



Universidad Simón Bolívar  
Decanato de Estudios Profesionales  
Coordinación de Ingeniería de la Computación

@títuloProyecto

Por:  
Alejandro Flores V.

Realizado con la asesoría de:  
Emely Arraiz B.

PROYECTO DE GRADO  
Presentado ante la Ilustre Universidad Simón Bolívar  
como requisito parcial para optar al título de  
Ingeniero de Computación

Sartenejas, septiembre de 2014



UNIVERSIDAD SIMÓN BOLÍVAR  
DECANATO DE ESTUDIOS PROFESIONALES  
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

ACTA FINAL PROYECTO DE GRADO

@TÍTULO PROYECTO

Presentado por:

**ALEJANDRO FLORES V.**

Este Proyecto de Grado ha sido aprobado por el siguiente jurado examinador:

---

Emely Arráiz B.

---

@jurado1

---

@jurado2

Sartenejas, @día de @mes de @año

## **Resumen**

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

**Palabras clave:** @palabra1, @palabra2, @palabra3.

# Agradecimientos

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

# Índice general

<b>Resumen</b>	<b>I</b>
<b>Agradecimientos</b>	<b>II</b>
<b>Índice de Figuras</b>	<b>V</b>
<b>Lista de Tablas</b>	<b>VI</b>
<b>Índice de algoritmos</b>	<b>VII</b>
<b>Acrónimos y Símbolos</b>	<b>VIII</b>
<b>Introducción</b>	<b>1</b>
<b>1. Selección de Instancias</b>	<b>2</b>
1.1. Reducción de Datos . . . . .	2
1.2. Selección de Instancias . . . . .	4
1.2.1. Regla del Vecino Más Cercano (NN) . . . . .	5
1.2.2. Definiciones relevantes . . . . .	7
1.3. Algorítmos de aproximación para Selección de Instancias . . . . .	7
1.3.1. Métodos basados en la regla NN . . . . .	7
1.3.2. Métodos basados en eliminación ordenada . . . . .	8
1.3.3. Métodos basados en muestreo aleatorio . . . . .	9
1.3.4. Métodos basados en metaheurísticas . . . . .	9
1.3.5. Criterios de comparación . . . . .	10
<b>2. Metaheurísticas</b>	<b>11</b>
2.1. Descripción general . . . . .	11
2.2. Metaheurísticas inspiradas en la naturaleza . . . . .	12
2.2.1. Algoritmos Evolutivos . . . . .	12
2.2.1.1. Generational Genetic Algorithm (GGA) . . . . .	13
2.2.1.2. Steady-State Genetic Algorithm (SGA) . . . . .	14
2.2.1.3. CHC Adaptive Search Algorithm . . . . .	14
2.2.1.4. Population-Based Incremental Learning (PBIL) . . . . .	15
2.2.2. Inteligencia de Enjambre . . . . .	17

2.2.2.1. Particle Swarm Optimization (PSO) . . . . .	17
<b>3. Adaptación al problema de Selección de Instancias</b>	<b>20</b>
3.1. Consideraciones generales . . . . .	20
3.1.1. Representación . . . . .	20
3.1.2. Función de evaluación . . . . .	21
3.1.3. Generación de soluciones iniciales . . . . .	22
3.1.3.1. Condensed Nearest Neighbor . . . . .	24
3.1.3.2. Nearest Enemy Hypersphere Selection . . . . .	24
3.1.3.3. Closest Nearest Enemy . . . . .	26
3.1.3.4. Farthest Nearest Enemy . . . . .	26
3.2. Modificaciones particulares . . . . .	26
3.2.1. Adaptación de GGA, SGA y CHC . . . . .	26
3.2.2. Adaptación de PBIL . . . . .	27
3.2.3. Adaptación de PSO . . . . .	27
<b>4. Evaluación Experimental</b>	<b>29</b>
4.1. Diseño Experimental . . . . .	29
4.1.1. Conjuntos de datos . . . . .	30
4.1.2. Particiones y ejecuciones . . . . .	31
4.1.3. Parámetros . . . . .	31
4.2. Resultados . . . . .	32
<b>Conclusiones y Recomendaciones</b>	<b>33</b>

# Índice de figuras

1.1.	Diagramas de Voronoi y NN . . . . .	6
3.1.	Algoritmos de Selección de Instancias . . . . .	23
3.2.	Selección obtenida por NEHS . . . . .	25

# Índice de Tablas

4.1.	Conjuntos de datos pequeños . . . . .	30
4.2.	Conjuntos de datos medianos . . . . .	30
4.3.	Conjuntos de datos grandes . . . . .	30
4.4.	Parámetros usados en cada metaheurística . . . . .	32

# Índice de algoritmos

2.1.	Generational Genetic Algorithm . . . . .	14
2.2.	Steady-State Genetic Algorithm . . . . .	15
2.3.	CHC Adaptive Search Algorithm . . . . .	16
2.4.	Population-Based Incremental Learning . . . . .	17
2.5.	Particle Swarm Optimization . . . . .	18
3.1.	Generador de vector de probabilidades inicial . . . . .	23
3.2.	Condensed Nearest Neighbor . . . . .	24
3.3.	Nearest Enemy Hypersphere Selection . . . . .	25
3.4.	Population-Based PSO . . . . .	28

# Acrónimos y Símbolos

**KDD** Knowledge Discovery in Databases

**MD** Minería de Datos

**SI** Selección de Instancias

**NN** Nearest Neighbor

---

$\in$  Relación de pertenencia, «*es un elemento de*»

## ***Dedicatoria***

A @personasImportantes, por @razonesDedicatoria.

# Introducción

El avance de la ciencia y la tecnología durante las últimas décadas ha traído como consecuencia un aumento sin precedentes en la cantidad de datos generados y recopilados por la actividad humana. El *Proyecto Genoma Humano*, el *Instituto SETI* y el *Gran Colisionador de Hadrones*, tienen algo en común: generan una enorme cantidad de datos, por lo que resulta imposible usarlos y mucho menos analizarlos de forma tradicional.

Por esta razón, nuevos cambios de estudio, como el Descubrimiento de Conocimiento en Bases de Datos (*KDD*) y Minería de Datos (*DM*), emergen para afrontar el creciente problema que se genera al intentar usar y analizar enormes cantidades de datos.

Bajar complejidad, disminuir los datos.

# Capítulo 1

## Selección de Instancias

Este capítulo describe el proceso de reducción de datos y sus diferentes estrategias. En particular, se hace especial énfasis en el problema de *Selección de Instancias*: se define formalmente, se describen sus principales características, y se realiza un breve análisis del estado del arte.

