided subgroup discovery. In Proc. the 6th European Conference on Principles of Data Mining and Knowledge Discovery, Aug. 2002, pp.163-175

[41] Piatetsky-Shapiro G, Matheus, C (1994) The interestingness of deviation. In: Proceedings of the AAAI-94 Workshop on Knowledge Discovery in Databases. Seattle, Washington, pp 25–36

[42] María J. del Jesus, P. González, and F. Herrera (2008). <<Subgroup Discovery with Linguistic Rules>>

[43] Boley M, Grosskreutz H (2009) Non-redundant subgroup discovery using a closure system. In: Proceedings of the European conference on machine learning and principles and practice of knowledge discovery in databases, vol 5781. Springer, LNAI, pp 179-194

[44] Grosskreutz H, Rüping S (2009) On subgroup discovery in numerical domains. Data Mining Knowl Discov 19(2):210-216

[45] Grosskreutz H, Rüping S, Wrobel S (2008) Tight optimistic estimates for fast subgroup discovery. In: European conference on machine learning and principles and practice of knowledge discovery in databases, pp 440-456

[46] Atzmueller M, Puppe F (2006) SD-Map—a fast algorithm for exhaustive subgroup discovery. In: Proceedings of the 17th European conference on machine learning and 10th European conference on principles and practice of knowledge discovery in databases, vol 4213. Springer, LNCS, pp 6-17

[47] Mueller M, Rosales R, Steck H, Krishnan S, Rao B, Kramer S (2009) Subgroup discovery for test selection: a novel approach and its application to breast cancer diagnosis. In: Proceedings of the 8th international symposium on intelligent data analysis, vol 5772. Springer, LNCS, pp 119-130

[48] Siebes A (1995) Data Surveying: foundations of an inductive query language. In: Proceedings of the 1st international conference on knowledge discovery and data mining. AAAI Press, pp 269-274

[49] Wrobel S (2001) Inductive logic programming for knowledge discovery in databases. Springer, chap Relational Data Mining, pp 74-101

[50] Leeuwen, M, Knobbe, A. Diverse subgroup set discovery. Data Mining and Knowledge Discovery, 2012, 25(2): 208-242.

[51] Lavrac N, Cestnik B, Gamberger D, Flach PA (2004) Decision support through subgroup discovery: three case studies and the lessons learned. Mach Learn 57(1-2):115-143

[52] Lavrac N, Zelezny F, Flach PA (2003) RSD: relational subgroup discovery through first-order feature construction. In: Proceedings of the 12th international conference inductive logic programming, vol 2583. Springer, LNCS, pp 149-165

[53] Zelezny F, Lavrac N (2006) Propositionalization-based relational subgroup discovery with RSD. Machine Learning 62:33-63

[54] Kavsek B, Lavrac N, Jovanoski V (2003) APRIORI-SD: adapting association rule learning to subgroup discovery. In: Proceedings of the 5th international symposium on intelligent data analysis, vol 2810. Springer, LNCS, pp 230-241

[55] C.J. Carmona, P. González, M.J. del Jesus (2015) FuGePSD: Fuzzy Genetic Programming-based algorithm for Subgroup Discovery. Published by Atlantis Press.

[56] C.J. Carmona, V. Ruiz-Rodado, M.J. del Jesus, A. Weber, M. Grootveld, P. González, D. Elizondo (2014) A fuzzy genetic programming-based algorithm for subgroup discovery and the application to one problem of pathogenesis of acute sore throat conditions in humans, Information Sciences 298 (2015) 180-197.

[57] Helal S. (2016) Subgroup discovery algorithms: A survey and empirical evaluation. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 31(3): 561–576

[58] C.J. Carmona, F. Herrera, M.J. del Jesus (2018) A unifying analysis for the supervised descriptive rule discovery via the weighted relative accuracy. Knowledge-Based Systems 139, 89-100

Anexo I Manual de instalación

Este primer anexo se dedicará a mostrar la forma de instalar las herramientas necesarias para ejecutar este proyecto, tanto en un sistema operativo Windows como en un sistema operativo Linux, particularmente, en la distribución Ubuntu. Debe destacarse que estos pasos pueden variar dependiendo de la versión utilizada de R, pero incluso en este caso no debería haber demasiada diferencia.

Anexo I.1. Instalación de R

Comenzaremos con las instrucciones necesarias para instalar el software R en nuestro sistema, primero desde el sistema operativo Windows y después desde Ubuntu.

Anexo I.1.1. Windows

Para empezar, mencionaré que la versión de Windows utilizada en la realización del proyecto fue Windows 10, por lo tanto los pasos que serán mostrados seguirán este punto de vista, sin embargo la instalación es similar para versiones anteriores de Windows, variando únicamente el estilo de las ventanas de las imágenes de ejemplo.

Anexo I.1.1.1. Requisitos previos

En la instalación del software R sobre el sistema Windows no se requiere ningún requisito previo. Sin embargo, se recomienda tener activada la conexión a Internet para, de haberla, se pueda descargar la última versión de este software.

Anexo I.1.1.2. Instalación

Suponiendo que la conexión a Internet está disponible, podemos encontrar la última versión de este software en el siguiente enlace: http://cran.r-project.org/bin/windows/base/. Para descargar la última versión, simplemente tenemos que pinchar en el enlace "Download R 4.2. for Windows". Esta versión 4.2 es actualmente la versión más reciente de R, aunque esta irá cambiando a lo largo del tiempo, el procedimiento es el mismo. En nuestro caso usaremos la versión 4.0.0, que era la última versión en el momento de inicio de este proyecto.

Una vez tenga el fichero de instalación, lo abriremos. En primer lugar, aparecerá la pantalla de selección de idioma. En nuestro caso, seleccionaremos el lenguaje

Español y pulsaremos Aceptar. La siguiente pantalla en aparecer nos muestra el acuerdo de licencia de uso de R, que deberíamos leer a pesar de que usualmente se suele omitir dicha lectura. Una vez leído, y suponiendo que estemos conformes con este acuerdo de licencia, pulsamos en el botón "**Siguiente**".

