

Análisis de la eficacia del tuning local en un modelo de clasificación basado en reglas difusas de asociación (FARC-HD)



Grado en Ingeniería Informática

Trabajo Fin de Grado

Alumno: Oier Etxeberria Urrestarazu

Director: José Antonio Sanz

Pamplona, fecha de defensa (~~I don't know~~
~~yet men~~)

upna

Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Agradecimientos

Patxi Mario ~~Rodolfo el tercero~~

Oier.

Resumen

Hoy en día el campo del *Machine Learning* o Aprendizaje Automático es un campo de investigación que está en auge. Dentro de este campo hay una gran variedad algoritmos y métodos capaces de extraer información. Dentro del subcampo de los algoritmos de clasificación nos encontramos con el FARC-HD.

FARC-HD es un algoritmo muy preciso y para ello en su etapa final aplica un algoritmo evolutivo para realizar un ajuste en las funciones de pertenencia y una selección de las mejores reglas. El ajuste de las funciones de pertenencia se hace de forma global para mantener la interpretabilidad del sistema y el objetivo del trabajo es analizar la eficacia de aplicar un ajuste local a expensas de perder la interpretabilidad del sistema.

Palabras clave

- Sistemas de clasificación
- Reglas de asociación difusas
- ~~FARC-HD~~
- Tuning lateral
- Tuning local

Summary

Key words

Laburpena

Hitz Klabeak

Índice de Contenido

1.- Introducción	10
2.- Preliminares.....	11
2.1.- Machine Learning.....	11
2.2.- Problemas de Clasificación	12
2.3.- Lógica Difusa	12
2.3.1- Operadores	13
2.3.2.- Implicación	15
2.3.3.- Inferencia	15
2.3.4.- Reglas Difusas.....	15
2.4.- Sistemas de Clasificación Basados en Reglas Difusas.....	16
2.4.1.- Método De Razonamiento Difuso	17
2.5.- Algoritmos Genéticos	18
2.5.1.- Codificación.....	19
2.5.2.- Cruce.....	20
2.5.3.- Mutación.....	20
2.5.4.- Selección	21
2.6.- FARC-HD	21
2.6.1.- Proceso de aprendizaje.....	22
2.6.2.- Proceso de Clasificación.....	22
2.6.3.- Algoritmo Genético en FARC-HD.....	23
3.- FARC-HD-LOCAL	26
3.1.- Tuning local en FARC-HD	26
3.1.1.- Bases Tuning local	26
3.1.2.- Codificación tuning local.....	27
3.2.- Mejorando efectividad del tuning local	27
4.- Marco Experimental.....	30
4.1.- Datasets	30
4.2.- Configuración de parámetros	30
4.3.- Medidas de Rendimiento	31
5.- Estudio Experimental	33
5.1.- Comparando FARC-HD global y FARC-HD local con las configuraciones originales.....	33
5.2.- Estudiando los modelos FARC-HD global y FARC-HD local incluyendo cambios en los parámetros de búsqueda	35
5.2.1.- FARC-HD con tuning Global	35

5.2.2.- FARC-HD con tuning Local.....	37
5.3.- Comparando los mejores resultados obtenidos con todos los demás	39
6.- Conclusiones.....	41
7.- Líneas Futuras.....	42
Bibliografía	43

Índice de Ilustraciones

Ilustración 1. Ejemplo lógica Clásica	12
Ilustración 2. Ejemplo Lógica difusa.....	13
Ilustración 3. Variable difusa con sus etiquetas lingüísticas	13
Ilustración 4. Esquema de un SCBRD	17
Ilustración 5. Esquema algoritmo evolutivo.....	19
Ilustración 6. Ejemplo cromosoma en codificación binaria.....	20
Ilustración 7. Cruzamiento sobre punto.....	20
Ilustración 8. Cruzamiento uniforme	20
Ilustración 9. Método de la ruleta	21
Ilustración 10. Etiquetas lingüísticas FARC-HD	22
Ilustración 11. Desplazamiento de una etiqueta lingüística	23
Ilustración 12. Desplazamiento lateral de una etiqueta	24
Ilustración 13. Ejemplo codificación genética tuning lateral y selección de reglas.....	24
Ilustración 14. Esquema CHC.....	25

Índice de Tablas

Tabla 1. Descripción de las características de los dataset utilizados	31
Tabla 2. Configuración de Parámetros.....	32
Tabla 3. FARC-HD global vs FARC-HD local.....	34
Tabla 4. Resultados de todas las configuraciones de FARC-HD con tuning Global	36
Tabla 5. Resultados de todas las configuraciones de FARC-HD con tuning Local.....	38
Tabla 6. Comparacion wilcoxon entre todas las configuraciones utilizadas	39
Tabla 7. Todos los resultados obtenidos	40

1.- Introducción

La inteligencia artificial es la ciencia que tiene como objetivo hacer que los ordenadores puedan realizar tareas de humanos tal y como los haría un humano. Dentro de la ciencia de la inteligencia Artificial se encuentra el subcampo del *Machine Learning* o Aprendizaje Automático. Este subcampo se encarga de entrenar a la máquina para que aprenda por sí solo.

El aprendizaje automático tiene como objetivo aprender de una base de datos llena de ejemplos y posteriormente ser capaz de aplicar el aprendizaje en ejemplos desconocidos para la máquina.

Para realizar el aprendizaje podemos empezar de diferentes puntos de partida. Si en los ejemplos proporcionados para aprender sabemos la salida que debemos obtener, estamos ante lo que se llama aprendizaje supervisado. Por otro lado, si no lo sabemos, estamos ante aprendizaje no supervisado. Aparte de estos dos métodos existen más métodos, como el aprendizaje semisupervisado, por refuerzo, multi tarea, transducción etc. En nuestro caso utilizaremos el aprendizaje supervisado.

Este TFG está basado en el algoritmo de clasificación basado en reglas difusas de asociación FARC-HD (*Fuzzy Rule-Based Classification model for High-Dimensional problems*) [1].

El algoritmo tiene tres fases. En la primera fase, se extraen reglas difusas de asociación. En la segunda fase, se minimizan las reglas preseleccionándolas conforme su calidad. Por último, en la tercera fase, se aplica un algoritmo genético para seleccionar las mejores reglas y ajustar las funciones de pertenencia.

Este trabajo se centra en cambiar el ajuste a las funciones de pertenencia. En el FARC-HD se ajusta de forma global con el fin de mantener la interpretabilidad del sistema. Nosotros estudiaremos el efecto de ajustarnos de forma local para cada regla.

Al ajustarnos de forma local, perdemos la interpretabilidad del sistema, pero logramos ajustarnos mejor a cada regla, por lo tanto, deberíamos mejorar el rendimiento. Por otro lado, para ajustarnos de forma local necesitaremos una base de datos para cada regla, lo que supone un aumento en el tiempo de computación.

2.- Preliminares

En este apartado del proyecto explicaremos en detalle el marco teórico necesario para entender el funcionamiento ~~del algoritmo empleado~~ en el TFG. Para ello, explicaremos los conceptos básicos paso a paso.

2.1.- Machine Learning

En los últimos años empujado por el Big Data el *Machine Learning* ha adquirido gran importancia. El *Machine Learning* se define como el aprendizaje de las máquinas y es un subcampo dentro de la Inteligencia Artificial.

Debido a la gran cantidad de datos que se generan diariamente y al potencial de conocimiento que estos datos tienen, surge la necesidad de analizarlos y procesarlos. La cantidad de datos es tal, que resulta imposible tratar los datos manualmente. De aquí nace el *Machine Learning*.

Todos los sistemas del *Machine Learning* generan un modelo y este es el que se encarga de procesar toda la información y tomar decisiones. Este modelo trabaja obteniendo una entrada de datos, procesándolos y obteniendo una salida sobre estos.

Toda técnica de Inteligencia Artificial tiene en común que intenta emular la inteligencia que aportaría un humano par la realización de esa labor. Lo que diferencia el *Machine Learning* del resto de técnicas IA es que este realiza un aprendizaje y actualizado automático de los parámetros de su modelo.

Hoy en día, la gran cantidad información que se obtiene diariamente ha generado que el *Machine Learning* se haya convertido en una de las técnicas más extendidas del mundo, gracias a la rápida y precisa capacidad de trabajo.

Dentro del *Machine Learning* podemos empezar de diferentes puntos de partida. Si en los ejemplos proporcionados para aprender sabemos la salida que debemos obtener, estamos ante lo que se llama aprendizaje supervisado. Por otro lado, si no lo sabemos, estamos ante aprendizaje no supervisado. Aparte de estos dos métodos existen más métodos, como el aprendizaje semisupervisado, por refuerzo, multi tarea, etc.

Nosotros nos centraremos en el aprendizaje supervisado, concretamente en los problemas de clasificación.

2.2.- Problemas de Clasificación

Los problemas de clasificación son un subcampo del *Machine Learning*. En este subcampo recibimos ejemplos y tratamos de clasificarlos en la clase a la que pertenecen. Para ello, partimos de un conjunto de ejemplos de los cuales sabemos a qué clase pertenecen. El objetivo es aprender los patrones generales que se repiten para pertenecer a cada clase y así clasificar correctamente ejemplos desconocidos.

Desarrollar....



2.3.- Lógica Difusa

La Lógica difusa la introdujo Lofti Zadeh [2] en 1965. En la lógica clásica, un elemento solo puede obtener dos valores: verdadero o falso. Es decir, un elemento puede pertenecer o no a un conjunto, pero el grado de pertenencia será absoluto. En la lógica difusa, por lo contrario, cada elemento tiene un grado de pertenencia con un valor real en el rango $[0,1]$ para cada conjunto.

Vamos a explicar el ejemplo clásico que se suele utilizar para entender la lógica difusa. Vamos a determinar si un paciente tiene fiebre o no. Desde la perspectiva de la lógica clásica tendremos fiebre si la temperatura del paciente es mayor que 39° y sino no. Es decir, tendremos fiebre si la temperatura del paciente pertenece al conjunto de "Fiebre Fuerte". Esta situación está representada en la Ilustración 1.