### 1.1. Reducción de Datos

Como parte del proceso de “*Knowledge Discovery in Databases*” (*KDD*), la fase de *Preprocesamiento de los Datos* juega un rol fundamental para la aplicación efectiva de técnicas de *Minería de Datos* (*MD*). Una de las estrategias de mayor uso durante la fase de preprocesamiento es la de *Reducción de Datos*.

El problema de *Reducción de Datos* consiste en decidir qué datos deben ser utilizados durante la aplicación de algoritmos de *MD* con el objetivo de construir modelos representativos de los datos originales. Dicha decisión debe basarse en la relevancia de los datos con respecto a los objetivos que se persiguen, o inclusive, por limitaciones técnicas. En términos prácticos, la importancia del problema de *Reducción de Datos* radica en los siguientes factores: *a) Tiempo y Espacio*: Mientras mayor sea el número de datos a utilizar, mayor será el espacio necesario para almacenarlos y el tiempo requerido para analizarlos. *b) Sensibilidad al ruido*: Al aumentar el número de instancias en el conjunto de datos, también lo hace la probabilidad de aparición de datos atípicos, inconsistentes o redundantes. Su

eliminación se vuelve necesaria para evitar un impacto negativo en los modelos de representación creados a partir de los datos.

En función de estos criterios, y basados en la definición de los datos, se han formulado diferentes estrategias para llevar a cabo la fase de reducción. En los procesos de *KDD*, el conjunto de datos está definido en función de un conjunto de clases  $\Omega$  y un conjunto  $T$  de  $n$  observaciones de un evento, cada observación con  $m$  mediciones, donde:

**Definición 1.** Una **instancia**  $t_i$  (con  $i = 1 \dots n$ ) es una observación del evento; donde  $t_i = (v_{i,1}, v_{i,2}, \dots, v_{i,m})$  es una tupla de  $m$  valores/mediciones (un punto en un espacio  $m$ -dimensional). Adicionalmente, cada instancia en  $t_i$  pertenece a la clase  $\omega_{t_i} \in \Omega$ .

**Definición 2.** Un **atributo**  $p_j$  (con  $j = 1 \dots m$ ) define el conjunto de mediciones «*de un mismo tipo*» para todas las observaciones, *i.e.*  $p_j = \{v_{i,j} \mid i = 1 \dots n\}$ . Cada atributo puede presentarse en diferentes formatos: *nominales*, *discretos*, o *continuos*.

A continuación se presentan las estrategias de *Reducción de Datos* más estudiadas en la literatura:

- **Selección de Instancias** [BL97, LM02]

Busca la reducción del conjunto de datos mediante la selección de un subconjunto de instancias, de forma tal que dicho subconjunto conserve las capacidades de representación del conjunto original.

La sección 1.2 está dedicada a describir esta estrategia en amplitud.

- **Selección de Atributos** [BL97, LM98]

Esta técnica permite eliminar atributos del conjunto de datos original, que no contribuyen (o que influyen negativamente) a la construcción de un modelo representativo.

- **Discretización de Atributos** [FI93, LHTD02]

Esta estrategia busca convertir atributos *continuos* en *discretos* (cuantificando el espacio de posibles valores), o disminuir el número de valores *discretos* (combinando valores adyacentes).

## 1.2. Selección de Instancias

Dado un conjunto inicial de instancias  $T = \{t_i \mid i = 1 \dots n\}$  donde  $t_i = (v_{i,1}, v_{i,2}, \dots, v_{i,m})$  y  $\omega_i \in \Omega$  (siendo  $\Omega$  el conjunto de posibles clases para las instancias en  $T$ ), el problema de *Selección de Instancias (SI)* consiste en seleccionar un  $R \subseteq T$  que mantenga (o mejore) la capacidad de representación del conjunto original  $T$ .

Más aún, este problema puede ser formulado como un *problema de optimización*, donde se busca el  $R^* \subseteq T$  de menor cardinalidad, que mantenga (o mejore) la capacidad de representación del conjunto original.

En particular, la literatura se ha enfocado en la aplicación del problema de *SI* para su uso en clasificadores [GK14, Tou02]. El subconjunto seleccionado se usa como conjunto de entrenamiento, en base al cuál el clasificador estima la clase  $\hat{\omega}$  de instancias previamente desconocidas. En este sentido, el problema de optimización de *SI* busca conseguir un  $R^* \subseteq T$  *consistente* y de cardinalidad mínima, donde:

**Definición 3.** Un conjunto  $R$  es **consistente** con  $T$ , si y solo si toda instancia  $t \in T$  es clasificada correctamente (*e.i.*  $\hat{\omega}_t = \omega_t$ ) mediante el uso de un clasificador  $M$  y las instancias en  $R$  como conjunto de entrenamiento.

La complejidad del problema de selección ha sido estudiada por diferentes autores: *Bien* y *Tibshirani* [BT12] describen la reducción del problema de *SI* al problema de *Conjunto de Cobertura* (“*Set Cover*” en inglés), cuya versión de optimización es NP-Dura. Más aún, *Wilfong* [Wil91] y *Zukhba* [Zuk10] muestran que el problema de selección es NP-Duro.

En general, la literatura relacionada con el problema de *SI* se ha enfocado en el uso de clasificadores  $k$ -NN por su simplicidad, y sobretodo, por su capacidad de representación de modelos sin información adicional sobre la distribución de los datos. El caso particular del problema de *SI* para su uso con clasificadores  $k$ -NN también es conocido como *Selección de Prototipos (SP)*. A continuación se describen los clasificadores NN.

### 1.2.1. Regla del Vecino Más Cercano (NN)

Inicialmente descrita por *Fix* y *Hodges* [FH51], la regla del *Vecino Más Cercano* (“*Nearest Neighbor*”, *NN*) es una regla de inferencia basada en la idea de que instancias con atributos similares (cercanas en un espacio de  $m$  dimensiones) tienden a compartir la misma clase. La regla NN estima la clase  $\hat{\omega}_x$  de un punto  $x$  en un espacio  $m$ -dimensional, dado un conjunto  $T$  de instancias de entrenamiento y una función de distancia  $\varphi$  entre dos puntos en dicho espacio:

$$\hat{\omega}_x = \omega_{t^*}, \quad t^* = \arg \min_{t \in T} \varphi(t, x) \quad (1.1)$$

La generalización de la regla de inferencia NN se conoce como el clasificador  $k$ -NN: dado un  $k \in \mathbb{N}$ , se estima la clase  $\hat{\omega}_x$  de un punto  $x$  en función a la clase de las  $k$  instancias más cercanas a  $x$ . En general, se usa la estrategia del «*voto de la mayoría*», asignando la clase más común entre las  $k$  instancias más cercanas. En particular, el clasificador 1-NN corresponde a la regla NN.