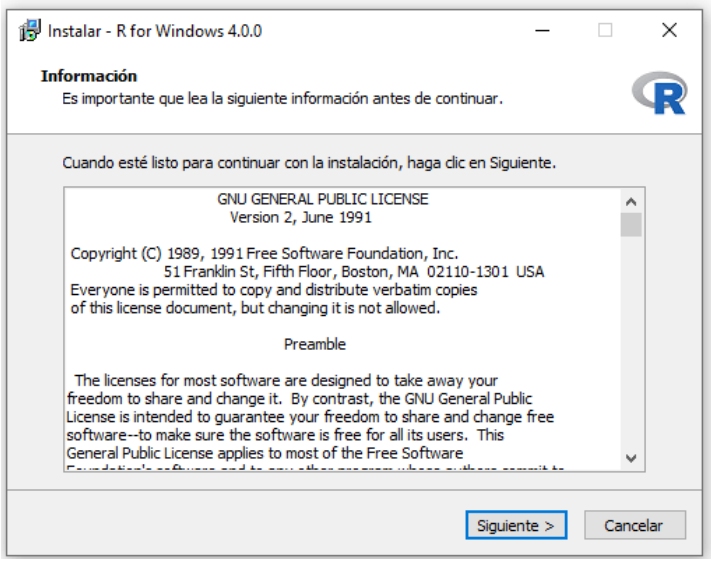


Figura 28. Acuerdo de licencia de uso de R

La siguiente es la pantalla para seleccionar la ruta de instalación, donde se nos pregunta en qué carpeta deseamos que se instale el software R en nuestro ordenador. La carpeta de instalación por defecto será "C:\Program Files\R\R-4.0.0\" o similar, pero en caso de que queramos escoger una carpeta diferente a la escogida por defecto, pulsaremos en el botón "Examinar…" donde seleccionaremos un nuevo directorio de instalación. Una vez seleccionada nuestra carpeta de instalación, pulsaremos el botón "**Siguiente**".

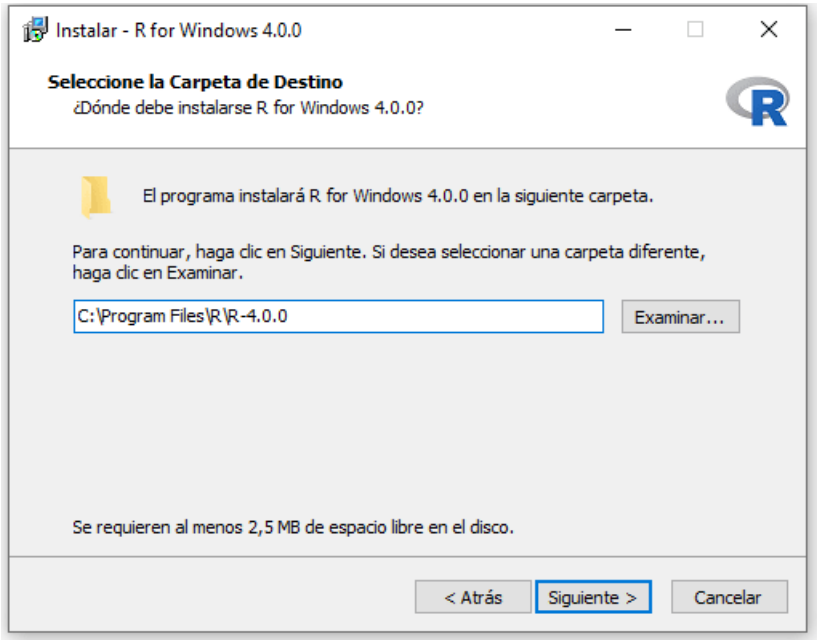


Figura 29. Selección de carpeta de destino de R

En la siguiente pantalla deberemos seleccionar los componentes que deberán instalarse. En nuestro caso, instalaremos los componentes seleccionados por defecto, por lo que no tocaremos nada y haremos click en el botón "**Siguiente**".

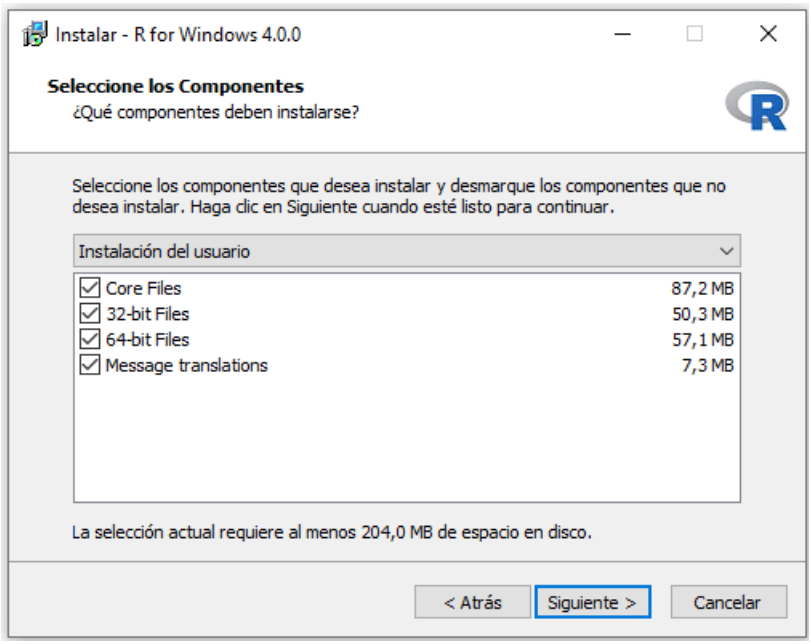


Figura 30. Instalación de componentes adicionales

A continuación, la siguiente pantalla nos pregunta si deseamos utilizar las opciones de configuración adicionales de R. En el caso de nuestra librería no necesitamos el uso de las opciones de configuración, por lo que marcamos en "**no**" y haremos click en "**Siguiente**".

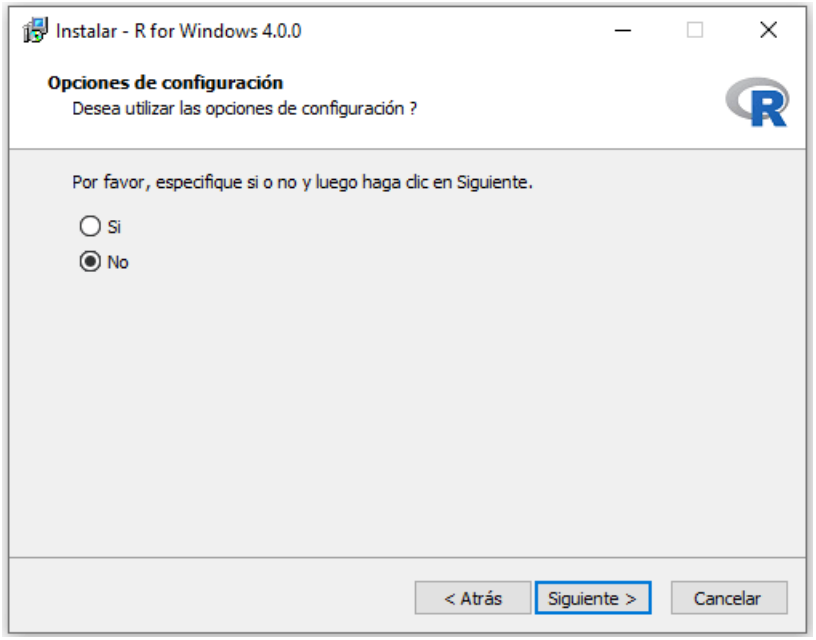


Figura 31. Opciones de configuración de R

En la siguiente pantalla se nos pregunta la ubicación de los accesos directos del menú de inicio. En nuestro caso, dejaremos la configuración por defecto, pero pueden cambiarlo a su gusto. Una vez conformes con la configuración deseada, pulsaremos el botón "**Siguiente**".