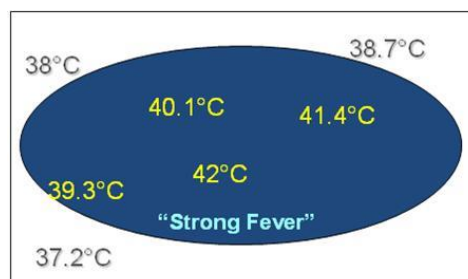


Ilustración 1. Ejemplo lógica Clásica

Por lo contrario, si representamos este caso con la lógica difusa, obtendríamos un valor real entre $[0,1]$ para el conjunto de "Fiebre Fuerte". Este caso, está representado en la Ilustración 2.

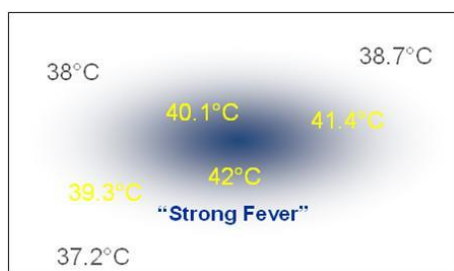


Ilustración 2. Ejemplo Lógica difusa.

La lógica difusa nos permite representar mejor la realidad, ya que podemos representar conceptos como mucho frio, poco alto, muy gordo...

A la hora de representar conjuntos difusos, tendremos las variables lingüísticas como altura, peso, temperatura... y también tendremos etiquetas lingüísticas como bajo, medio, alto... Podemos representarlo como en la Ilustración 3.

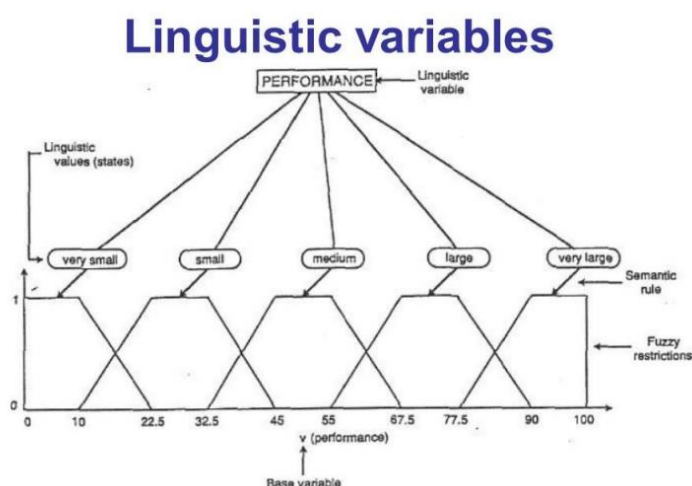


Ilustración 3. Variable difusa con sus etiquetas lingüísticas

Como estamos tratando con conjuntos, con la lógica difusa también se necesitan operadores de unión, intersección y complementario. Para estas operaciones, existen diferentes métodos llamados *t-normas*, *t-conormas*, y *negaciones*.

2.3.1- Operadores

1. t-normas

Definición: Una *t-norma* es una función $T: [0,1]^2 \rightarrow [0,1]$ que satisface:

- Cota: $T(x, 1) = x \forall x \in [0,1]$
- Monotonía: $T(x, y) \leq T(z, u)$ si $x \leq z$ e $y \leq u$

- c. Conmutativa: $T(x, y) = T(y, x)$
- d. Asociativa: $T(T(x, y), z) = T(x, T(y, x))$

Estos son algunos de los operadores más utilizados para representar la intersección:

- *Mínimo:* $T(x, y) = \min(x, y)$
- *Producto:* $T(x, y) = x * y$
- *Lukasiewicz:* $T(x, y) = \max(0, x + y - 1)$



2. T-conormas

Definición: Una t-conorma es una función $S: [0,1]^2 \rightarrow [0,1]$ que satisface:

- a. *Cota:* $S(x, 0) = x \quad \forall x \in [0,1]$
- b. *Monotonía:* $S(x, y) \leq S(z, u)$ si $x \leq z$ e $y \leq u$
- c. *Conmutativa:* $S(x, y) = S(y, x)$
- d. *Asociativa:* $S(S(x, y), z) = S(x, S(y, x))$

Estos son algunos de los operadores más utilizados para representar la unión:

- *Máximo:* $S(x, y) = \max(x, y)$
- *Suma algebraica:* $S(x, y) = x + y - x * y$
- *Lukasiewicz:* $S(x, y) = \min(1, x + y)$



3. Negaciones

Definición: $c: [0,1] \rightarrow [0,1]$ es una negación difusa si y solo si:

- a. $c(0) = 1$ y $c(1) = 0$
- b. $c(x) \leq c(y)$ si $x \geq y$

Además, se dice que una negación es estricta si:

- c. $c(x)$ es continua
- d. $c(x) < c(y)$ si $x > y$

E involutiva si:

- e. $c(c(x)) = x$

Cuando una negación es estricta e involutiva se dice que es una negación fuerte. Así, dado un conjunto A, su complementario es $A_c = c(\mu_A(u))$ siendo c una negación fuerte.

2.3.2.- Implicación

En dos universos U y V , una relación difusa es un conjunto dado por el producto cartesiano $U \times V$ que se puede definir con una función de pertenencia $R(u, v)$.

$$IF a \in A THEN b \in B \quad (1)$$

Una implicación (1) se define como un caso especial de relación difusa, la cual es posible definir mediante una función de pertenencia $\mu_{A \rightarrow B}(x, y)$. Y a partir de esa función podemos construir una matriz R que represente la Regla.

Los operadores de implicación son funciones $I: [0,1] \times [0,1] \rightarrow [0,1]$ que satisfacen las propiedades:

1. Si $x \leq z$ entonces $I(x, y) \geq I(z, y)$
2. Si $y \leq t$ entonces $I(x, y) \leq I(x, t)$
3. $I(0, x) = 1$ (dominancia falsedad)
4. $I(x, 1) = 1$
5. $I(1, 0) = 0$

Estos son algunos de los operadores de implicación más utilizados:

- Kleene-Dienes: $\mu_{A \rightarrow B}(x, y) = (1 - \mu_A(x)) \cup \mu_B(y)$
- Lukasiewicz: $I(x, y) = \min(1, 1 - x + y)$
- Mamdani: $I(x, y) = \mu_{A \rightarrow B}(x, y) = \mu_A(x) \cap \mu_B(y)$
- Zadeh: $\mu_{A \rightarrow B}(x, y) = (1 - \mu_A(x)) \cup (\mu_A(x) \cap \mu_B(y))$
- Larsen: $\mu_{A \rightarrow B}(x, y) = \mu_A(x) * \mu_B(y)$

2.3.3.- Inferencia

2.3.4.- Reglas Difusas

Para construir una relación R que represente una regla, se pueden utilizar operadores de implicación o t-normas. Las t-normas en la práctica suelen mejorar el rendimiento, pero no aprovechan parte de la teoría de la lógica clásica que con los operadores de implicación sí.

Para la regla (1) una opción para construir la regla puede ser así:

$$R(x, y) = I(A(x), B(y)) \quad (2)$$

Existen distintos tipos de reglas difusas:

1. Reglas difusas con una clase en el consecuente

R_k : Si X_1 es A_1^k y ... y X_n es A_n^k entonces Y es B_j



Donde X_1, \dots, X_n son las variables de entrada, A_1^k, \dots, A_n^k son las etiquetas lingüísticas de las variables e Y es la variable de salida que indica a cuál de las clases B_j del conjunto de clases pertenece el ejemplo.

2. Reglas difusas con una clase y un grado de certeza asociado

R_k : Si X_1 es A_1^k y ... y X_n es A_n^k entonces Y es B_j con grado r^k

Donde X_1, \dots, X_n son las variables de entrada, A_1^k, \dots, A_n^k son las etiquetas lingüísticas de las variables e Y es la variable de salida que indica a cuál de las clases B_j del conjunto de clases pertenece el ejemplo y r^k es el grado de certeza de la regla R_k .

3. Reglas difusas con grados de certeza asociados a cada clase del consecuente

R_k : Si X_1 es A_1^k y ... y X_n es A_n^k entonces (r_1^k, \dots, r_m^k)

Donde X_1, \dots, X_n son las variables de entrada, A_1^k, \dots, A_n^k son las etiquetas lingüísticas de las variables y el consecuente es un vector de pesos en el que cada r_j^k denota el grado de certeza de la regla R_k para predecir la clase B_j .

2.4.- Sistemas de Clasificación Basados en Reglas Difusas

Los sistemas de clasificación basados en Reglas Difusas (SCBRD), se utilizan para resolver problemas de clasificación. La ventaja que tienen es que proporcionan resultados precisos a la vez que interpretables. Esto se logra gracias a los términos lingüísticos utilizados en los antecedentes de las reglas.

El esquema de este tipo de clasificador se muestra en la Ilustración 4. Este tipo de clasificador está formado por dos componentes principales:

1. Base de Conocimiento (BC): La base del conocimiento se encarga de almacenar toda la información aprendida a partir de una base de datos de un problema específico. La componen los siguientes elementos:

- Base de Datos (BC):** Contiene la definición de los conjuntos difusos asociados a los términos lingüísticos utilizados por la Base de Reglas.
- Base de Reglas (BR):** Está formada por un conjunto de n Reglas de Difusas de clasificación ~~(más atrás)~~.

$$R = \{R_1, \dots, R_n\}$$



2. **Método de Razonamiento Difuso (MDR):** Utiliza la información contenida en la Base de Conocimiento para determinar la clase a la que pertenece cualquier patrón de datos admisible. Para ello, utiliza el razonamiento aproximado propio de los sistemas difusos.