$k$ -NN es un clasificador no paramétrico de *aprendizaje perezoso* (debido a que la etapa de aprendizaje consiste en guardar el conjunto de entrenamiento), caracterizado por su sencillez en términos de implementación. Esa simplicidad y su probada utilidad para numerosas aplicaciones, han hecho del clasificador  $k$ -NN uno de los más estudiados en la literatura.

Uno de los trabajos de mayor relevancia es el de *Cover* y *Hart* [CH67], quienes mostraron que cuando el número de instancias de entrenamiento tiende a infinito, el clasificador  $k$ -NN garantiza un error no mayor al doble de la tasa de error de Bayes: la menor tasa de error posible para un clasificador dado. Adicionalmente, probaron que para un conjunto de entrenamiento de cardinalidad finita, el clasificador 1-NN es admisible dentro de la clase de clasificadores  $k$ -NN: *e.i.* No existe  $k > 1$  tal que  $k$ -NN tenga menor probabilidad de error frente a 1-NN, para toda posible distribución de los datos.

Adicionalmente, algunos trabajos en geometría computacional han contribuido significativamente en la comprensión del problema. En este sentido, el clasificador 1-NN para espacios euclidianos puede definirse de forma alternativa en función de *Diagramas de Voronoi* [Vor08]: una partición del espacio  $\mathbb{R}^m$  en *Celdas de Voronoi*, cada una definida por una instancia  $t \in T$  donde  $t$  es el *vecino más cercano* para

todos los puntos dentro del espacio dentro de dicha celda (ver Figura 1.1). Esto ha permitido el desarrollo de nuevos enfoques para la búsqueda de vecinos más cercanos basados en *Diagramas de Voronoi*, como el descrito por *Kolahdouzan* y *Shahabi* [KS04].

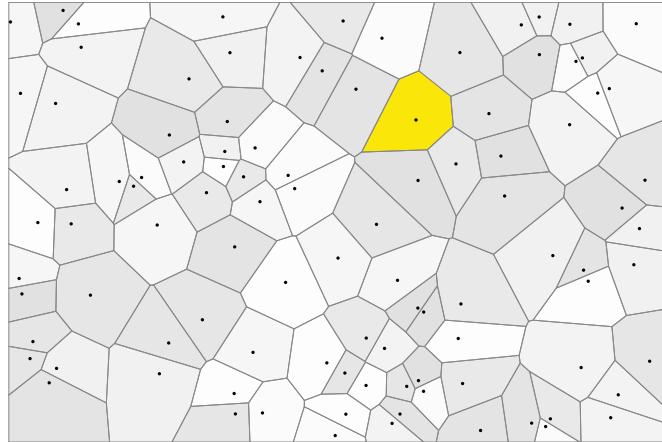


FIGURA 1.1: Diagrama de Voronoi para instancias en un espacio  $\mathbb{R}^2$ . En amarillo la Celda de Voronoi de un punto  $t \in T$ , representando el espacio de puntos para los que  $t$  es su *vecino más cercano*.

Similarmente, esta relación ha permitido avances importantes en términos de complejidad. En particular, mediante el uso de *kd-trees* [Ben75] (árboles de búsqueda binaria en múltiples dimensiones) se ha logrado disminuir la complejidad en tiempo de clasificación, de  $\mathcal{O}(n)$  (de un enfoque “ingenuo” revisando todas las instancias) a  $\mathcal{O}(\log n)$ , a costas de un aumento en el tiempo necesario para el entrenamiento del clasificador: de  $\mathcal{O}(1)$  a  $\mathcal{O}(n \log n)$ , el tiempo necesario para la construcción del árbol.

Sin embargo, los clasificadores *k*-NN presentan ciertas propiedades desalentadoras; el problema de conseguir el vecino más cercano de un punto dado, requiere –en cualquiera de los casos– almacenar todas las instancias de entrenamiento: *e.i.*  $\mathcal{O}(n)$  en espacio. Adicionalmente, trabajos más recientes [KL04] muestran que en espacios euclidianos de altas dimensiones, la búsqueda del vecino más cercano requiere  $\mathcal{O}(n)$  en tiempo: un fenómeno conocido como la «*maldición de la dimensionalidad*» (“*curse of dimensionality*” en inglés). Finalmente, según *Shwartz* y *David* [SSBD14] los clasificadores NN tienden a sobre-ajustar el modelo con respecto al conjunto de entrenamiento (*overfitting* en inglés); efecto que puede mitigarse aumentando el *k* del clasificador [DGKL94, SSBD14] y eliminando instancias del conjunto de datos [GKK13].

### 1.2.2. Definiciones relevantes

A continuación se definen algunos conceptos relevantes para la descripción de métodos de selección de instancias. Dado un conjunto de instancias  $Q \subseteq T$ :

**Definición 4.** Los **asociados** en  $Q$  de una instancia  $t$  son aquellas instancias en  $Q$  para las cuales  $t$  pertenece a su conjunto de  $k$  instancias más cercanas:

$$\text{asociados}_Q(t) = \{q \in Q \mid t \in kNN(q)\} \quad (1.2)$$

**Definición 5.** Los **enemigos** en  $Q$  de una instancia  $t \in T$  son aquellas instancias en  $Q$  con una *clase* diferente a la *clase* de  $t$ :

$$\text{enemigos}_Q(t) = \{q \in Q \mid \omega_q \neq \omega_t\} \quad (1.3)$$

**Definición 6.** El **enemigo más cercano** (NE, *nearest enemy*) en  $Q$  de una instancia  $t \in T$  —denotada como  $\text{NE}_Q(t)$ —, es la instancia más cercana a  $t$  con diferente clase (*i.e.* la instancia más cercana a  $t$  perteneciente a  $\text{enemigos}_Q(t)$ ):

$$\text{NE}_Q(t) = \arg \min_{e \in \text{enemigos}_Q(t)} \varphi(t, e) \quad (1.4)$$

## 1.3. Algoritmos de aproximación para Selección de Instancias

Debido a la complejidad del problema de *SI*, la literatura se ha enfocado en la definición de heurísticas para conseguir soluciones aproximadas. De nuevo, el uso de clasificadores  $k$ -NN es una práctica extendida a lo largo de estos trabajos, por lo que las características de la regla NN han servido para el desarrollo de muchos métodos de selección.

### 1.3.1. Métodos basados en la regla NN

- *Condensed Nearest Neighbor* (CNN) [Har68]

Inicialmente el conjunto  $R$  se inicializa con una instancia cualquiera. Luego se itera sobre cada instancia  $t \in T$ ; si  $t$  no es clasificada correctamente

usando  $R$ ,  $t$  se agrega a  $R$ . CNN consigue un conjunto consistente, reduciendo considerablemente el conjunto de datos original. Sin embargo, no asegura un conjunto consistente mínimo, pues depende del orden en el que son revisadas las instancias en  $T$ .