Finalmente aparecerá una pantalla en la que se nos presentarán cuatro casillas para marcar. Las dos primera se refieren al caso en que queramos tener accesos directos adiciones en el Escritorio o en el Menú Inicio. Las dos últimas casillas se refieren al manejo del registro de Windows. Cuando hayamos decidido las tareas adicionales que deben realizarse, pulsaremos el botón "**Siguiente**" para comenzar el proceso de instalación.

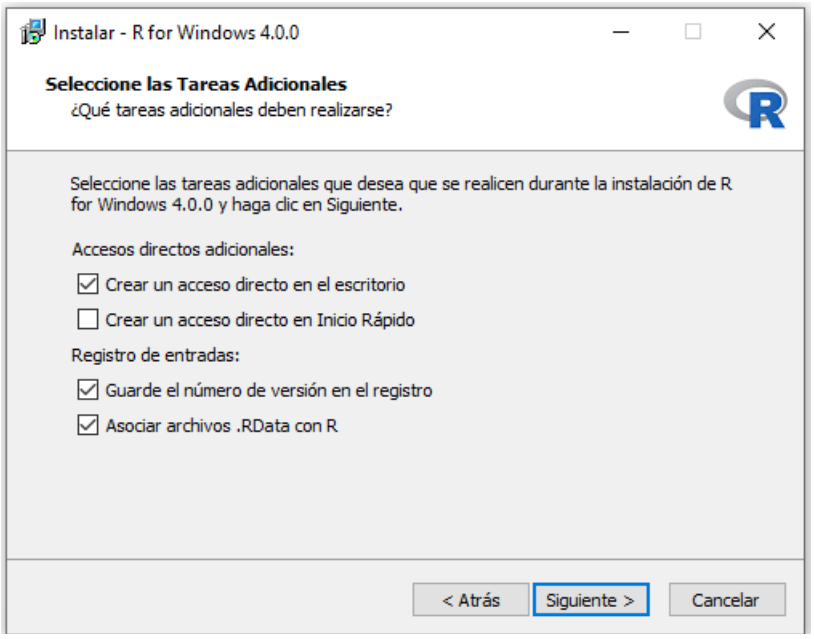


Figura 32. Selección de carpeta del Menú de Inicio

Terminado el proceso, pulsaremos el botón "**Finalizar**" para acabar la instalación de R en nuestro sistema.

Anexo I.1.2. Ubuntu

En este apartado, como se prometió anteriormente se muestra el proceso de instalación del software R en un sistema Linux. En nuestro caso, utilizaremos el sistema Ubuntu 22.04 LTS (Jammy Jellyfish). Para empezar, la instalación de R se realizará mediante la línea de comandos, para ello necesitaremos permisos de administrador.

Anexo I.1.2.1. Requisitos Previos

Los requisitos necesarios para la instalación del software R en Ubuntu son los dos siguientes: la disponibilidad de conexión a Internet, así como una cuenta de administrador.

Anexo I.1.2.2. Instalación

Como primer paso de la instalación del software R en Ubuntu, se deberá añadir una entrada en el fichero "*/etc/apt/sources.list*". Este paso puede ser realizado tanto desde la shell como desde una aplicación gráfica, como mostraremos a continuación.

Primero realizaremos este paso desde la aplicación gráfica, donde deberemos abrir la aplicación "**Software y actualizaciones**". Una vez abierta, iremos a la pestaña "**Otro software**", y buscaremos en la esquina inferior izquierda el botón "**Añadir...**" en el que pulsaremos. A continuación, aparecerá la siguiente pantalla:

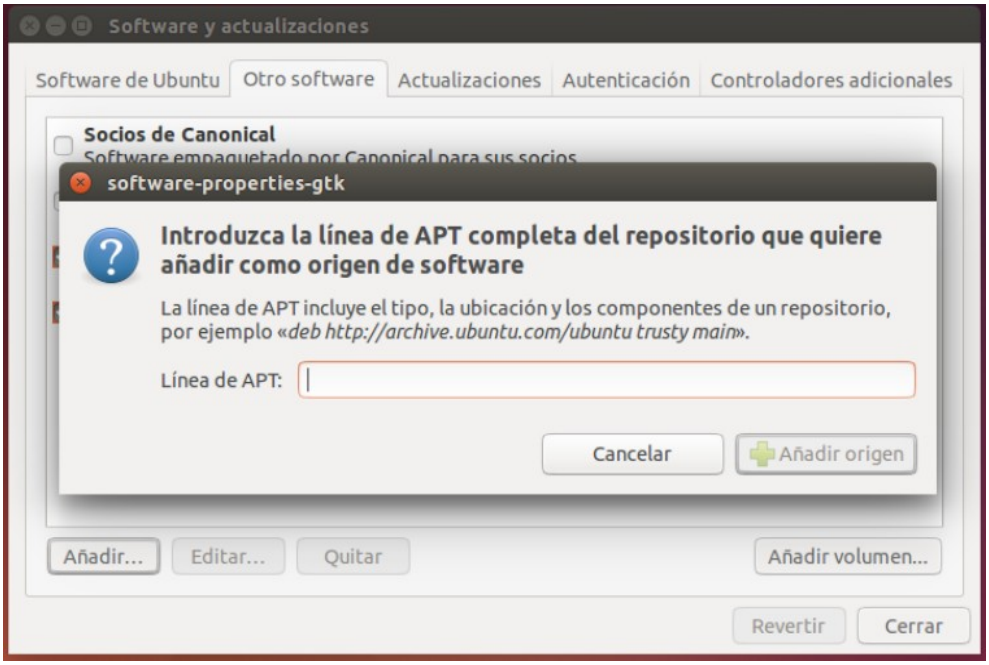


Figura 33. Adición de línea de APT para instalación de R

Como podemos ver en esta ventana, donde dice Línea de APT, debemos añadir la línea correspondiente a nuestro sistema operativo Ubuntu:

- Ubuntu 22.04 - Jammy Jellyfish (amd64):
  deb https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/
- Ubuntu 21.10 - Impish Indri (amd64):
  deb https://cloud.r-project.org/bin/linux/ubuntu impish-cran40/
- Ubuntu 20.04 - Focal Fossa (LTS y amd64):
  deb https://cloud.r-project.org/bin/linux/ubuntu focal-cran40/
- Ubuntu 18.04 - Bionic Beaver (LTS):
  deb https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/
- Ubuntu 16.04 - Xenial Xerus (LTS):
  deb https://cloud.r-project.org/bin/linux/ubuntu xenial-cran40/

Una vez añadida, pulsamos en "**Añadir origen**" y "**Cerrar**".