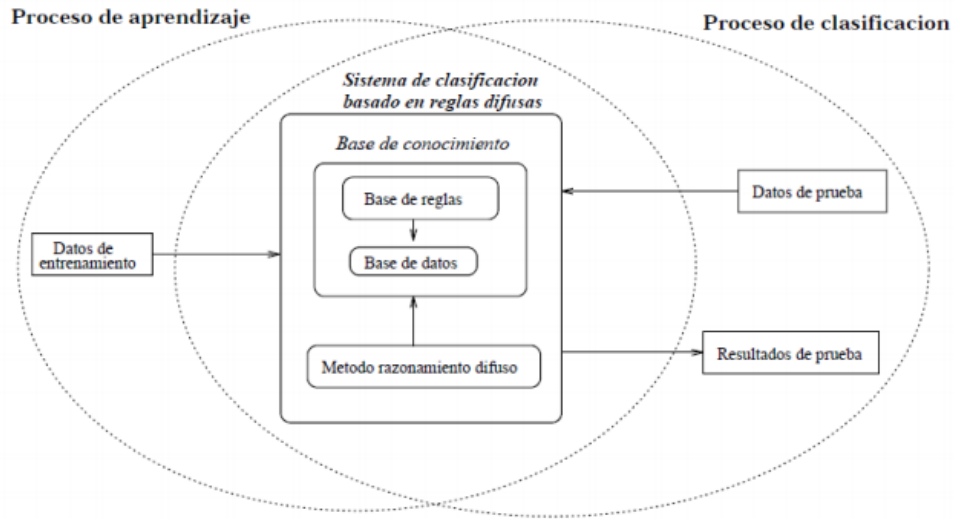


Ilustración 4. Esquema de un SCBRD

2.4.1.- Método De Razonamiento Difuso

El método de razonamiento difuso (MRD) es un procedimiento de inferencia que utiliza la información de la BC para predecir una clase ante un ejemplo no clasificado. Al final, indica como aplicar la información contenida en la BR, por ello, es uno de los elementos más importantes de un SCBRD, ya que determinará el rendimiento.

Sea $x_p = (x_{p1}, \dots, x_{pn})$ un nuevo ejemplo a clasificar. Generalmente, un MDR utilizará el siguiente método para clasificarlo:

1. *Grado de emparejamiento.* Se calcula el grado de emparejamiento del ejemplo x_p con los antecedentes de cada regla de la BR:

$$\mu_{A_j}(x_p) = T\left(\mu_{A_{j1}}(x_{p1}), \dots, \mu_{A_{jn_j}}(x_{pn_j})\right) \text{ con } j = \{1, \dots, L\} \quad (2.1)$$

Siendo $\mu_{A_{ji}}(x_p)$ el grado de pertenencia del ejemplo con el antecedente i -ésimo de la regla, T una t-norma y n_j es el número de antecedentes de la regla R_j .

2. *Grado de asociación.* El grado de asociación del patrón x_p con cada regla de la BR se calcula así:

$$b_j(x_p) = \mu_{A_j}(x_p) * RW_j \quad (2.2)$$

Donde RW_j es el peso de la regla j -ésima, R_j .

3. *Grado de confianza.* Se calcula el grado de confianza para cada clase. Se pueden utilizar diferentes funciones de agregación para calcularlo. Las más típicas son el máximo (MRD de la regla ganadora) y la suma (MRD de combinación aditiva) [3].
4. *Clasificación.* Se predice la clase con grado de confianza más alto.

$$Clase = \arg \max_{l=1, \dots, m} (conf_l(x_p)) \quad (2.3)$$

2.5.- Algoritmos Genéticos

Dentro de los modelos de computación bio-inspirados se encuentran los algoritmos evolutivos, las redes neuronales, los algoritmos basados en enjambres etc. Nosotros vamos a fijarnos en los algoritmos evolutivos.

Los algoritmos evolutivos tienen como objetivo resolver problemas de optimización o búsqueda y lo que los distingue es que el elemento clave de su diseño es algún mecanismo de evolución. Estos algoritmos intentan imitar el funcionamiento de los cromosomas en la vida real.

Los principales componentes de los algoritmos evolutivos son los siguientes:

- Población de individuos. Cada individuo representa una posible solución y se le llama cromosoma.
- Procedimiento de selección. Tiene algún mecanismo de selección basado en la aptitud de los individuos para resolver el problema.
- Procedimiento de transformación. En la evolución del programa se generan nuevos individuos a partir de los anteriores.

Existen distintos tipos de algoritmos evolutivos, pero en todos ellos el funcionamiento general es el siguiente (Ilustración 5):

1. Partimos de una población inicial de cromosomas. El número de cromosomas que conforman la población se mantiene fija.
2. Se evalúa la aptitud de cada cromosoma de la población.
3. Se seleccionan los cromosomas que se utilizarán para crear una nueva generación.
4. Se combinan los cromosomas seleccionados y se generan nuevos cromosomas. Los cromosomas pueden sufrir mutación.

5. Se genera la nueva población. Esta nueva población la completan los mejores cromosomas resultantes del conjunto de cromosomas de la generación anterior y la nueva.
6. Se repite el proceso de evaluación, selección y combinación hasta optimizar el resultado.

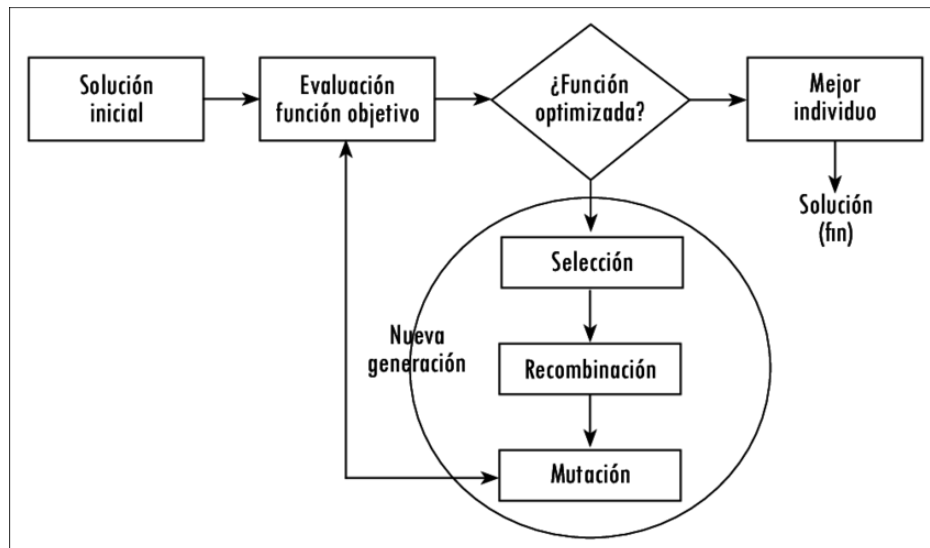


Ilustración 5. Esquema algoritmo evolutivo

~~Los algoritmos genéticos son un tipo de algoritmo evolutivo. Las principales características de un algoritmo genético son 3:~~

- ~~1. Los cromosomas se representan mediante codificación binaria.~~
- ~~2. Como método de selección suelen emplear el método de la ruleta con elitismo.~~
- ~~3. Como método de combinación emplean el cruce y la mutación.~~

A continuación, vamos a explicar más detalladamente en que se basan o como se realizan los apartados de codificación, cruce, mutación y selección.

2.5.1.- Codificación

La codificación es la forma en la que se representa la información del mundo real en nuestro cromosoma. Recordemos que un cromosoma debe representar una posible solución a nuestro problema, por lo tanto, la codificación es la forma en la que representamos una posible solución en un cromosoma.

Tenemos diferentes formas de codificación como la representación real, codificación entera, alfanumérica, codificación binaria etc. En todos ellos, la codificación se basa en una secuencia de reales, enteros, ceros y unos etc. ~~Los algoritmos genéticos utilizan la codificación binaria (Ilustración 5).~~

1	1	0	1	0	1	1	0
---	---	---	---	---	---	---	---

Ilustración 6. Ejemplo cromosoma en codificación binaria

2.5.2.- Cruce

Los operadores de cruce se encargan de generar nuevos cromosomas para una nueva generación a partir de cromosomas de la generación anterior. Por ejemplo, a partir de dos cromosomas padres, generamos uno o más descendientes para la siguiente generación. De este modo permitimos evolucionar al algoritmo genético. Existen diferentes operadores de cruce como el *cruzamiento sobre un punto*: A partir de dos cromosomas de longitud K, se elige un punto entre 0 y K-1 y se mezclan a partir de dicho punto (Ilustración 7).

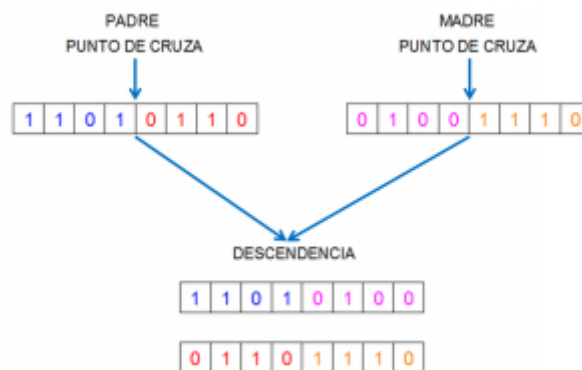


Ilustración 7. Cruzamiento sobre punto

Otro ejemplo de cruzamiento es el *cruzamiento uniforme*: a partir de dos cromosomas, elegimos aleatoriamente que genes hereda el progenitor de cada padre (Ilustración 8).

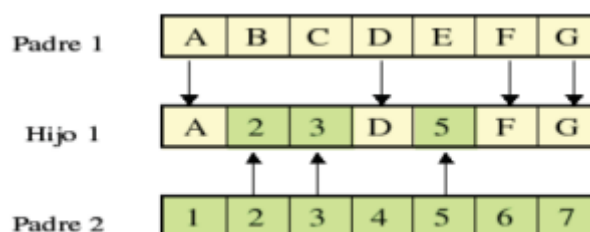


Ilustración 8. Cruzamiento uniforme

2.5.3.- Mutación

Basándose en la naturaleza la mutación consiste en modificar ligeramente el cromosoma después de aplicar el cruce. El objetivo de la mutación es dar más

flexibilidad de evolución a la población. El algoritmo al final evoluciona gracias a los operadores de cruce y la mutación.

La mutación puede aplicarse de distintos modos, como cambiando un valor del cromosoma, invertir el orden de los valores del cromosoma en una parte o intercambiar valores del cromosoma etc.