- *Edited Nearest Neighbor* (ENN) [Wil72]

Comienza con  $R = T$ . Luego itera sobre las instancias en  $R$ ; aquellas que no sean bien clasificadas usando  $R$  son eliminadas. Tiende a eliminar instancias ruidosas o cercanas a los bordes de decisión. Sin embargo, depende del orden en que itera sobre las instancias, y presenta bajas tasas de reducción dado que mantiene puntos internos.

- *Repeated Edited Nearest Neighbor* (RENN) [Wil72]

Aplica ENN al conjunto de datos  $R$  (inicialmente  $R = T$ ) hasta que no ocurran cambios en  $R$ . Amplía la distancia entre clases y “suaviza” los bordes de decisión.

- *Reduced Nearest Neighbor* (RNN) [Gat72]

RNN extiende a CNN, usándola como solución inicial  $R = R_{CNN}$ . Luego, itera sobre cada instancia  $t \in R$ : si todas las instancias en  $T$  son correctamente clasificadas usando  $R \setminus \{t\}$ , se elimina  $t$  de  $R$ . En caso contrario, se mantiene  $R$  y continua la iteración. La precisión de RNN puede mejorar respecto a CNN, pero es más costoso y su consistencia depende de la consistencia del conjunto resultante de CNN y del orden en que se iteren las instancias en  $R$ .

### 1.3.2. Métodos basados en eliminación ordenada

- *Decremental Reduction Optimization Procedure 1* (DROP1) [WM97]

Comienza con una solución inicial  $R = T$ . Itera sobre cada instancia  $t \in R$ : si todos sus *asociados* en  $R$  son correctamente clasificados con  $R \setminus \{t\}$ ,  $t$  se elimina de  $R$ . Reduce considerablemente el conjunto de datos inicial, pero obtiene baja precisión de clasificación, y el subconjunto resultante depende del orden en que se iteró sobre  $T$ .

- *Decremental Reduction Optimization Procedure 2* (DROP2) [WM97]

Es una mejora sobre DROP1 en la cuál se elimina una instancia  $t$  cuando todos sus *asociados* en  $T$  son clasificadas correctamente usando  $R \setminus \{t\}$ . Además, DROP2 ordena las instancias con respecto a la distancia de su

*enemigo* más cercano, en un intento de eliminar primero instancias centrales, y luego los puntos en los bordes de decisión.

- *Decremental Reduction Optimization Procedure 3* (DROP3) [WM97]

Dado que el orden en que se iteran las instancias en DROP2 se ve alterado por puntos ruidosos, DROP3 filtra instancias ruidosas antes de ordenar el conjunto de entrenamiento.

### 1.3.3. Métodos basados en muestreo aleatorio

- *Random Mutation Hill Climbing* (RMHC) [Ska94]

Se selecciona un subconjunto de instancias aleatorias  $R$  de tamaño fijo. En cada iteración el algoritmo intercambia una instancia en  $R$  por una en  $T \setminus R$ ; si el cambio mejora la precisión, se mantiene, en caso contrario se deshace.

### 1.3.4. Métodos basados en metaheurísticas

Las metaheurísticas son métodos de búsqueda estocástica de propósito general, usadas para encontrar soluciones óptimas o casi óptimas a problemas de optimización combinatoria. Por esta razón, muchos trabajos se han enfocado en el uso de estas técnicas para conseguir soluciones al problema de *SI*.

Algunos de los primeros trabajos se enfocaron en adaptar el algoritmo de *Búsqueda Tabú* para solucionar el problema de *SI*. En particular, los estudios de *Cerverón et al.* [CF01] y *Zhang et al.* [ZS02] describen dos enfoques diferentes de modificación del algoritmo.

Sin embargo, la mayoría de los estudios se han enfocado en el uso de *Algoritmos Evolutivos* (*AE*), adaptándolos para la búsqueda de soluciones al problema de selección. Entre ellos destaca el trabajo realizado por *Cano et al.* [CHL03]; un completo estudio comparativo entre algoritmos “tradicionales” de *SI* y adaptaciones de *Generational Genetic Algorithm* (GGA), *Steady-State Genetic Algorithm* (SGA), *CHC Adaptive Search Algorithm* (CHC) y *Population-Based Incremental Learning* (PBIL). Con este estudio, resulta evidente la utilidad de los *AE* frente a los algoritmos tradicionales de *SI* en función de la capacidad de reducción y precisión de los conjuntos seleccionados.

Existen también otras adaptaciones y modificaciones sobre *AE*, entre los que destacan: *Estimation of Distribution Algorithm* (EDA) [SLI<sup>+</sup>01], *Intelligent Genetic Algorithm* (IGA) [HLL02], *Steady-State Memetic Algorithm* (SSMA) [GCH08] y *Genetic Algorithm* [GPY08] basado en Error Cuadrático Medio, *Clustered Crossover* y *Fast Smart Mutation* (GA-MSE-CC-PSM).

### 1.3.5. Criterios de comparación

Para comparar métodos de *SI* se consideran una serie de criterios usados para evaluar las ventajas y desventajas de cada algoritmo. A continuación se describen los factores más relevantes:

- *Reducción*: El objetivo principal de métodos de *SI* es el de reducir el número de instancias del conjunto de datos. Esto no solo disminuye el espacio necesario para almacenar los datos, sino que acelera el proceso de clasificación.
- *Precisión*: Un algoritmo exitoso debe reducir el conjunto de datos, afectando en la menor medida posible su capacidad de generalización.
- *Tiempo*: A pesar de que el proceso de preprocesamiento y aprendizaje debe realizarse solo una vez, la complejidad de los algoritmos pueden volverlos poco prácticos para su uso sobre conjuntos de datos “grandes”.

# Capítulo 2

## Metaheurísticas

### 2.1. Descripción general

Las metaheurísticas son métodos estocásticos de búsqueda de propósito general sobre espacios combinatorios. Son usados generalmente para tratar problemas de optimización combinatoria, donde su complejidad hace imposible evaluar todas las soluciones factibles en un tiempo razonable. Estos algoritmos son capaces de conseguir “buenas” soluciones a un problema en un período de tiempo mucho menor. Sin embargo, para muchos problemas la complejidad de estos algoritmos sigue siendo un factor prohibitivo, debido al uso de funciones “costosas” para la evaluación de soluciones intermedias (*fitness*).