También puede realizarse este paso mediante la línea de comandos:

sudo sh -c 'echo "deb https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/" >> /etc/apt/sources.list'

O también podemos modificar el fichero "*sources.list*" manualmente, tal como en la versión gráfica, debemos de añadir la línea de APT correspondiente en el fichero "sources.list" y guardar el archivo.

Añadida la línea de APT, podemos dirigirnos a la consola de comandos para introducir los siguientes comandos:

sudo apt-get update

sudo apt-get install r-base

sudo apt-get install r-base-dev

Donde "sudo" es el comando que indica que se utilizarán permisos root, por ello se indicó anteriormente que era necesario utilizar una cuenta de administrador.

Una vez introducidos los comandos y finalicen los procesos de estos comandos, R estará instalado en nuestro sistema. El software R funciona desde la línea de comandos. Para ello, tal como se muestra en la Figura 34, abriremos la consola y escribimos "**R**" (por supuesto, sin comillas) para iniciar el programa, en donde podremos introducir comandos de R.

Para poder utilizar el software R en una aplicación gráfica es necesario instalar un software adicional llamado RStudio, el cual veremos a continuación.

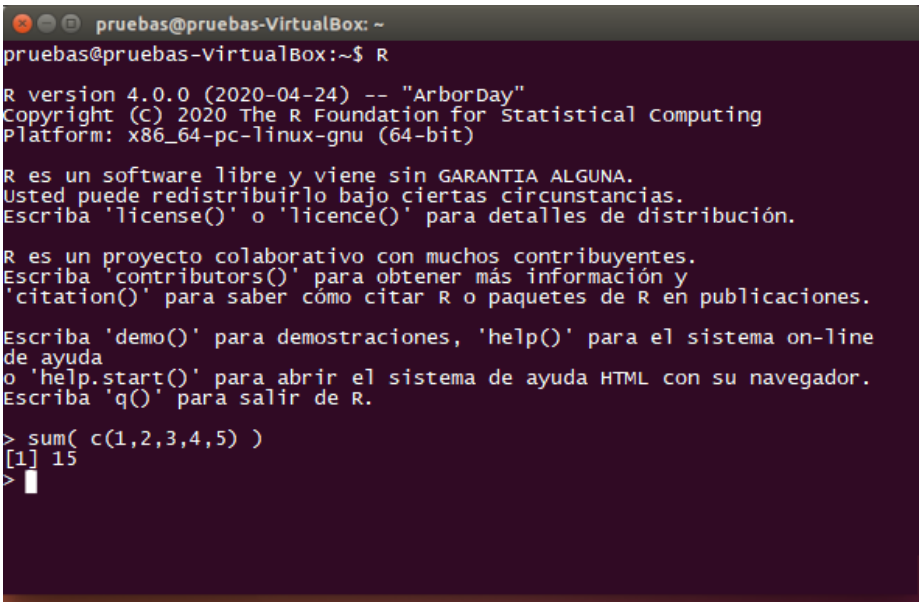


Figura 34: Consola de R en línea de comando de Ubuntu

Anexo I.2. Instalación de RStudio

El software RStudio anteriormente mencionado funciona como un entorno de desarrollo. Si bien no es necesaria la instalación de este software para la utilización de nuestra librería, se recomienda la instalación de este para una mayor facilidad de uso así como un mayor control de nuestras sesiones en R.

Anexo I.2.1. Requisitos Previos

Para el correcto funcionamiento del entorno de desarrollo RStudio, se requiere la previa instalación del software R en nuestro sistema operativo. Además, la versión instalada deberá ser igual o posterior a la versión 3.3.0, en caso contrario habrá que actualizar la versión del software R.

Anexo I.2.1.1. Windows

Del mismo modo que para el software R, mostraremos los pasos a seguir para la instalación de RStudio tanto para el sistema Windows como para Linux. En este apartado se mostrará los procedimientos para la instalación de RStudio en Windows.

Como con cualquier otro software partiremos de la obtención del archivo de instalación. Con respecto a este archivo de instalación, se puede descargar la última versión del software RStudio desde el siguiente enlace: http://www.rstudio.com/products/rstudio/download/. A continuación buscaremos el archivo correspondiente a nuestro sistema operativo para descargarlo. Una vez descargado el fichero, vamos a la carpeta de descarga y abrimos este fichero para comenzar el proceso de instalación.

La instalación comienza dando la bienvenida a la instalación de RStudio, por lo que empezaremos pulsando en "**Siguiente**" para comenzar. La siguiente pantalla en aparecer está destinada a preguntar la ruta de instalación deseada en nuestro ordenador. La carpeta de instalación por defecto será "C:\Program Files\RStudio\" o similar a esta, pero en caso de que queramos escoger una carpeta diferente a la escogida por defecto, pulsaremos en el botón "**Examinar…**" donde seleccionaremos un nuevo directorio de instalación. Una vez seleccionada nuestra carpeta de instalación, pulsaremos el botón "**Siguiente**".

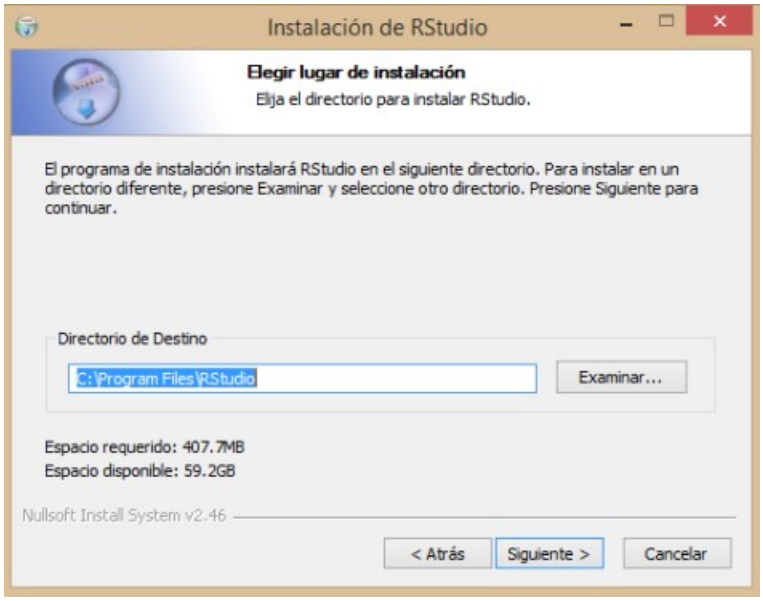


Figura 35. Selección de carpeta de destino de RStudio

La siguiente y última pantalla pregunta el nombre que se asignará a la carpeta de accesos directos en el Menú de Inicio. También es posible no crear accesos directos. Recomendamos dejar el nombre por defecto, y finalmente pulsar el botón "**Instalar**" para dar comienzo a la instalación. Una vez terminado el proceso pulsaremos el botón "Finalizar" para salir y comenzar a ejecutar RStudio.