2.5.4.- Selección

El objetivo principal de los métodos de selección es escoger los individuos mejor adaptados del conjunto de cromosomas de la anterior generación y la nueva generación obtenida a partir del cruce y la mutación. Para realizar la selección existe diferentes métodos como el método de la ruleta. Este método se basa en dividir el intervalo $[0,1]$ en tantos subintervalos como progenitores. El tamaño de cada subintervalo depende directamente de la probabilidad de selección de cada progenitor. De este modo, cuanto mejor adaptado este un progenitor mayor probabilidad de selección tiene. Para seleccionar un progenitor generamos un número aleatorio entre $[0,1]$ y vemos a que cromosoma pertenece dicho intervalo.

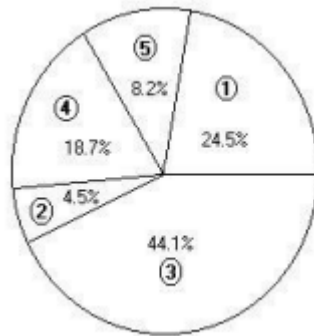


Ilustración 9. Método de la ruleta

2.6.- FARC-HD

Uno de los SCBRDs más interpretables de la literatura es el algoritmo FARC-HD (Fuzzy Association Rule-Based Clasification Model for High-Dimensional problems with Genetic Rule Selection and Lateral Tuning) [1]. El algoritmo utiliza la siguiente estructura de reglas:

$$\text{Regla } R_j: \text{ Si } x_1 \text{ es } A_{j1} \text{ y } \dots \text{ y } x_n \text{ es } A_{jn} \text{ entonces Clase} = C_j \text{ con } RW_j \quad (3.1)$$

Donde R_j es la etiqueta de la regla j-ésima, $x = (x_1, \dots, x_n)$ representa un ejemplo a través de un vector n-dimensional, A_{ji} representa un conjunto difuso, $C_j \in \mathbb{C}$ es la etiqueta de la clase y RW_j es el peso de la regla.

El peso RW_j , se calcula mediante el valor de confianza o grado de certeza definido en [5]:

$$RW_j = CF_j = \frac{\sum_{x_p \in \text{class } C_j} \mu_{A_j}(x_p)}{\sum_{p=1}^N \mu_{A_j}(x_p)} \quad (3.2)$$

donde $\mu_{A_j}(x_p)$ representa el grado de emparejamiento del ejemplo x_p con los antecedentes de la regla difusa calculado mediante la Ecuación 2.1. En este caso, se utiliza la t-norma producto.

A la hora de modelar las etiquetas lingüísticas, en el FARC-HD se modelan mediante funciones de pertenencia triangulares uniformemente distribuidas, tal y como se muestra en la Ilustración 10.

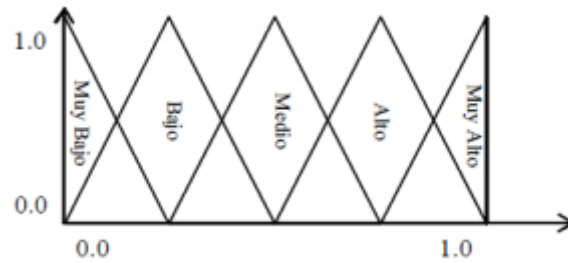


Ilustración 10. Etiquetas lingüísticas FARC-HD

2.6.1.- Proceso de aprendizaje

El proceso de aprendizaje que se desarrolla en FARC-HD, está compuesto por tres etapas:

1. **Extracción de reglas difusas para clasificación:** Utilizamos un árbol de búsqueda para generar reglas difusas de asociación. Limitamos la profundidad máxima del árbol.
2. **Filtrado de reglas candidatas:** A causa de que obtenemos demasiadas reglas, aplicamos un filtrado basado en la mejora relativa de la medida de precisión (wWR_{Acc}) junto al algoritmo APRIORI-SD [7]. De este modo nos quedamos con las reglas más relevantes.
3. **Selección evolutiva de reglas y ajuste de etiquetas:** Por último, se emplea un algoritmo genético CHC [8] que mezcla la selección de reglas con el tuning [9].

2.6.2.- Proceso de Clasificación

FARC-HD aplica el método de Razonamiento Difuso llamado *combinación aditiva* [10]. El proceso empleado a la hora de clasificar un ejemplo dado $x_p = (x_{p1}, \dots, x_{pn})$ es el descrito en el apartado 2.4.1.- Método De Razonamiento Difuso.

La combinación aditiva se basa en utilizar el método de la suma para obtener el grado de confianza:

$$\text{conf}_l(x_p) = \sum_{R_j \in RB; C_j=l} B_j(x_p), l = 1, 2, \dots, m \quad (3.2)$$

2.6.3.- Algoritmo Genético en FARC-HD

Ahora explicaremos las características del algoritmo evolutivo empleado en FARC-HD:

1. *Codificación de los Cromosomas:* A la hora de codificar los cromosomas, codificamos dos partes distintas. Por un lado, codificamos el ajuste de las etiquetas C_T y por otro la selección de reglas C_S .

Para la selección de reglas, empleamos una codificación binaria donde el 1 significa que hemos seleccionado la regla y el 0 indica que no. De este modo, obtenemos la siguiente forma $C_S = \{C_1, \dots, C_{P_r}\}$, siendo P_r en número de reglas candidatas.

Para el ajuste de las etiquetas (tuning), empleamos una codificación en valores reales que representan el parámetro α . Este parámetro indica el desplazamiento de las etiquetas lingüísticas tal y como se ve en la Ilustración 11.

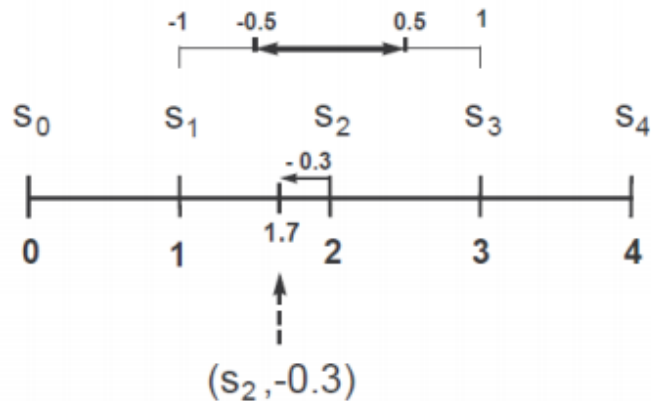


Ilustración 11. Desplazamiento de una etiqueta lingüística

Las etiquetas lingüísticas, tienen un rango máximo de desplazamiento que es el ± 0.5 . Con esto evitamos que las etiquetas se intercambien las posiciones. Un ejemplo de desplazamiento lateral hacia la izquierda la podemos ver en la Ilustración 12.

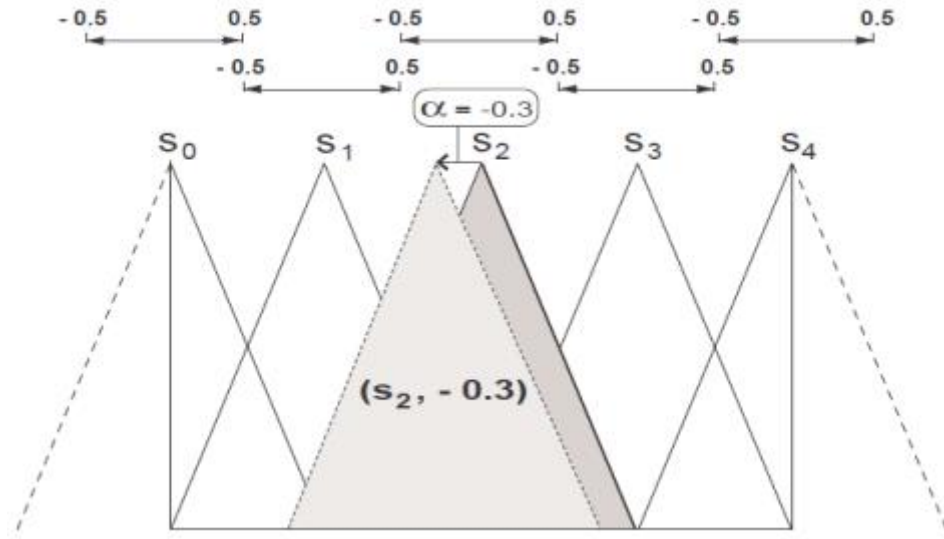


Ilustración 12. Desplazamiento lateral de una etiqueta

De este modo tenemos una codificación tal que así: $C_T = \{C_{11}, \dots, C_{1m^1}, C_{21}, \dots, C_{2m^2}, \dots, C_{n1}, \dots, C_{nm^n}\}$. Donde (m^1, m^2, \dots, m^n) indica el número de etiquetas por cada una de las n variables.

En total, un cromosoma nos quedaría tal y como se ve en la Ilustración 13. $C = \{C_T, C_S\}$.

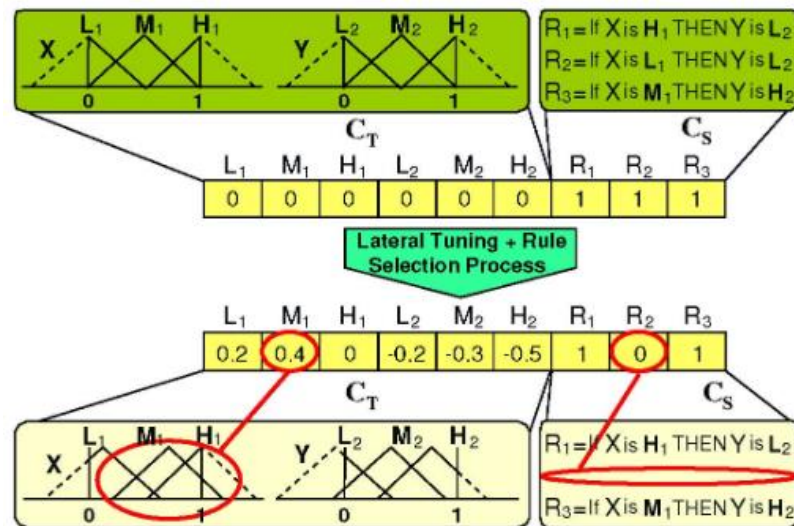


Ilustración 13. Ejemplo codificación genética tuning lateral y selección de reglas

2. *Evaluación de los cromosomas:* Para la evaluación de los cromosomas se emplea la siguiente función fitness:

$$Fitness(C) = \frac{\#Hits}{N} - \delta \cdot \frac{NR_{initial}}{NR_{initial} - NR + 1.0} \quad (3.3)$$

Donde $\#Hits$ es el número de ejemplos correctamente clasificados, $NR_{initial}$ es el número de reglas candidatas, NR el número de reglas seleccionadas y δ un parámetro que determina la compensación entre el porcentaje de acierto y la complejidad, ~~dado por un $\{C_T, C_S\}$ experto.~~

3. *Operadores de cruce:* En cada parte del cromosoma emplea un operador de cruce distinto.