La idea es desarrollar algoritmos que recorran solo una fracción del espacio de soluciones, y que sean capaces de encontrar soluciones óptimas o casi óptimas al problema en cuestión. Para lograrlo, las metaheurísticas combinan procesos de *diversificación* e *intensificación* (o *exploración* y *explotación* respectivamente) [Yan08]. La fase de *diversificación* implica la generación de soluciones diversas con el objeto de explorar el espacio de búsqueda, mientras que la fase de *intensificación* se refiere al mejoramiento de soluciones (conseguir óptimos locales) mediante el uso de métodos de búsqueda local. La selección de las mejores soluciones asegura la convergencia a soluciones óptimas, mientras que la exploración aleatoria de soluciones evita que el algoritmo quede “atrapado” en óptimos locales. La combinación en el uso de ambos procesos hace posible conseguir buenas soluciones al problema, sin la necesidad de recorrer el espacio de búsqueda completo.

Cada metaheurística está caracterizada por las estrategias que usa para cada fase, así como el orden y la frecuencia en que las aplica. Esto permite clasificarlas en función de su similitud. En este sentido, a continuación se describe un conjunto de metaheurísticas caracterizadas por tener a la naturaleza como fuente de inspiración.

## 2.2. Metaheurísticas inspiradas en la naturaleza

La habilidad de la naturaleza para moldear soluciones a situaciones complejas mediante procesos y reglas caracterizadas por su simplicidad, la ha convertido en una fuente inagotable de inspiración para el desarrollo de algoritmos de optimización. Estos algoritmos a menudo presentan buen desempeño para aproximar soluciones a todo tipo de problemas, dado que no requieren información sobre la distribución del espacio de búsqueda. Por esta razón, existe una amplia literatura sobre enfoques bio-inspirados [BS12] para resolver gran variedad de problemas en diversas áreas de computación.

En particular, los enfoques más comunes en la literatura sobre metaheurísticas inspiradas en la naturaleza se apoyan en *a)* la evolución de poblaciones (Algoritmos Evolutivos) y *b)* el comportamiento colectivo (Inteligencia de Enjambre).

### 2.2.1. Algoritmos Evolutivos

Los Algoritmos Evolutivos (*AE*) son metaheurísticas basadas en procesos de evolución biológica con el objetivo de explorar en amplitud espacios de solución con distribución desconocida. Con el fin de replicar los procesos evolutivos, los *AE* mantienen un conjunto de soluciones candidatas al problema (una *población* de *cromosomas/individuos*), que modifican iterativamente apoyándose en el uso de operadores de *mutación*, *recombinación* y/o *selección*.

Los *AE* codifican cada cromosoma como una cadena de genes de tamaño *l* (análogo a la estructura del ADN), donde cada gen representa una parte de la solución al problema en cuestión. A partir de esta representación, los *AE* definen un conjunto de operadores que cumplen la función de las estrategias de *exploración* y *explotación*:

- *Mutación*: Modifica los genes de soluciones intermedias con la finalidad de explorar el espacio de soluciones e introducir nueva información a la población. Simula la variabilidad en las poblaciones, fenómeno clave para la aparición de nuevos genes que aumenten la posibilidad de supervivencia.
- *Recombinación/Crossover*: Permite el intercambio de información entre individuos de la población. Simula la reproducción entre individuos, necesaria para la transmisión de genes relevantes a las siguientes generaciones.
- *Selección*: Las estrategias de selección permiten definir aquellos individuos que participarán en la fase de reproducción, y por ende, los genes que pasarán a la siguiente generación. Esto simula el proceso de selección natural en el que sobreviven los individuos mejor adaptados al ambiente.

En la literatura se han desarrollado diferentes esquemas que definen el uso de estos operadores. Los *AE* más “tradicionales” son conocidos como *Algoritmos Genéticos* (*AG*) [Hol75], que suponen la aplicación más directa de los conceptos del proceso evolutivo. Sin embargo, dentro de la clase de *AE* existe otro grupo de algoritmos que aplican dichos conceptos de forma diferente. La clase de *Algoritmos de Estimación de Distribución* (“*Estimation of Distribution Algorithm*” - *EDA*) aplican los operadores de *mutación*, *recombinación* y *selección* sobre una población de soluciones implícita en un modelo de distribución probabilístico.

A continuación se describen cuatro algoritmos pertenecientes a la clase de *AE*: GGA, SGA y CHC, variantes del grupo de *AG*, y PBIL, perteneciente a los *EDA*.

#### 2.2.1.1. Generational Genetic Algorithm (GGA)

GGA es el esquema “tradicional” de aplicación de los *AG* [Bac96, Muh91]. Mantiene una población de individuos que evolucionan durante un número de iteraciones. Su principal característica es que en cada iteración se genera una nueva población, *i.e.* un proceso de evolución *generacional*.

En cada iteración el proceso evolutivo consiste en la creación de una nueva población de tamaño *pop* mediante: *a)* la selección de los individuos para el proceso de reproducción (*padres*), *b)* la recombinación (con probabilidad *cp*) de pares de individuos *padres* usando una estrategia particular de cruce/*crossover*, y *c)* la mutación de los individuos de la nueva población (llamados *descendencia*), usando una probabilidad de mutación de cada gen igual a *mp*. Ver el algoritmo 2.1.

---

**Algoritmo 2.1** Generational Genetic Algorithm

---

**Input:**  $\text{pop}$  tamaño de la población,  $\text{cp}$  probabilidad de cruce,  $\text{mp}$  probabilidad de mutación

**Output:** Una solución al problema

```

1:  $P \leftarrow$  Generar población aleatoria de  $\text{pop}$  individuos
2:  $s^* \leftarrow$  el mejor individuo en  $P$ 
3: while  $\neg$  Condición de parada do
4:    $P' \leftarrow \emptyset$ 
5:   while  $|P'| < \text{pop}$  do
6:      $p_1 \leftarrow$  Seleccionar un individuo en  $P$ 
7:      $p_2 \leftarrow$  Seleccionar un individuo en  $P$ 
8:      $c_1, c_2 \leftarrow$  recombinar  $p_1$  y  $p_2$  con probabilidad  $\text{cp}$ 
9:     Mutar  $c_1$  y  $c_2$  con probabilidad  $\text{mp}$ 
10:     $P' \leftarrow P' \cup \{c_1, c_2\}$ 
11:    $P \leftarrow P'$ 
12:   if El mejor individuo en  $P$  es mejor que  $s^*$  then
13:      $s^* \leftarrow$  el mejor individuo en  $P$ 
14: return  $s^*$ 
```

---

### 2.2.1.2. Steady-State Genetic Algorithm (SGA)

Descrito por *Whitley et al.* [WK88], SGA es una modificación del esquema general de *AG* que sigue una estrategia reproductiva no generacional. SGA comienza con una población de tamaño  $\text{pop}$ , y en cada iteración se producen un máximo de dos nuevos individuos (no una nueva población).