Anexo I.2.1.2. Ubuntu

Para instalar RStudio en un sistema Linux Ubuntu, primero necesitamos obtener el fichero de instalación. Para ello, vamos a la pagina de descarga de RStudio en el enlace http://www.rstudio.com/products/rstudio/download/ y descargamos la última versión de RStudio que corresponda al sistema operativo Ubuntu 22.

A continuación, vamos a la carpeta de descargas y abrimos el archivo que acabamos de descargar. Al abrirlo, aparecerá en pantalla una ventana del Centro de software de Ubuntu, en la que se nos pide que confirmemos que queremos instalar el software RStudio, por tanto debemos hacer click en el botón "**Instalar**". Puesto que se requieren permisos de administrador, nos preguntará la contraseña de root.

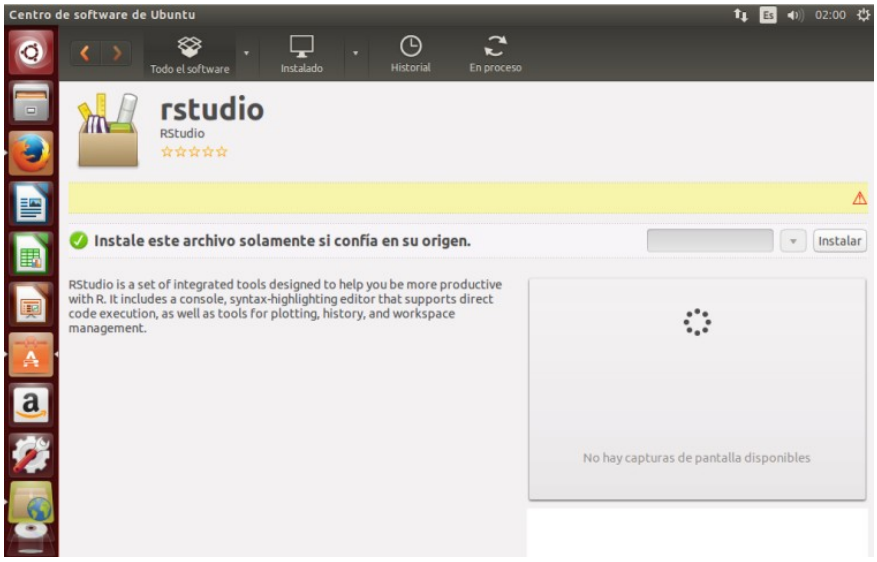


Figura 36. Instalación de RStudio desde el Centro de Software de Ubuntu

Anexo I.3. Instalación del paquete "SDRCPP"

Adjunto a esta memoria, se entregará una carpeta con el código fuente, así como un archivo llamado "**SDRCPP_1.0.tar.gz**". Este archivo es un paquete comprimido, listo para instalar nuestra librería en R.

Para instalar nuestro paquete, debemos abrir R (o RStudio). Una vez abierto, nos situaremos en el directorio en que se encuentre el fichero del paquete usando el comando "**setwd(ruta)**" donde el parámetro ruta es la ruta absoluta o relativa en que se encuentra el fichero. Una vez situados en la carpeta correcta, instalaremos el paquete en R con el comando "**install.packages("SDRCPP_1.0.tar.gz", repos = NULL, type = "source")**" tal como hacemos en el ejemplo de la Figura 37:

```
> setwd("~/Universidad/TFG/")
> install.packages("SDRCPP_1.0.tar.gz", repos = NULL, type = "source")
Installing package into 'E:/Documents/R/win-library/3.2'
(as 'lib' is unspecified)
* installing *source* package 'SDRCPP' ...
** R
** data
*** moving datasets to lazyload DB
** prepating package for lazy loading
Creating a generic function for 'print' from package 'base' in package 'SDRCPP'
Creating a generic function for 'summary' from package 'base' in package 'SDRCPP'
Creating a generic function for 'plot' from package 'graphics' in package 'SDRCPP'
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded
*** arch - i386
*** arch - x64
* DONE (SDRCPP)
>
```

Figura 37: Instalación del paquete SDRCPP en R

Anexo II Manual de Usuario

En este anexo introducimos el manual de usuario para un uso adecuado de nuestro paquete.

Anexo II.1. Manual de Uso del paquete

Para poder ejecutar nuestro paquete y los algoritmos que este implementa deberemos enlazar el paquete en el llamado "*search path*" de R: Esta es la forma de uso más común, para ello primero necesitamos instalar el paquete tal como indicamos en el anterior anexo. Si el paquete ya está instalado, entonces lo enlazamos utilizando la función library(SDRCPP).

La ejecución del programa dependerá de un fichero de parámetros que se llamará "**paramFile.txt**" (debe llamarse así) que contendrá la información que indicará al programa qué algoritmo ejecutar, las direcciones de los conjuntos de datos, así como los distintos parámetros necesarios para la correcta ejecución de los algoritmos. El formato y los distintos parámetros los especificaremos más adelante.

Antes de iniciar el paquete, deberemos asegurarnos de estar situados en la carpeta de trabajo donde se encuentre el fichero de parámetros "paramFile.txt" utilizando la función setwd(ruta), indicando como parámetro de la función la ruta. Además, en la carpeta de trabajo debe existir la carpeta "/output/" en la que se generarán los ficheros de salida con los resultados. El programa se inicia con la función llamada SDRCPP().

```
>
>
>
> library(SDRCPP)
> setwd("C:/Users/rcres/Documents/Universidad/TFG")
> SDRCPP()
```

Figura 38: Ejemplo de ejecución del paquete RCPP

Una vez ejecutado el algoritmo, se irá mostrando por pantalla los detalles de la ejecución del algoritmo escogido. Una vez finalizada la ejecución, los resultados de la ejecución se habrán almacenado en diversos ficheros clasificados según su función, que son: Información de la ejecución, fichero de reglas generadas y fichero de medidas de calidad.