Para el ajuste de las etiquetas (C_T), emplea el operador *Parent Centric BLX* (PCBLX) [11]. El operador se basa en el concepto de vecindad, y permite a los genes resultantes del cruce desplazarse a una zona próxima a la de los progenitores.

Para la selección de reglas (C_S), emplea el operador *Half Uniform Crossover Scheme* (HUX) [12]. El operador intercambia la mitad de los genes que sean diferentes en ambos progenitores.

4. *Modelo Evolutivo CHC:* FARC-HD utiliza el modelo evolutivo CHC [8]. Este modelo, emplea mecanismos de selección elitistas en la población para avanzar en la búsqueda global. Además, también emplea mecanismos de prevención de incesto mediante la distancia *hamming*. El mecanismo de incesto se basa en asegurar que los padres seleccionados para el cruce no son muy parecidos, y para ello solo los cruza si la distancia entre ellos es mayor que un umbral. Por último, el algoritmo cada vez que no se generan nuevos individuos, decrece el umbral, e itera hasta que el umbral es menor que cero. Podemos ver un esquema del funcionamiento en la Ilustración 14.

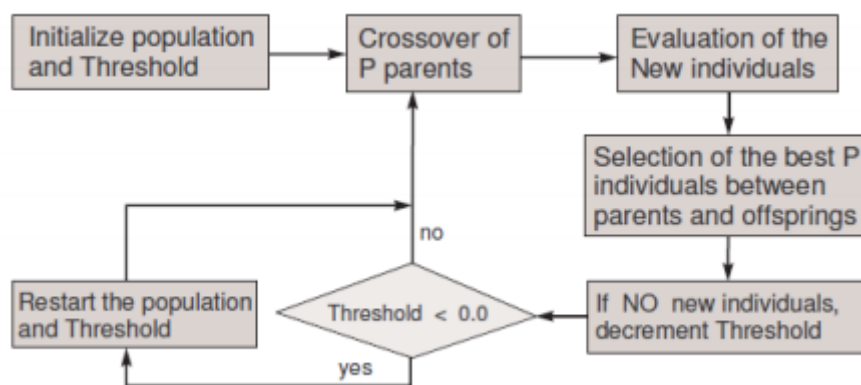


Ilustración 14. Esquema CHC

3.- FARC-HD-LOCAL

La idea principal de este TFG, consiste en aplicar tuning local en vez de tuning global en la última fase del FARC-HD original. Con este cambio se pretende dar más flexibilidad a las reglas para poder ajustarnos mejor y así obtener mejores resultados a costa de perder la interpretabilidad.

En esta sección primero explicaremos en qué consistirá el tuning local y cómo afectará al algoritmo y luego estudiaremos como mejorar la efectividad del tuning local. En el marco experimental se verán los resultados obtenidos para cada configuración y si es efectivo o no.

3.1.- Tuning local en FARC-HD

El proceso de entrenamiento del FARC-HD tiene tres fases, tal y como se explica anteriormente en **2.6.3.- Algoritmo Genético en FARC-HD**:

1. Extracción de reglas difusas para clasificación
2. Filtrado de reglas candidatas
3. Selección evolutiva y ajuste de la posición de las etiquetas

Para aplicar el tuning lateral local, nosotros solo modificaremos la última fase del FARC-HD original, en las demás fases no será necesaria ninguna modificación. En esta sección vamos a describir todos los cambios que conlleva aplicar el tuning local, primero explicaremos en que se basa el tuning local y luego como codificamos dichos cambios.

3.1.1.- Bases ~~Tuning~~ local

Se le llama tuning al proceso del ajuste de la posición de las etiquetas. En el FARC-HD original, el tuning se hace de forma lateral y global. De tal modo que si desplazamos a la izquierda una etiqueta como en la Ilustración 12, para todas las reglas tenemos el mismo desplazamiento.

El tuning lateral local consiste en ~~ampliar el rango de búsqueda y de las reglas.~~ Para ello, ~~dejamos que cada regla se ajuste independientemente a cada etiqueta.~~ De

este modo si por ejemplo tenemos R_1, \dots, R_n reglas y E_1, \dots, E_5 etiquetas, el desplazamiento de la etiqueta E_1 para la regla R_1 no es el mismo que el desplazamiento de la etiqueta E_1 para todas las demás reglas. Al final esto se traduce en tener lo mismo que tenemos en la Ilustración 12 para cada etiqueta de cada regla.

3.1.2.- Codificación tuning local

Como ahora vamos a tener un ajuste independiente para cada etiqueta de cada regla, necesitamos un cromosoma que contenga toda esta información. El cromosoma que vamos a utilizar para codificar el tuning local, consistirá en una matriz G donde cada fila será igual que el C_T del cromosoma original empleado en FARC-HD, más una última celda que representa si seleccionamos la regla o no. Tendremos tantas filas como reglas. La siguiente matriz es una ilustración gráfica:

$$G = \begin{pmatrix} a_{11} & \cdots & e_{11} & \cdots & a_{m1} & \cdots & e_{m1} S_1 \\ \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ a_{1n} & \cdots & e_{1n} & \cdots & a_{m1} & \cdots & a_{mn} S_n \end{pmatrix}$$

donde hay cinco etiquetas a, b, c, d, e . Además, tenemos m variables y n reglas. Por último, la última columna de la matriz representa el C_S del cromosoma.

A la hora de programar, también hay que tener en cuenta que como vamos a ajustarnos de forma independiente para cada regla, también necesitamos una base de datos independiente para cada regla.

3.2.- Mejorando efectividad del tuning local

El FARC-HD original, emplea distintos parámetros a lo largo de todo el algoritmo, los cuales están optimizados para aprender de la mejor manera posible. Ahora, como hemos cambiado el ajuste de las etiquetas aplicando tuning local, debemos estudiar si los parámetros que afecten de forma directa siguen teniendo un valor óptimo o no. Para ello debemos recordar el **2.6.3.- Algoritmo Genético en FARC-HD**.



Para empezar, si nos fijamos en el funcionamiento del modelo evolutivo, también mostrado en la Ilustración 14, vemos que estamos ante un proceso cíclico que depende del *threshold* o *umbral* y el número de iteraciones.

Modificar el número de iteraciones es sencillo, puede pasar que el tuning local necesite más iteraciones dado que tiene un espacio de búsqueda más amplio. Por lo tanto, puede pasar que localmente vayamos ajustándonos poco a poco, pero sin pausa y que el número de iteraciones predefinido en le FARC-HD original se nos quede corto. Para evaluar la influencia hay que estudiar la convergencia del algoritmo a lo largo de las iteraciones.

Por otro lado, tenemos el *threshold* o *umbral*. El operador de cruce empleado para realizar el ~~entrecruzamiento~~ de los padres es un operador basado en la vecindad. Ahora, como hemos modificado la codificación del cromosoma también debemos estudiar como se ~~define la vecindad entre distintos cromosomas~~ y comprobar si es mejorable o no.

En el FARC-HD original [1], el mecanismo de incesto se controla mediante un parámetro denominado L. Este parámetro, empieza con un valor grande que representa la distancia entre dos cromosomas, es decir, cuanto mayor sea el valor, más distintos significa que son. Luego, cada vez que itera y no obtiene cromosomas nuevos decrementa el valor de L para forzar el cruce entre cromosomas más parecidos. L originalmente se inicializa con el siguiente valor:

$$L = \frac{nLabels * BITS_GEN + nReglas}{4}$$

donde *nLabels* es el número de variables por el número de etiquetas; *BITS_GEN* es una constante ~~que introducimos nosotros~~ y *nReglas* es el número de reglas.

Como ahora tenemos más valores en el cromosoma, la distancia que inicialmente utilizábamos ~~puede quedarnos demasiado corta, por lo que interesa estudiar si este parámetro se nos queda demasiado exigente y por lo tanto merece la pena ampliar la distancia para llegar a mejores resultados.~~ En el **5.- Estudio**

Experimental estudiamos este caso con la siguiente inicialización:

$$L = \frac{(nLabels * BITS_GEN * nReglas) + nReglas}{2}$$

Otra modificación con intención de no perder toda la interpretabilidad y obtener un modelo semi-interpretable, puede consistir en modificar la inferencia del algoritmo. El FARC-HD original, emplea la inferencia basado en la suma de pertenencia, tal y como se explica en **2.6.2.- Proceso de Clasificación**. En vez de utilizar este método, podríamos emplear otro método para calcular el grado de confianza, que mantenga la interpretabilidad. Un método que cumpla dichos requisitos puede ser el máximo. Con el cálculo del grado de confianza basado en el máximo, solo tendríamos en cuenta el desplazamiento de la etiqueta de la regla seleccionada, por lo tanto, mantenemos la interpretabilidad. En el caso del calculo del grado de confianza basado en la suma, estamos sumando el grado de pertenencia de todas las etiquetas ajustadas independientemente conforme a las otras, y este proceso es el que hace perder la interpretabilidad.

Para desarrollar el modelo semi-interpretable hay que tener en cuenta que, con el máximo, perdemos información de las otras reglas y con la suma sin embargo no. Por lo tanto, este modelo generalmente obtendrá peores resultados ya que no emplea toda la información disponible. Es por eso, que esta dirección de investigación solo hay que desarrollarla si conseguimos una mejora considerable con el ajuste local frente al ajuste global.