En cada iteración *a*) se seleccionan dos individuos padres de la población actual, *b*) se crea su descendencia (con probabilidad  $\text{cp}$ ) mediante algún método de cruce/recombinación, *c*) se agrega variabilidad mediante la mutación (con probabilidad  $\text{mp}$ ) de la nueva descendencia, y *d*) se sigue alguna estrategia de selección para reemplazar individuos en la población por la nueva descendencia, y así mantener el tamaño de la población igual a  $\text{pop}$ . Ver el algoritmo 2.2.

### 2.2.1.3. CHC Adaptive Search Algorithm

CHC [Esh90] se basa en el esquema de evolución generacional aplicado por GGA: mantiene una población de individuos de tamaño fijo ( $\text{pop}$ ), generando una nueva población en cada iteración. Sin embargo, en cada iteración CHC aplica una estrategia de reemplazo “elitista”, donde sobreviven los mejores individuos entre la población actual y la descendencia producida.

---

**Algoritmo 2.2** Steady-State Genetic Algorithm

---

**Input:**  $\text{pop}$  tamaño de la población,  $\text{cp}$  probabilidad de cruce,  $\text{mp}$  probabilidad de mutación

**Output:** Una solución al problema

- 1:  $P \leftarrow$  Generar población aleatoria de  $\text{pop}$  individuos
  - 2:  $s^* \leftarrow$  el *mejor* individuo en  $P$
  - 3: **while**  $\neg$  Condición de parada **do**
  - 4:      $p_1 \leftarrow$  Seleccionar un individuo en  $P$
  - 5:      $p_2 \leftarrow$  Seleccionar un individuo en  $P$
  - 6:      $c_1, c_2 \leftarrow$  recombinar  $p_1$  y  $p_2$  con probabilidad  $\text{cp}$
  - 7:     Mutar  $c_1$  y  $c_2$  con probabilidad  $\text{mp}$
  - 8:     Seguir algún criterio de reemplazo de individuos en  $P$  por  $c_1$  y  $c_2$
  - 9:     **if** El *mejor* individuo en  $P$  es *mejor* que  $s^*$  **then**
  - 10:          $s^* \leftarrow$  el *mejor* individuo en  $P$
  - 11: **return**  $s^*$
- 

La fase de reproducción aplicada por CHC tiene dos particularidades. En primer lugar, implementa un operador de recombinación uniforme media llamado HUX (“*Half Uniform Crossover*”), que intercambia la mitad de los genes que difieren entre los dos padres de forma aleatoria. Adicionalmente, CHC emplea “prevención de incesto”: antes de realizar el cruce usando HUX, calcula la *distancia de Hamming* entre ambos padres; si dicha distancia es mayor a cierto umbral (inicialmente  $l/4$ , donde  $l$  es la longitud de los cromosomas), se realiza el cruce. En caso de no generarse ninguna descendencia durante una iteración particular, se disminuye el umbral en 1.

Durante el proceso de evolución de CHC no se aplica el operador de mutación: cuando el umbral de prevención de incesto llega a cero se considera que la población convergió, y comienza un proceso de repoblación en el que se usa la mejor solución encontrada hasta el momento. Se modifican hasta 35 % de sus genes de forma aleatoria para generar los  $\text{pop} - 1$  individuos restantes de la nueva población, y luego continuar el proceso evolutivo.

El pseudocódigo de CHC de presenta en el algoritmo 2.3.

#### 2.2.1.4. Population-Based Incremental Learning (PBIL)

PBIL es una metaheurística perteneciente a la clase de *Algoritmos de Estimación de Distribución* desarrollada por Baluja [Bal94] para su uso sobre cromosomas con representación binaria. PBIL destaca por ser más simple que los algoritmos

---

**Algoritmo 2.3** CHC Adaptive Search Algorithm

---

**Input:** pop tamaño de la población**Output:** Una solución al problema

```

1:  $P \leftarrow$  Generar población aleatoria de pop individuos
2:  $s^* \leftarrow$  el mejor individuo en  $P$ 
3:  $\mu \leftarrow l/4$  ▷ Umbral de cruce
4: while  $\neg$  Condición de parada do
5:   for  $i \in [1 \dots \text{pop}/2]$  do
6:      $p_1 \leftarrow$  Seleccionar un individuo en  $P$ 
7:      $p_2 \leftarrow$  Seleccionar un individuo en  $P$ 
8:     if  $\text{hamming}(p_1, p_2) > \mu$  then
9:        $c_1, c_2 \leftarrow$  recombinar  $p_1$  y  $p_2$  usando HUX
10:       $P \leftarrow P \cup \{c_1, c_2\}$ 
11:    if  $|P| = \text{pop}$  then
12:       $\mu \leftarrow \mu - 1$ 
13:    if  $\mu = 0$  then
14:       $P \leftarrow$  Generar población de pop individuos usando  $s^*$ 
15:       $\mu \leftarrow l/4$ 
16:    else
17:       $P \leftarrow$  pop mejores individuos en  $P$ 
18:      if El mejor individuo en  $P$  es mejor que  $s^*$  then
19:         $s^* \leftarrow$  el mejor individuo en  $P$ 
20: return  $s^*$ 

```

---

genéticos tradicionales y por lograr mejores soluciones para gran variedad de problemas [Bal95, BC95].

Este algoritmo mantiene una población *implícita* de soluciones, mediante el uso de un vector de probabilidades  $V$  de tamaño  $l$ , donde  $V_i$  (con  $i \in [1 \dots l]$ ) es la probabilidad que el  $i$ -esimo bit/gen de una solución en la población esté “prendido” (sea igual a 1). PBIL usa este vector de probabilidades para generar poblaciones de tamaño pop en cada iteración, y guiar el proceso evolutivo en base a las soluciones generadas.

Inicialmente  $V_i = 0.5 \ \forall i \in [1 \dots l]$ . Luego en cada iteración: *a*) se generan pop cromosomas binarios basados en las probabilidades en  $V$ , *b*) se “acerca”  $V$  hacia la mejor solución generada (usando una tasa de aprendizaje **lr**), *c*) se “aleja”  $V$  de la peor solución generada (usando una tasa de aprendizaje negativa **nlr**), *d*) se sigue una estrategia de mutación sobre  $V$  en la que se aumenta o disminuye  $V_i$  en **ms** (*mutation shift*) con probabilidad de mutación **mp**. Ver algoritmo 2.4.