Un punto a mencionar es que el programa da por supuesto que en el conjunto de datos tenemos a nuestra variable objetivo como la última variable del conjunto de datos.

```
1  ------------------------------------------
2  |             Parameters Echo            |
3  ------------------------------------------
4  Algorithm name:                                  MESDIF
5  Input file name training:                        data/iris/iris-5-1tra.dat
6  Input file name test:                            data/iris/iris-5-1tst.dat
7  Tracking file name:                              output/iris_info.txt
8  Rules file name:                                 output/iris_rules.txt
9  Quality measures file name:                      output/iris_qmeasure.txt
10 Random generator seed:                           1000
11 Selected class of the target variable:           NULL
12 Number of labels for the continuous variables:   3
13 Number of evaluations:                           10000
14 Number of individuals in the Population:         100
15 Cross probability:                               0.600000
16 Mutation probability:                            0.100000
17 Minimum confidence threshold:                    0.600000
18 Representation of the Rules:                     CAN
19 Objective 1:                                     CSUP
20 Objective 2:                                     CCNF
21 Objective 3:                                     NULL
22 Objective 4:                                     NULL
23 Number of individuals in the Elite:              3
24 Echo?                                            true
25
26 ------------------------------------------
27 |              Dataset Echo              |
28 ------------------------------------------
29 Number of examples in Train Dataset: 600
30 Number of examples in Test Dataset: 150
31 Number of variables: 5
32
33 ------------------------------------------
34 |       Computation of the info gain     |
35 ------------------------------------------
36 Points for computation of the info gain:
37     Variable SepalLength: 5.200000  7.000000  8.800000
38     Variable SepalWidth: 2.600000  3.800000  5.000000
39     Variable PetalLength: 2.475000  5.425000  8.375000
40     Variable PetalWidth: 0.700000  1.900000  3.100000
41 Information Gain of the variables:
42     Variable SepalLength: 0.543243
43     Variable SepalWidth: 0.266632
44     Variable PetalLength: 0.875922
```

Figura 39: Fichero de salida con información de la ejecución

La información contenida en cada uno de estos ficheros es la siguiente:

- Información de la ejecución: Este fichero comienza listando los parámetros utilizados para la ejecución, así como información de los conjuntos de datos y así como diversa información de los procesos realizados durante la ejecución. Mostrado en la Figura 39.

- Fichero de reglas generadas: En este fichero se almacenan las reglas generadas durante la ejecución. En caso de que no halla asignada una clase objetivo o ésta tenga el valor "NULL", se generarán las reglas para todas las clases de la variable objetivo por orden. Mostrado en la Figura 40.

- Fichero de medidas de calidad: En este fichero se almacenan las medidas de calidad resultadas por las reglas generadas por el conjunto de entrenamiento y aplicadas sobre el conjunto de prueba. Mostrado en la Figura 41.

```
GENERATED RULE 0
   Antecedent
      Variable PetalWidth = Label 1 (-1.100000 0.100000 1.300000)
   Consecuent: 0 - Iris-setosa
GENERATED RULE 2
   Antecedent
      Variable PetalLength = Label 1 (-1.950000 1.000000 3.950000)
   Consecuent: 0 - Iris-setosa
GENERATED RULE 0
   Antecedent
      Variable PetalLength = Label 1 (-1.950000 1.000000 3.950000)
      Variable PetalWidth = Label 1 (-1.100000 0.100000 1.300000)
   Consecuent: 0 - Iris-setosa


GENERATED RULE 0
   Antecedent
      Variable PetalLength = Label 2 (1.000000 3.950000 6.900000)
      Variable PetalWidth = Label 2 (0.100000 1.300000 2.500000)
   Consecuent: 1 - Iris-versicolor
GENERATED RULE 2
   Antecedent
      Variable PetalWidth = Label 2 (0.100000 1.300000 2.500000)
   Consecuent: 1 - Iris-versicolor
GENERATED RULE 0
   Antecedent
      Variable PetalWidth = Label 2 (0.100000 1.300000 2.500000)
   Consecuent: 1 - Iris-versicolor


GENERATED RULE 1
   Antecedent
      Variable SepalWidth = Label 2 (2.000000 3.200000 4.400000)
   Consecuent: 2 - Iris-virginica
GENERATED RULE 2
   Antecedent
      Variable SepalWidth = Label 2 (2.000000 3.200000 4.400000)
   Consecuent: 2 - Iris-virginica
GENERATED RULE 1
   Antecedent
      Variable SepalWidth = Label 2 (2.000000 3.200000 4.400000)
   Consecuent: 2 - Iris-virginica

```

Figura 40: Fichero de salida de reglas generadas

```
Rule        Class              NVar       Unusualness FConfidence     CConfidence
1           Iris-setosa        4          0.153333    0.878613    0.640000
2           Iris-setosa        3          0.004444    0.475410    0.317073
3           Iris-setosa        4          0.008889    0.328671    0.300000
4           Iris-versicolor    3          0.068889    0.323782    0.478873
5           Iris-versicolor    2          0.020000    0.628606    0.354610
6           Iris-virginica     3          0.133333    0.000000    0.000000
7           Iris-virginica     2          0.173333    0.889706    0.694444
8           Iris-virginica     3          0.146667    0.891791    0.764706
-           8                  3.000000   0.051944    0.552072    0.443713

True Positives: 246
False Negatives: 154
False Positives: 305
True Negatives: 495
True Positive Rate: 0.615000
False Positive Rate: 0.381250
```

Figura 41: Fichero de salida de medidas de calidad

Anexo II.2. Configuración del fichero de parámetros

En este apartado explicaremos como como configurar el fichero de parámetros necesario para ejecutar los algoritmo apropiadamente.

Lo primero es, antes de nada, será desplazarse al directorio en que se encuentran los datos en caso de que no se haya especificado la ruta completa de los ficheros de datos. Esto se realiza con la función setwd() indicando la ruta que queremos establecer, que será el lugar donde se colocará el fichero de parámetros. Ahora deberemos crear un documento de texto con el nombre "paramFile.txt". Una vez abierto, el fichero de parámetros tendrá el siguiente formato: 'Palabra_clave' = 'valor'. Donde cada palabra clave corresponderá a un parámetro. Las palabras clave permitidas y sus posibles valores son las siguientes:

- **Algorithm**: Acepta como valor las siguientes valores clave; "SDIGA", "MESDIF", "NMEEF_SD", o "FuGePSD". Cada una de estos valores clave correspondiente a uno de los algoritmos implementados. Obligatoria.

- **InputTrainData**: Es la ruta para el conjunto de datos de entrenamiento. Acepta una o más direcciones para el conjunto de datos separadas por comas. Obligatoria.

- **InputTestData**: Es la ruta para el conjunto de datos de prueba. Acepta una o más direcciones para el conjunto de datos separadas por comas.

- **Seed**: Es el parámetro de la semilla utilizada para generar números pseudo-aleatorios. Acepta un número entero. Obligatoria.