Por último, también puede pasar que al ajustarnos localmente dejemos algún ejemplo sin cubrir y lo metamos en la clase por defecto. La clase por defecto es la clase que más ejemplos tiene, por lo que lo introducimos ahí por estadística de acierto. Sin embargo, si son muchos los ejemplos que clasificamos de este modo conseguimos un efecto contraproducente y provocamos un aprendizaje menos eficiente. Si se diera este caso, podríamos aplicar una clasificación más eficiente como el basado en los centroides como el KNN por ejemplo. Es por ello por lo que es importante estudiar si obtenemos muchos ejemplos no cubiertos o no.

4.- Marco Experimental

En este apartado se presenta el contexto elegido para la realización de los experimentos. En primer lugar, describimos los conjuntos de datos (*datasets*) utilizados. Luego, mostramos los parámetros con los que se han ejecutado las pruebas. Por último, definimos las medidas de rendimiento empleadas para evaluar los resultados.

4.1.- Datasets

Hemos seleccionado veintinueve conjuntos de datos (*datasets*) que son los mismos utilizados en [13] para evaluar el aprendizaje de sistemas de clasificación basadas en reglas difusas. Estos conjuntos de datos contienen datos del mundo real obtenidos del repositorio de KEEL [14], disponibles públicamente desde el sitio web del proyecto¹. En la Tabla 1. [Descripción de las características de los dataset utilizados](#) resumimos las características principales de cada conjunto: número de ejemplos (#Ej.), número de atributos (#Atr.), y número de clases (#Clas.). Los ejemplos con valores nulos como los del dataset *bands*, *Cleveland* y *Wisconsin* han sido eliminados tal y como en [13].

Para evaluar el rendimiento de los algoritmos, se ha utilizado un modelo de validación cruzada de 5 particiones [15]. Los resultados finales mostrados en el estudio experimental son el promedio del rendimiento *accuracy rate*, que [son](#) el porcentaje de ejemplos clasificados correctamente [para](#) cada una de las 5 particiones.

4.2.- Configuración de parámetros

En este apartado se muestran los parámetros con los que se ha realizado la experimentación. Dichos parámetros son los mismos que se utilizan en el FARC-HD [1] para realizar las pruebas. En el FARC-HD los parámetros que se utilizan son los recomendados por los autores para cada sección del algoritmo global [16]. Los parámetros que hemos utilizado se muestran en la Tabla 2.

¹ <https://sci2s.ugr.es/keel/datasets.php>

Tabla 1. Descripción de las características de los dataset utilizados

Dataset	#Ej.	#Atr.	#Clas.
<i>banana</i>	5300	2	2
<i>bupa</i>	345	6	2
<i>cleveland</i>	297	13	5
<i>ecoli</i>	336	7	8
<i>glass</i>	214	9	6
<i>haberman</i>	306	3	2
<i>ion</i>	351	33	2
<i>iris</i>	150	4	3
<i>magic</i>	1902	10	2
<i>newthyroid</i>	215	5	3
<i>pageblocks</i>	547	10	5
<i>penbased</i>	1099	16	10
<i>phoneme</i>	5404	5	2
<i>pima</i>	768	8	2
<i>ring</i>	740	20	2
<i>satimage</i>	643	36	7
<i>shuttle</i>	5800	9	7
<i>tae</i>	151	5	3
<i>titanic</i>	2201	3	2
<i>twonorm</i>	740	20	2
<i>vehicle</i>	846	18	4
<i>wine</i>	178	13	3
<i>wisconsin</i>	683	11	2
<i>bands</i>	365	19	2
<i>spectfheart</i>	267	44	2
<i>wdbc</i>	569	30	2
<i>yeast</i>	1484	8	10
<i>winequality-red</i>	1599	11	11
<i>balance</i>	625	4	3

4.3.- Medidas de Rendimiento

Para evaluar el rendimiento de las pruebas, utilizamos el grado de precisión o *accuracy rate*. Este parámetro calcula el porcentaje de ejemplos clasificados correctamente. Por otro lado, con tal de aportar soporte estadístico a los resultados, utilizamos la prueba de rangos con signo Wilcoxon [17].

Tabla 2. Configuración de Parámetros

Parámetros
<i>Número de etiquetas lingüísticas por variable: 5</i>
<i>Soporte mínimo: 0.05</i>
<i>Confianza mínima: 0.8</i>
<i>Profundidad máxima Árbol: 3</i>
<i>Parámetro K: 2</i>
<i>Número máximo de evaluaciones: 20000</i>
<i>Número de individuos: 50</i>
<i>Parámetro α: 0.02</i>
<i>Bits por gen: 30</i>
<i>Inferencia: 1 (Combinación aditiva)</i>

5.- Estudio Experimental

En este capítulo, se muestran y analizan los resultados obtenidos por los dos métodos de tuning utilizados, con distinta configuración de parámetros. Para ello, hemos dividido el estudio experimental en distintas etapas:

1. Se presentan los resultados obtenidos con el FARC-HD global y FARC-HD local en sus configuraciones, por defecto y se analizan.
2. Se comparan los resultados obtenidos en FARC-HD global y FARC-HD local con las distintas configuraciones de parámetros.
3. Se comparan todas las configuraciones y sus respectivos resultados.

En las tablas el nombre de los algoritmos esta codificado con su configuración. La primera letra indica si es global (g) o local (l). El siguiente número indica el número de iteraciones; 20000 (20) o $num_etiquetas * num_variables * 5000$ (var). Por último, el último número indica si el parámetro L esta modificado (2) o no (4). Tal y como se explica en **3.2.- Mejorando efectividad del tuning local**.

5.1.- Comparando FARC-HD global y FARC-HD local con las configuraciones originales

En esta sección, primero mostramos los resultados obtenidos en los dos modelos con la misma configuración de parámetros con los que se ejecuta el FARC-HD [1]. Para obtener dichos resultados se utilizan los ~~los 4.1.-~~ **Datasets** definidos previamente. Podemos observar los resultados en la Tabla 3.

Los resultados muestran que localmente sin modificar ningún parámetro, no obtenemos el efecto de mejor ajuste que estábamos buscando ya que el tuning global logra resultados ligeramente mejores.



Tabla 3. FARC-HD global vs FARC-HD local



	g20-4		l20-4	
	Train	Test	Train	Test
<i>banana</i>	86,9811321	86,1886792	86,7641509	85,8867925
<i>bupa</i>	79,4202899	66,0869565	80,0724638	62,6086957
<i>cleveland</i>	89,8996561	56,8926554	88,8887707	58,2485876
<i>ecoli</i>	92,1880375	82,1378402	92,1874827	81,8481124
<i>glass</i>	81,3096695	67,2978959	82,0121039	64,5182724
<i>haberman</i>	81,2084309	74,4949762	81,8618267	71,2268641
<i>ion</i>	98,8607016	88,8933602	98,6474326	88,889336
<i>iris</i>	98,5	94,6666667	98,6666667	94,6666667
<i>magic</i>	84,2533484	80,9650504	84,5030804	81,280978
<i>newthyroid</i>	99,1860465	95,8139535	99,1860465	95,8139535
<i>pageblocks</i>	97,0796018	93,9749791	97,0340437	93,6113428
<i>penbased</i>	98,4772727	92,1818182	98,6136364	91,4545455
<i>phoneme</i>	83,2809352	81,513482	83,5075697	81,3658992
<i>pima</i>	83,8212971	74,0853917	83,7890946	75,1243528
<i>ring</i>	97,0945946	90,5405405	97,2635135	91,7567568
<i>satimage</i>	85,5754599	81,0222868	85,4974878	79,934593
<i>shuttle</i>	97,1494253	96,9655172	96,5862069	96,2758621
<i>tae</i>	76,4917355	59,6129032	76,822314	57,6344086
<i>titanic</i>	79,0663427	78,8733251	79,0663427	78,8733251
<i>twonorm</i>	97,8040541	90,2702703	97,6689189	90
<i>vehicle</i>	80,4074275	69,2655761	80,2006765	68,5638705
<i>wine</i>	100	96,047619	100	93,8095238
<i>wisconsin</i>	98,8286424	96,9246458	98,6090631	96,9246458
<i>bands</i>	89,1095801	69,6973839	88,8351384	69,6748401
<i>spectfheart</i>	93,3522004	78,6373166	92,8844719	79,0216632
<i>wdbc</i>	98,5937922	96,6604565	98,6378446	95,9587021
<i>yeast</i>	64,2185829	58,490536	64,6059301	58,3563109
<i>winequality-red</i>	65,9162432	60,9112461	65,4473343	60,4715909
<i>balance</i>	92,16	85,76	92,16	86,72
<i>Media</i>	88,6287759	80,857701	88,6213659	80,3627756

5.2.- Estudiando los modelos FARC-HD global y FARC-HD local incluyendo cambios en los parámetros de búsqueda

En esta sección se estudia la efectividad del cambio de los parámetros propuestos en **3.2.- Mejorando efectividad del tuning local**. Al igual que en la sección anterior, primero se muestran los resultados obtenidos y luego se analiza su rendimiento. En primer lugar, compararemos las distintas configuraciones con FARC-HD global y luego las del FARC-HD local.

Uno de los cambios propuestos es ver la cantidad de ejemplos no cubiertos y en el caso de que fueran muchos, desarrollar un clasificador basado en centroides para estos casos. Tras varias ejecuciones con distintos datasets, llegamos a la misma conclusión. A la hora de entrenar, sí que quedaban ejemplos no cubiertos, pero a medida que avanzaba el algoritmo, la mejor solución obtenida siempre cubría todos los ejemplos. Por lo tanto, concluimos que no merece la pena avanzar en esta dirección.



Por otro lado, otro cambio propuesto se basa en crear un modelo que utilice el tuning local, pero sea semi-interpretable. Para ello necesitamos que el tuning local mejore el rendimiento del global, ya que, en el proceso de ~~votación~~, donde obtenemos la semi-interpretabilidad, estaríamos perdiendo información y por lo tanto obteniendo peores resultados. Tras los resultados obtenidos en **5.3.- Comparando los mejores resultados obtenidos con todos los demás**, también concluimos que no merece la pena avanzar en esta dirección.