---

**Algoritmo 2.4** Population-Based Incremental Learning

---

**Input:**  $\text{pop}$  tamaño de la población,  $\text{mp}$  probabilidad de mutación,  $\text{ms}$  mutation shift,  $\text{l}\text{r}$  learning rate,  $\text{n}\text{l}\text{r}$  negative learning rate

**Output:** Una solución al problema

```

1:  $V \leftarrow$  Vector de probabilidades de tamaño  $l$ 
2:  $s^* \leftarrow$  Una solución cualquiera
3: while  $\neg$  Condición de parada do
4:    $P \leftarrow$  Generar población de tamaño  $\text{pop}$  según las probabilidades en  $V$ 
5:    $b \leftarrow$  El mejor individuo en  $P$ 
6:    $w \leftarrow$  El peor individuo en  $P$ 
7:   if  $b$  es mejor que  $s^*$  then
8:      $s^* \leftarrow b$ 
9:   for  $i \in [1 \dots l]$  do            $\triangleright$  Actualizar el vector de probabilidades
10:     $V_i \leftarrow V_i * (1 - \text{l}\text{r}) + b_i * \text{l}\text{r}$ 
11:    if  $b_i \neq w_i$  then
12:       $V_i \leftarrow V_i * (1 - \text{n}\text{l}\text{r}) + b_i * \text{n}\text{l}\text{r}$ 
13:    if  $\text{Unif}(0, 1) < \text{mp}$  then            $\triangleright$  Mutación con probabilidad  $\text{mp}$ 
14:       $V_i \leftarrow V_i * (1 - \text{ms}) + \text{UnifDiscreta}(0, 1) * \text{ms}$ 
15: return  $s^*$ 
```

---

## 2.2.2. Inteligencia de Enjambre

*Inteligencia de Enjambre* [BDT99] (*IE*) es un paradigma emergente entre los sistemas de cómputo bio-inspirados. Surge como una extensión de los *AE*, pero no se basa en la adaptación genética de poblaciones, sino en el comportamiento colectivo de grupos de organismos. Las estrategias de *IE* exhiben patrones de búsqueda descentralizada y auto-organizada, mediante la simulación de la inteligencia colectiva de grupos de “agentes” sencillos.

Durante la última década se han desarrollado numerosos enfoques basados en la explotación de inteligencia colectiva, inspirados en el comportamiento de colonias de hormigas, abejas, luciérnagas, etc. Uno de los más estudiados se conoce como PSO, inspirado en el vuelo de grupos de aves.

### 2.2.2.1. Particle Swarm Optimization (PSO)

PSO [KE95] se inspira en el comportamiento de organismos biológicos, en particular del vuelo de bandadas. Cada ave o “partícula” representa una solución que se mueve en el espacio de soluciones del problema, y modifica su “vuelo” en relación a su propia experiencia y la de sus “compañeras”. Diferentes estudios [SE98, KS98]

muestran que PSO obtiene mejores resultados que los algoritmos genéticos y en menor tiempo de cómputo.

La  $i$ -esima partícula de PSO tiene asociado *a*) un vector de posición  $x_i \in \mathbb{R}^l$  en un espacio euclíadiano  $l$ -dimensional que representa una solución al problema (*i.e.* un cromosoma de tamaño  $l$  con genes en pertenecientes a  $\mathbb{R}$ ), y *b*) un vector de velocidad  $v_i \in \mathbb{R}^l$  que modifica la posición de la partícula en cada iteración. Ver algoritmo 2.5.

---

**Algoritmo 2.5** Particle Swarm Optimization

---

**Input:** part número de partículas, vmax velocidad máxima, w peso de inercia, c1 peso del mejor local, c2 peso del mejor global

**Output:** Una solución al problema

```

1: for  $i \in [1 \dots \text{part}]$  do
2:    $\vec{X}_i \leftarrow$  Solución inicial aleatoria  $\in \mathbb{R}^l$ 
3:    $\vec{V}_i \leftarrow$  Vector de velocidades aleatorias entre  $[-\text{vmax}, \text{vmax}]$ 
4:    $p_i \leftarrow \vec{X}_i$             $\triangleright$  La mejor solución encontrada por la particula  $i$ 
5:    $s^* \leftarrow$  La mejor solución  $p_i, i \in [1 \dots \text{part}]$ 
6: while  $\neg$  Condición de parada do
7:   for  $i \in [1 \dots \text{part}]$  do
8:      $\vec{X}_i \leftarrow \vec{X}_i + \vec{V}_i$ 
9:      $\vec{V}_i \leftarrow w\vec{V}_i + c1 \text{ Unif}(0, 1)(p_i - \vec{X}_i) + c2 \text{ Unif}(0, 1)(s^* - \vec{X}_i)$ 
10:    Limitar valores en  $\vec{V}_i$  entre  $[-\text{vmax}, \text{vmax}]$ 
11:    if  $\vec{X}_i$  es mejor que  $p_i$  then
12:       $p_i \leftarrow \vec{X}_i$ 
13:      if  $p_i$  es mejor que  $s^*$  then
14:         $s^* \leftarrow p_i$ 
15: return  $s^*$ 
```

---

El vector de velocidad se modifica en función de varios parámetros: *a*) el peso de inercia  $w \in \mathbb{R}$  que evita cambios bruscos respecto a la velocidad anterior, *b*) el peso del mejor local  $c1 \in \mathbb{R}$  que indica la importancia de la mejor solución encontrada por dicha particula, y *c*) el peso del mejor global  $c2 \in \mathbb{R}$  que establece la importancia de la mejor solución global encontrada hasta el momento. Estos parámetros se encargan de dar dirección a la búsqueda de cada partícula. Usualmente los valores del vector de velocidad se limitan a ciertos rangos para permitir la explotación de soluciones locales: se usa el parámetro  $\text{vmax} \in \mathbb{R}$  para limitar la velocidad entre  $[-\text{vmax}, \text{vmax}]$ .

A pesar de estar definido para cromosomas con representación entre los números reales, PSO puede modificarse para optimizar soluciones con representaciones variadas, definiendo apropiadamente los operadores usados para la actualización de

la velocidad. Otro enfoque radica en mapear el espacio de búsqueda del problema a un dominio continuo: en el caso de soluciones con representación binaria se puede limitar la posición de las partículas entre  $[0, 1]$  y obtener soluciones redondeando dichos valores.

# Capítulo 3

## Adaptación al problema de Selección de Instancias

Al tratarse de métodos de búsqueda de propósito general, las metaheurísticas pueden modificarse para encontrar soluciones a todo tipo de problemas de optimización combinatoria. Para ello debe definirse la representación a usar, que permita codificar las posibles soluciones al problema y una función de *fitness* para evaluar dichas soluciones en función a los objetivos de optimización del problema.

A continuación se describen éstas y otras consideraciones generales para la aplicación de metaheurísticas al problema de *Selección de Instancias*. Adicionalmente se plantea para cada metaheurística, las estrategias a usar durante el proceso de búsqueda, así como algunas modificaciones particulares.