- **NLabels**: Es el parámetro para especificar el número de etiquetas en que se dividirán las particiones difusas. Acepta un número entero. Obligatoria.

- **NEval**: Es el parámetro para especificar el número de evaluaciones máximas para los algoritmos genéticos. Acepta un número entero. Obligatoria.

- **PopLength**: Es el parámetro para especificar el número de individuos que contendrá la población principal para los algoritmos genéticos. Acepta un número entero. Obligatoria.

- **TargetClass**: Es el parámetro para especificar el valor de la variable objetivo para buscar subgrupos. La variable objetivo es siempre la última variable. Acepta una cadena con el valor de una de las clases de la variable objetivo. Si no encuentra dicho valor o el valor es "NULL", el algoritmo es ejecutará

repetidamente para cada uno de los valores de la variable objetivo. Parámetro para los algoritmos: SDIGA, MESDIF, NMEEF-SD.

- **CrossProb**: Es el parámetro para especificar la probabilidad del cruce para el operador de cruce de los algoritmos genéticos. Acepta un número de coma flotante entre cero y uno [0-1]. Obligatoria.

- **MutProb**: Es el parámetro para especificar la probabilidad del mutación para el operador de mutación de los algoritmos genéticos. Acepta un número de coma flotante entre cero y uno [0-1]. Obligatoria.

- **InsertProb**: Es el parámetro para especificar la probabilidad de inserción de variable para el operador genético de inserción. Acepta un número de coma flotante entre cero y uno [0-1]. Obligatoria. Parámetro para el algoritmo FuGePSD.

- **DropProb**: Es el parámetro para especificar la probabilidad de eliminación de variable para el operador genético de eliminación. Acepta un número de coma flotante entre cero y uno [0-1]. Obligatoria. Parámetro para el algoritmo FuGePSD.

- **MinConf**: Es el parámetro utilizado para especificar la confianza mínima necesaria para aceptar las reglas de los algoritmos. Acepta un número de coma flotante entre cero y uno [0-1]. Obligatoria. Parámetro para los algoritmos: SDIGA, NMEEF-SD y FuGePSD.

- **RulesRep**: Es el parámetro utilizado para especificar el tipo de representación usaremos en las reglas. Acepta los siguiente dos valores: "CAN" o "DNF". Toma por defecto las reglas canónicas.

- **Obj1**: Es el parámetro utilizado para especificar el primer objetivo. Acepta los siguientes valores: "CSUP" (Soporte Nítido), "FSUP" (Soporte difuso), "CCNF" (Confianza Nítida), "FCNF" (Confianza Difusa), "UNUS" (Inusualidad), "COVE" (Cobertura), "SIGN" (Significancia) o "SENS" (Sensibilidad). No es sensible a mayúsculas. Obligatoria

- **Obj2**: Es el parámetro utilizado para especificar el segundo objetivo. Acepta los siguientes valores: "CSUP" (Soporte Nítido), "FSUP" (Soporte difuso), "CCNF" (Confianza Nítida), "FCNF" (Confianza Difusa), "UNUS" (Inusualidad), "COVE" (Cobertura), "SIGN" (Significancia) o "SENS" (Sensibilidad). No es sensible a mayúsculas. Obligatoria

- **Obj3**: Es el parámetro utilizado para especificar el tercer objetivo. Acepta los siguientes valores: "CSUP" (Soporte Nítido), "FSUP" (Soporte difuso), "CCNF" (Confianza Nítida), "FCNF" (Confianza Difusa), "UNUS" (Inusualidad), "COVE" (Cobertura), "SIGN" (Significancia), "SENS" (Sensibilidad) o "NULL" (Objetivo nulo). No es sensible a mayúsculas. Obligatoria

- **Obj4**: Obj3: Es el parámetro utilizado para especificar el cuarto objetivo. Acepta los siguientes valores: "CSUP" (Soporte Nítido), "FSUP" (Soporte difuso), "CCNF" (Confianza Nítida), "FCNF" (Confianza Difusa), "UNUS" (Inusualidad), "COVE" (Cobertura), "SIGN" (Significancia), "SENS" (Sensibilidad) o "NULL" (Objetivo nulo). No es sensible a mayúsculas. Parámetro para los algoritmos: MESDIF.

- **W1**: Es el parámetro utilizado para especificar el peso utilizado para ponderar el primer objetivo. Acepta un número de coma flotante entre cero y uno [0-1]. La suma de los tres pesos deben ser igual a uno. Obligatorio. Parámetro para los algoritmos: SDIGA y FuGePSD.

- **W2**: Es el parámetro utilizado para especificar el peso utilizado para ponderar el segundo objetivo. Acepta un número de coma flotante entre cero y uno [0-1]. La suma de los tres pesos deben ser igual a uno. Obligatorio. Parámetro para los algoritmos: SDIGA y FuGePSD.

- **W3**: Es el parámetro utilizado para especificar el peso utilizado para ponderar el tercer objetivo. Acepta un número de coma flotante entre cero y uno [0-1]. La suma de los tres pesos deben ser igual a uno. Obligatorio. Parámetro para los algoritmos: SDIGA y FuGePSD.

- **LSearch**: Es el parámetro utilizado para especificar el uso de la función de Optimización Local. Acepta uno de los siguientes dos valores: "true" o "false". No es sensible a mayúsculas. Parámetro para los algoritmos: SDIGA.

- **EliteLength**: Es el parámetro utilizado para especificar el número de individuos de la población de élite. Acepta un número entero. Debe ser menor a "PopLength". Obligatorio. Parámetro para los algoritmos: MESDIF.

- **ReInitCov**: Es el parámetro utilizado para especificar el uso de la Reinicialización por Cobertura cuando es necesario. Acepta uno de los siguientes dos valores: "true" o "false". No es sensible a mayúsculas. Parámetro para los algoritmos: NMEEF-SD.

- **Coverage**: Es el parámetro utilizado para especificar el porcentaje máximo de variables que participan en las reglas generadas en la reinicialización basada en cobertura. Acepta un número de coma flotante entre cero y uno [0-1]. Parámetro para los algoritmos: NMEEF-SD.

- **StrictDominance**: Es el parámetro utilizado para especificar si la comparación entre individuos debe ser hecha por Dominancia Estricta o no. Acepta uno de los siguientes dos valores: "yes" o "no". No es sensible a mayúsculas. Parámetro para los algoritmos: NMEEF-SD.

- **Diversity**: Es el parámetro utilizado para especificar la función de diversidad. Acepta uno de los siguientes valores: "CROWDING", "KNEE" o "UTILITY". No es sensible a mayúsculas. Valor por defecto: "Crowding". Parámetro para los algoritmos: NMEEF-SD.