5.2.1.- FARC-HD con tuning Global

Los resultados obtenidos para las distintas configuraciones probadas las tenemos en la Tabla 4. En dicha tabla en primer lugar comparamos en la parte izquierda la efectividad de la modificación en el parámetro L y luego comparamos la efectividad ~~de la suma~~ de iteraciones. Todo ello explicado anteriormente en **3.2.- Mejorando efectividad del tuning local**

Los resultados muestran que si comparamos g20-4 contra g20-2, obtenemos mejores resultados de media en g20-2. Esto significa que dándole más libertad de búsqueda al principio (L), el algoritmo cruza mejor los individuos y llega a mejores resultados.



Ahora puede darse el caso de que el algoritmo se nos quede corto con el número de iteraciones en la parte genética y por eso comparamos g20-2 contra gvar-2.

Sin embargo, los resultados muestran que dejando iterar más en la fase genética no llegamos a mejores resultados y que nos quedamos prácticamente con la misma efectividad. 


En ambos casos el operador estadístico *wilcoxon* muestra que no estamos ante una evidencia estadística suficiente como para  afirmar que estamos ante dos algoritmos completamente distintos.

Tabla 4. Resultados de todas las configuraciones de FARC-HD con tuning Global

g20-4		g20-2			g20-2		gvar-2	
Train	Test	Train	Test		Train	Test	Train	Test
86,981	86,189	86,849	86,623	banana	86,849	86,623	86,434	85,698
79,420	66,087	80,000	64,348	bupa	80,000	64,348	80,652	64,638
89,900	56,893	89,900	56,887	cleveland	89,900	56,887	90,321	56,215
92,188	82,138	92,114	84,232	ecoli	92,114	84,232	92,337	83,336
81,310	67,298	81,895	69,158	glass	81,895	69,158	81,895	69,158
81,208	74,495	81,372	71,549	haberman	81,372	71,549	81,372	71,549
98,861	88,893	99,003	91,167	ion	99,003	91,167	99,003	91,167
98,500	94,667	98,500	94,000	iris	98,500	94,000	98,500	94,000
84,253	80,965	84,227	80,388	magic	84,227	80,388	84,621	80,861
99,186	95,814	98,953	96,744	newthyroid	98,953	96,744	98,953	96,744
97,080	93,975	97,034	94,519	pageblocks	97,034	94,519	97,034	94,519
98,477	92,182	98,523	92,455	penbased	98,523	92,455	98,636	92,182
83,281	81,513	84,067	81,939	phoneme	84,067	81,939	84,239	82,050
83,821	74,085	83,594	75,383	pima	83,594	75,383	83,822	75,254
97,095	90,541	97,331	90,946	ring	97,331	90,946	97,534	91,216
85,575	81,022	85,342	80,406	satimage	85,342	80,406	86,276	81,022
97,149	96,966	97,310	97,241	shuttle	97,310	97,241	97,391	97,195
76,492	59,613	76,988	59,613	tae	76,988	59,613	76,988	59,613
79,066	78,873	79,066	78,873	titanic	79,066	78,873	79,066	78,873
97,804	90,270	97,500	90,405	twonorm	97,500	90,405	97,736	90,541
80,407	69,266	80,319	67,261	vehicle	80,319	67,261	81,737	68,796
100,000	96,048	100,000	96,619	wine	100,000	96,619	100,000	96,619
98,829	96,925	98,938	96,485	wisconsin	98,938	96,485	98,938	96,485
89,110	69,697	89,179	67,768	bands	89,179	67,768	89,452	68,046
93,352	78,637	93,351	77,128	spectfheart	93,351	77,128	93,445	76,373
98,594	96,660	98,594	96,837	wdbc	98,594	96,837	98,682	96,483
64,219	58,491	64,202	58,626	yeast	64,202	58,626	64,522	58,694
65,916	60,911	65,901	60,411	winequality-red	65,901	60,411	66,323	60,599
92,160	85,760	92,160	88,160	balance	92,160	88,160	92,160	88,160
88,629	80,858	88,697	80,902	Media	88,697	80,902	88,899	80,900

Wilcoxon	0,7916	Wilcoxon	0,9826
----------	--------	----------	--------

5.2.2.- FARC-HD con tuning Local

Los resultados obtenidos para las distintas configuraciones probadas las tenemos en la Tabla 5. En dicha tabla en primer lugar comparamos en la parte izquierda la efectividad de la modificación en el parámetro L y luego comparamos la efectividad de la suma de iteraciones. Todo ello explicado anteriormente en **3.2.-**

Mejorando efectividad del tuning local.

En este caso los resultados también muestran que si comparamos l20-4 contra l20-2 obtenemos mejores resultados en l20-2. Es decir, obtenemos mejores resultados cuando dejamos más libertad de búsqueda al principio. El Algoritmo cruza mejor los cromosomas y llega a mejores resultados.

Ahora puede darse el caso de que el algoritmo se nos quede corto con el número de iteraciones en la parte genética y por eso comparamos l20-2 contra lvar-2.

Observando los resultados vemos que efectivamente el tuning local llega a mejores resultados cuando se le deja iterar más en la fase genética del algoritmo. Este resultado demuestra que con el tuning local al tener mayor espacio de búsqueda aún podemos movernos más para llegar a mejores resultados.

El operador estadístico *wilcoxon* en este caso muestra que no tenemos base estadística suficiente para afirmar que estamos ante algoritmo distintos cuando comparamos l20-4 y l20-2. Sin embargo, cuando comparamos l20-2 contra lvar-2, si que tenemos bastante evidencia estadística como para afirmar que el número de iteraciones implica un cambio significativo en el rendimiento del algoritmo.



Tabla 5. Resultados de todas las configuraciones de FARC-HD con tuning Local

I20-4		I20-2			I20-2		Ivar-2	
Train	Test	Train	Test		Train	Test	Train	Test
86,764	85,887	86,363	85,755	banana	86,363	85,755	86,019	85,453
80,072	62,609	79,565	65,217	bupa	79,565	65,217	80,362	66,377
88,889	58,249	89,480	55,548	cleveland	89,480	55,548	89,564	55,548
92,187	81,848	91,592	80,061	ecoli	91,592	80,061	91,890	80,944
82,012	64,518	81,310	68,250	glass	81,310	68,250	81,310	68,250
81,862	71,227	81,290	73,184	haberman	81,290	73,184	81,290	73,184
98,647	88,889	99,074	90,596	ion	99,074	90,596	99,074	90,596
98,667	94,667	98,667	94,000	iris	98,667	94,000	98,667	94,000
84,503	81,281	84,582	81,018	magic	84,582	81,018	84,648	80,808
99,186	95,814	99,070	95,814	newthyroid	99,070	95,814	99,302	97,209
97,034	93,611	96,897	93,972	pageblocks	96,897	93,972	97,080	94,337
98,614	91,455	98,500	92,091	penbased	98,500	92,091	98,614	93,182
83,508	81,366	83,508	81,366	phoneme	83,508	81,366	83,508	81,366
83,789	75,124	83,593	75,775	pima	83,593	75,775	83,659	75,775
97,264	91,757	97,128	90,541	ring	97,128	90,541	97,196	90,676
85,497	79,935	85,964	79,157	satimage	85,964	79,157	86,353	79,939
96,586	96,276	96,977	96,828	shuttle	96,977	96,828	97,207	97,011
76,822	57,634	76,490	58,968	tae	76,490	58,968	76,656	58,968
79,066	78,873	79,066	78,873	titanic	79,066	78,873	79,066	78,873
97,669	90,000	97,838	89,865	twonorm	97,838	89,865	97,939	90,405
80,201	68,564	80,880	69,978	vehicle	80,880	69,978	82,240	70,094
100,000	93,810	100,000	97,730	wine	100,000	97,730	100,000	97,730
98,609	96,925	98,646	96,780	wisconsin	98,646	96,780	98,792	97,219
88,835	69,675	88,564	69,431	bands	88,564	69,431	88,770	68,586
92,884	79,022	92,604	79,762	spectfheart	92,604	79,762	93,258	79,378
98,638	95,959	98,550	95,956	wdbc	98,550	95,956	98,814	95,956
64,606	58,356	64,337	57,412	yeast	64,337	57,412	64,421	57,681
65,447	60,472	66,072	60,349	winequality-red	66,072	60,349	66,338	60,662
92,160	86,720	92,120	85,760	balance	92,120	85,760	92,120	85,760
88,621	80,363	88,577	80,691	Media	88,577	80,691	88,764	80,895
Wilcoxon		0,5164			Wilcoxon		0,0352	

5.3.- Comparando los mejores resultados obtenidos con todos los demás

Tras los resultados obtenidos en los apartados anteriores, ahora debemos comparar si podemos llegar a mejores resultados aplicando el tuning local en vez del tuning global. Para ello, mostramos los resultados de todas las ejecuciones en la Tabla 7.

Los resultados muestran que pese a conseguir mejorar la efectividad a la hora de aplicar el tuning local, no llegamos a mejorar el rendimiento del FARC-HD con el tuning global original. Todas las ejecuciones terminan obteniendo rendimientos parejos y al final la configuración que mejor rinde es g20-2, es decir tuning global con 20000 iteraciones y con la L modificada.

En la Tabla 6, comparamos estadísticamente todas las configuraciones mediante el operador wilcoxon. Si comparamos las configuraciones en local con las mismas configuraciones en global obtenemos los siguientes resultados:

1. Tenemos base estadística alta como para asegurar que con la configuración inicial de 20000 iteraciones y sin modificar la L estamos ante dos algoritmos que no generalizan igual. Aunque rinden muy parejos.
2. Comparando la configuración de 20000 iteraciones y la L modificada, podemos decir que aún 80% estamos ante algoritmo que no generalizan igual. Aunque rinden parejos en este caso el tuning global obtiene mejores resultados.
3. Tenemos base estadística alta como para asegurar que con la configuración de más iteraciones y L modificada estamos ante dos algoritmos que generalizan muy parecidamente.

Tabla 6. Comparacion wilcoxon entre todas las configuraciones utilizadas

WILCOXON						
	g20-4	g20-2	gvar-2	l20-4	l20-2	lvar-2
g20-4						
g20-2	0,7917					
gvar-2	0,6766	0,9826				
l20-4	0,0511	0,0651	0,0372			
l20-2	0,2109	0,2905	0,2205	0,5165		
lvar-2	0,8476	0,8689	0,7916	0,1161	0,0352	

Tabla 7. Todos los resultados obtenidos

	g20-2		g20-4		gvar-2		l20-2		l20-4		lvar-2	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
<i>banana</i>	86,849	86,623	86,981	86,189	86,434	85,698	86,363	85,755	86,764	85,887	86,019	85,453
<i>bupa</i>	80,000	64,348	79,420	66,087	80,652	64,638	79,565	65,217	80,072	62,609	80,362	66,377
<i>cleveland</i>	89,900	56,887	89,900	56,893	90,321	56,215	89,480	55,548	88,889	58,249	89,564	55,548
<i>ecoli</i>	92,114	84,232	92,188	82,138	92,337	83,336	91,592	80,061	92,187	81,848	91,890	80,944
<i>glass</i>	81,895	69,158	81,310	67,298	81,895	69,158	81,310	68,250	82,012	64,518	81,310	68,250
<i>haberman</i>	81,372	71,549	81,208	74,495	81,372	71,549	81,290	73,184	81,862	71,227	81,290	73,184
<i>ion</i>	99,003	91,167	98,861	88,893	99,003	91,167	99,074	90,596	98,647	88,889	99,074	90,596
<i>iris</i>	98,500	94,000	98,500	94,667	98,500	94,000	98,667	94,000	98,667	94,667	98,667	94,000
<i>magic</i>	84,227	80,388	84,253	80,965	84,621	80,861	84,582	81,018	84,503	81,281	84,648	80,808
<i>newthyroid</i>	98,953	96,744	99,186	95,814	98,953	96,744	99,070	95,814	99,186	95,814	99,302	97,209
<i>pageblocks</i>	97,034	94,519	97,080	93,975	97,034	94,519	96,897	93,972	97,034	93,611	97,080	94,337
<i>penbased</i>	98,523	92,455	98,477	92,182	98,636	92,182	98,500	92,091	98,614	91,455	98,614	93,182
<i>phoneme</i>	84,067	81,939	83,281	81,513	84,239	82,050	83,508	81,366	83,508	81,366	83,508	81,366
<i>pima</i>	83,594	75,383	83,821	74,085	83,822	75,254	83,593	75,775	83,789	75,124	83,659	75,775
<i>ring</i>	97,331	90,946	97,095	90,541	97,534	91,216	97,128	90,541	97,264	91,757	97,196	90,676
<i>satimage</i>	85,342	80,406	85,575	81,022	86,276	81,022	85,964	79,157	85,497	79,935	86,353	79,939
<i>shuttle</i>	97,310	97,241	97,149	96,966	97,391	97,195	96,977	96,828	96,586	96,276	97,207	97,011
<i>tae</i>	76,988	59,613	76,492	59,613	76,988	59,613	76,490	58,968	76,822	57,634	76,656	58,968
<i>titanic</i>	79,066	78,873	79,066	78,873	79,066	78,873	79,066	78,873	79,066	78,873	79,066	78,873
<i>twonorm</i>	97,500	90,405	97,804	90,270	97,736	90,541	97,838	89,865	97,669	90,000	97,939	90,405
<i>vehicle</i>	80,319	67,261	80,407	69,266	81,737	68,796	80,880	69,978	80,201	68,564	82,240	70,094
<i>wine</i>	100,000	96,619	100,000	96,048	100,000	96,619	100,000	97,730	100,000	93,810	100,000	97,730
<i>wisconsin</i>	98,938	96,485	98,829	96,925	98,938	96,485	98,646	96,780	98,609	96,925	98,792	97,219
<i>bands</i>	89,179	67,768	89,110	69,697	89,452	68,046	88,564	69,431	88,835	69,675	88,770	68,586
<i>spectfheart</i>	93,351	77,128	93,352	78,637	93,445	76,373	92,604	79,762	92,884	79,022	93,258	79,378
<i>wdbc</i>	98,594	96,837	98,594	96,660	98,682	96,483	98,550	95,956	98,638	95,959	98,814	95,956
<i>yeast</i>	64,202	58,626	64,219	58,491	64,522	58,694	64,337	57,412	64,606	58,356	64,421	57,681
<i>winequality-red</i>	65,901	60,411	65,916	60,911	66,323	60,599	66,072	60,349	65,447	60,472	66,338	60,662
<i>balance</i>	92,160	88,160	92,160	85,760	92,160	88,160	92,120	85,760	92,160	86,720	92,120	85,760
<i>Media</i>	88,697	80,902	88,629	80,858	88,899	80,900	88,577	80,691	88,621	80,363	88,764	80,895

6.- Conclusiones

En el proyecto se ha estudiado el tuning local, junto a los distintos factores que pueden influir en su rendimiento. En dichos factores hemos estudiado la configuración de parámetros que afectaban directamente en la modificación realizada y algunos posibles casos de aprendizaje no óptimo.

Entre los casos no óptimos hemos estudiado el número de ejemplos no cubiertos, por si merecía la pena introducir un clasificador basado en centroides en vez de utilizar el sistema de clasificación basado en la clase con más ejemplos. Los resultados han mostrado que en las soluciones obtenidas no quedaban ejemplos no cubiertos por lo tanto esta rama del proyecto ha terminado sin la implementación de ningún clasificador basado en centroides.

Por otro lado, se ha estudiado la amplitud del rango de búsqueda mediante la configuración de parámetros. Los resultados se han analizado en la sección 5 y al final se concluye que con el tuning local no conseguimos mejorar el rendimiento del FARC-HD con tuning global. Después de probar distintas configuraciones se ha obtenido un rendimiento casi idéntico, pero el tuning local requiere más tiempo de ejecución y en conclusión no merece la pena para terminar a un rendimiento tan parejo.

Por último, se ha considerado implementar un modelo semi-interpretable en base a cambiar la inferencia en el algoritmo. Este proceso implica tener que obtener mejores resultados de forma local que de forma global, ya que en el cambio de inferencia perdemos información y por lo tanto ~~peor~~. Tras los resultados obtenidos con el tuning local, al ver que éste no mejoraba el tuning global, deja de tener sentido seguir explorando esta rama. Por lo tanto, esta vía también queda desechada.

7.- Líneas Futuras



En esta sección, proponemos distintas ideas para tratar de mejorar la propuesta actual:

El tuning local puede aportar mejoras si ampliamos el espacio de búsqueda. Con este motivo en mente, podríamos introducir el tuning de amplitud junto al tuning Lateral y observar si dando más dimensionalidad al espacio de búsqueda obtenemos mejores resultados ajustándonos localmente a las etiquetas.

Bibliografía

- [1] J. Alcalá-Fdez, R. Alcalá y F. Herrera, «A fuzzy association rule-based classification model for high-dimensional problems with genetic rule selection and lateral tuning», IEEE Transactions on Fuzzy Systems, vol. 19, nº 5, pp. 857-872, 2011.
- [2] L. A. Zadeh, «Fuzzy Sets», Information and Control, vol. 8, nº 3, pp. 338-353, 1965
- [3] O. Cordón, M. del Jesus y F. Herrera, «A proposal on reasoning methods in fuzzy rulebased classification systems», International Journal of Approximate Reasoning, vol. 20, nº 1, pp. 21-45, 1999
- [4] M. Mitchell, Introduction to Genetic Algorithms, 1998.
- [5] H. Ishibuchi y T. Yamamoto, «Rule weight specification in fuzzy rule-based classification systems», IEEE Transactions on Fuzzy Systems, vol. 13, nº 4, pp. 428- 435, 2005.
- [6] D. E. Goldberg, Genetic algorithms in search, optimization and machine learning, 1989.
- [7] B. Kavsek and N. Lavrac, «Apriori-sd: Adapting association rule learning to subgroup discovery», Applied Artificial Intelligence, vol. 20, no. 7, pp. 543–583, 2006.
- [8] L. Eshelman, «The chc adaptative search algorithm: How to have safe search when engaging in nontraditional genetic recombination», de Foundations of Genetic Algorithms, G. Rawling, Ed. Morgan Kaufmann, 1991, pp. 265-283.
- [9] R. Alcalá, J. Alcalá-Fdez, and F. Herrera, «A proposal for the genetic lateral tuning of linguistic fuzzy systems and its interaction with rule selection», IEEE Transactions on Fuzzy Systems, vol. 15, no. 4, pp. 616– 635, 2007.
- [10] O. Cordón, M. del Jesus y F. Herrera, «A proposal on reasoning methods in fuzzy rulebased classification systems», International Journal of Approximate Reasoning, vol. 20, nº 1, pp. 21-45, 1999.
- [11] L. Eshelman y J. Schaffer, «Real-coded genetic algorithms and interval schemata», de Foundations of Genetic Algorithms, vol. 2, D. Whitley, Ed. Morgan Kaufmann, 1993, pp. 187-202.
- [12] M. Lozano, F. Herrera, N. Krasnogor y D. Molina, «Real-coded memetic algorithms with crossover hill-climbing», Evolutionary Computation, vol. 12, nº 3, pp. 273-302, 2004
- [13] [ASOC](#)
- [14] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez y F. Herrera, «KEEL data-mining software tool: Data set repository, integration of

- algorithms and experimental analysis framework,» *Journal of Multiple-Valued Logic and Soft Computing*, vol. 17, no, 2-3, pp. 255-287, 2011
- [15] T. Wong, P. Yeh, Reliable accuracy estimates from k-fold cross validation, *IEEE Transactions on Knowledge and Data Engineering* 32 (8) (2020) 1586– 805 1594.
- [16] J. Alcala-Fdez, L. Sánchez, S. García, M. del Jesus, S. Ventura, J. Garrell, J. Otero, C. Romero, J. Bacardit, V. Rivas, J. Fernandez, and ´ F. Herrera, “KEEL: A software tool to assess evolutionary algorithms to data mining problems,” *Soft Computing*, vol. 13, no. 3, pp. 307–318, 2009
- [17] F. Wilcoxon, «Individual comparisons by ranking methods,» *Biometrics*, vol. 1, nº 6, pp. 80-83, 1945.