- **TournamentSize**: Es el parámetro utilizado para especificar el número de competidores que participan en el operador genético de selección por torneo. Acepta un número entero inferior al tamaño del conjunto de datos. Por defecto es igual a 2. Parámetro para el algoritmo: FuGePSD.

- **T-norm**: Es el parámetro utilizado para especificar la t-norma usada para calcular el grado de compatibilidad de las reglas. Acepta los siguientes valores: "MINIMUM/MAXIMUM" o "PRODUCT/PROBABILISTIC_SUM". Por defecto toma el valor "PRODUCT/PROBABILISTIC_SUM". Parámetro para el algoritmo: FuGePSD.

- **InferenceType**: Es el parámetro utilizado para especificar el método de razonamiento difuso (Fuzzy Reasoning Method) que se utilizará. Acepta uno de los siguientes valores:

  - "NORMALIZED_SUM": Utiliza el método de razonamiento difuso de Suma Normalizada o Combinación Aditiva.

  - "ARITHMETIC_MEAN": Utiliza el método de razonamiento difuso de la suma aritmética.

  - "WINNING_RULE": Utiliza el método de razonamiento difuso de la regla ganadora. Valor por defecto.

  No es sensible a mayúsculas. Parámetro para el algoritmo: FuGePSD.

- **RuleWeight**: Es el parámetro utilizado para especificar el método que se utilizará para calcular el peso de las reglas. Acepta uno de los siguientes valores:
  - "CERTAINTY_FACTOR": Utiliza el método clásico de ponderación del factor de certeza.
  - "AVERAGE_PENALIZED_CERTAINTY_FACTOR": Utiliza la ponderación del Factor de Certeza Penalizado II de Ishibuchi.
  - "PENALIZED_CERTAINTY_FACTOR_IV": Utiliza la ponderación del Factor de Certeza Penalizado II de Ishibuchi. Valor por defecto.
  - "NO_WEIGHTS": No pondera el cálculo.

  No es sensible a mayúsculas. Parámetro para el algoritmo: FuGePSD.

- **ALL_CLASS**: Es el parámetro utilizado para especificar qué hacer si ninguna ninguna regla generada por el algoritmo pasa los filtros. Acepta los siguientes valores:
  - Si es TRUE, el algoritmo devuelve, al menos, la mejor regla para cada clase objetivo, incluso si ninguna ha pasado los filtros.
  - Si es FALSE, solo devuelve una regla, la mejor regla respecto a la confianza. Valor por defecto.

  No es sensible a mayúsculas. Parámetro para el algoritmo: FuGePSD.

- **Echo**: Es el parámetro utilizado para especificar si se debe mostrar mensajes informativos por pantalla durante la ejecución del algoritmo. Acepta uno de los siguientes dos valores: "yes" o "no". No es sensible a mayúsculas.

A continuación, en la siguiente figura se muestra unos ejemplos de cómo se verían los ficheros de parámetros en caso de querer ejecutar los algoritmos.

```
Algorithm = SDIGA
InputTrainData = data/german/german-5-1tra.dat,data/german/german-5-2tra.dat,data/german/german-5-3tra.dat,data/german/german-5-4tra.dat,data/german/german-5-5tra.dat
InputTestData = data/german/german-5-1tst.dat,data/german/german-5-2tst.dat,data/german/german-5-3tst.dat,data/german/german-5-4tst.dat,data/german/german-5-5tst.dat
Seed = 0
NLabels = 3
NEval = 10000
PopLength = 100
CrossProb = 0.6
MutProb = 0.1
MinConf = 0.6
RulesRep = CAN
Obj1 = csup
Obj2 = ccnf
Obj3 = null
W1 = 0.7
W2 = 0.3
W3 = 0.0
LSearch = false
Echo = yes
```

Figura 42: Fichero de parámetros para ejecutar SDIGA

```
Algorithm = MESDIF
InputTrainData = data/german/german-5-1tra.dat,data/german/german-5-2tra.dat,data/german/german-5-3tra.dat,data/german/german-5-4tra.dat,data/german/german-5-5tra.dat
InputTestData = data/german/german-5-1tst.dat,data/german/german-5-2tst.dat,data/german/german-5-3tst.dat,data/german/german-5-4tst.dat,data/german/german-5-5tst.dat
Seed = 1
NLabels = 3
NEval = 10000
PopLength = 100
CrossProb = 0.6
MutProb = 0.1
RulesRep = CAN
Obj1 = csup
Obj2 = ccnf
Obj3 = null
Obj4 = null
EliteLength = 3
Echo = yes
```

Figura 43: Fichero de parámetros para ejecutar MESDIF

```
Algorithm = NMEEF_SD
InputTrainData = data/german/german-5-1tra.dat,data/german/german-5-2tra.dat,data/german/german-5-3tra.dat,data/german/german-5-4tra.dat,data/german/german-5-5tra.dat
InputTestData = data/german/german-5-1tst.dat,data/german/german-5-2tst.dat,data/german/german-5-3tst.dat,data/german/german-5-4tst.dat,data/german/german-5-5tst.dat
Seed = 1000
NLabels = 3
TargetClass = 1
NEval = 1000
PopLength = 100
CrossProb = 0.6
MutProb = 0.1
MinConf = 0.6
RulesRep = CAN
Obj1 = csup
Obj2 = ccnf
Obj3 = null
W1 = 0.7
W2 = 0.3
W3 = 0.0
ReInitCov = yes
Coverage = 0.5
StrictDominance = yes
Diversity = Crowding
Echo = yes
```

Figura 44: Fichero de parámetros para ejecutar NMEEF-SD

```
Algorithm = FuGePSD
InputTrainData = data/ecoli/ecoli-5-1tra.dat,data/ecoli/ecoli-5-2tra.dat,data/ecoli/ecoli-5-3tra.dat,data/ecoli/ecoli-5-4tra.dat,data/ecoli/ecoli-5-5tra.dat
InputTestData = data/ecoli/ecoli-5-1tst.dat,data/ecoli/ecoli-5-2tst.dat,data/ecoli/ecoli-5-3tst.dat,data/ecoli/ecoli-5-4tst.dat,data/ecoli/ecoli-5-5tst.dat
Seed = 0
NLabels = 3
NEval = 1000
PopLength = 50
CrossProb = 0.6
MutProb = 0.1
InsertProb = 0.15
DropProb = 0.15
MinConf = 0.6
Obj1 = csup
Obj2 = ccnf
Obj3 = unus
Obj4 = null
W1 = 0.5
W2 = 0.3
W3 = 0.2
tournamentSize = 3
inferenceType = ARITHMETIC_MEAN
RuleWeight = AVERAGE_PENALIZED_CERTAINTY_FACTOR
t-norm = PRODUCT
ALL_CLASS = yes
```

Figura 45: Fichero de parámetros para ejecutar FuGePSD