### 3.1. Consideraciones generales

#### 3.1.1. Representación

Una solución cualquiera al problema de *SI* está dada por un subconjunto de instancias  $R$  del conjunto inicial  $T$  (*i.e.*  $R \subseteq T$ ). Por lo tanto, para un orden dado de las instancias  $t_i \in T$  ( $i = 1 \dots n$ , con  $n = |T|$ ), una codificación usando mapas de bits es suficiente para representar el espacio de soluciones al problema.

En este sentido, una solución particular al problema de *SI* está descrita por un cromosoma  $s$  de tamaño  $n$ , donde cada gen  $s_i$  ( $i = 1 \dots n$ ) es un *bit* con valor

1/0 (“prendido” o “apagado” respectivamente) que denota la pertenencia o no de la instancia correspondiente  $t_i \in T$  al subconjunto seleccionado.  $R_s \subseteq T$  es el subconjunto seleccionado representado por el cromosoma  $s$ , donde:

$$R_s = \{t_i \in T \mid i = 1 \dots n \wedge s_i = 1\} \quad (3.1)$$

Este conjunto  $R_s$  permite evaluar la “bondad” de la solución  $s$  en función de los objetivos de optimización del problema.

### 3.1.2. Función de evaluación

El objeto de la aplicación de metaheurísticas es conseguir soluciones óptimas o casi óptimas a gran variedad de problemas. Para ello es necesario definir una estrategia de comparación entre soluciones, *i.e.* saber cuándo una solución es «*mejor*» que otra, que permita seleccionar aquellas soluciones que mejor se adapten a los objetivos del problema en cuestión.

Para el caso del problema de *SI*, el objetivo es encontrar un subconjunto  $R$  del menor tamaño posible, que mantenga un alto porcentaje de precisión de clasificación. *Cano et al.* [CHL03] definen una función de evaluación que combina ambos objetivos en función de un parámetro  $\alpha \in [0, 1]$ ; a continuación se presenta una modificación de dicha función (su complemento) usada en el presente trabajo:

$$\text{fitness}(R) = \alpha \text{ error}(R) + (1 - \alpha)|R| \quad (3.2)$$

Donde  $\text{error}(R)$  es el número de instancias en  $T$  que son erróneamente clasificadas usando el conjunto  $R \subseteq T$  como conjunto de entrenamiento de un clasificador 1-NN. El parámetro  $\alpha$  combina los objetivos de la búsqueda: se usa  $\alpha = 0.5$  siguiendo lo descrito por [CHL03] para obtener soluciones que satisfagan ambos objetivos del problema.

Para esta definición, el objetivo de las metaheurísticas para selección de instancias es *minimizar* la función de evaluación descrita. Para ello, deben minimizar ambos objetivos de la función: el error de clasificación usando el subconjunto seleccionado y el tamaño de dicho subconjunto. En este sentido, dadas dos posibles soluciones  $a$  y  $b$ ,  $a$  es «*mejor*» que  $b$  si y solo si  $\text{fitness}(R_a) < \text{fitness}(R_b)$ .

### 3.1.3. Generación de soluciones iniciales

Para muchos problemas con esquemas de representación binaria, la generación de soluciones iniciales usada por metaheurísticas sigue una estrategia común: al generar un cromosoma aleatorio, cada bit tiene 50 % de probabilidad de estar prendido ( $\delta = 0.5$ ). Esta estrategia genera soluciones con valor esperado de la mitad de los bits prendidos. Para el problema de *SI* esto implica soluciones con una reducción inicial del 50 % sobre el conjunto  $T$ , y hace necesario un alto número de iteraciones para lograr obtener soluciones con reducciones significativas [CHL03]. Por esta razón, en el presente trabajo se usa una probabilidad de aparición del 5 % por cada bit ( $\delta = 0.05$ ), generando soluciones iniciales con reducciones cercanas al 95 %. El rol de la función objetivo no recae en disminuir los porcentajes de reducción de las soluciones, sino en mantenerlos en niveles aceptables, permitiendo explotar el espacio de búsqueda para conseguir soluciones con mayor precisión.

Sin embargo, el punto inicial de la búsqueda en términos de precisión sigue siendo aleatorio. Una estrategia común en metaheurísticas de trayectoria es la generación de soluciones iniciales usando algoritmos de aproximación; *Cerveron et al.* [CF01] plantean una modificación del algoritmo de Búsqueda Tabú para el problema de *SI*, en la cuál usan CNN para generar la solución inicial de la búsqueda. Esta idea puede trasladarse a metaheurísticas poblacionales siguiendo un enfoque probabilístico: dada una selección inicial  $R_0 \subseteq T$ , los bits correspondientes a instancias en  $R_0$  tienen mayor probabilidad de aparición que los bits de instancias en  $T \setminus R_0$ . Estas probabilidades constituyen un vector de probabilidades a ser usado por PBIL como vector inicial, o por los algoritmos genéticos (GGA, SGA y CHC) para generar soluciones iniciales. El algoritmo 3.1 implementa este esquema de generación; para un  $\delta$  particular (que determina el número de bits prendidos en las soluciones iniciales), el vector de probabilidades  $V$  generado por el algoritmo cumple con que un máximo del 70 % de los bits prendidos en soluciones generadas usando  $V$ , pertenecen a la solución inicial  $R_0$ . Esto contribuye a guiar la búsqueda realizada por las metaheurísticas poblacionales, mientras se mantiene la variabilidad.

Cualquier solución generada por algoritmos de aproximación para el problema de *SI* (sección 1.3) sirve como “semilla” de este generador. En el presente trabajo se prueba el impacto del uso de soluciones iniciales calculadas por CNN. También se incluyen dos selecciones basadas en el orden según el *enemigo más cercano* (NE -

**Algoritmo 3.1** Generador de vector de probabilidades inicial**Input:**  $R_0$  solución inicial**Output:** Vector de probabilidades en base a  $R_0$ 

```

1:  $high \leftarrow \min(0.9, \frac{\delta|T|^{0.7}}{|R_0|})$ 
2:  $low \leftarrow \frac{\delta|T|-hi|R_0|}{|T \setminus R_0|}$ 
3:  $V \leftarrow$  Vector de probabilidades de tamaño n
4: for  $i \in [1 \dots n]$  do
5:   if  $t_i \in R_0$  then
6:      $V_i \leftarrow high$ 
7:   else
8:      $V_i \leftarrow low$ 
9: return  $V$ 

```

“*Nearest Enemy*”), seleccionando las instancias con mayor distancia NE (selección por *Farthest NE*), o menor distancia NE (selección por *Closest NE*). Finalmente, se prueba un método de selección propio llamado *Nearest Enemy Hypersphere Selection* (NEHS). En la figura 3.2 se pueden observar los subconjuntos seleccionados por estas cuatro (4) estrategias.

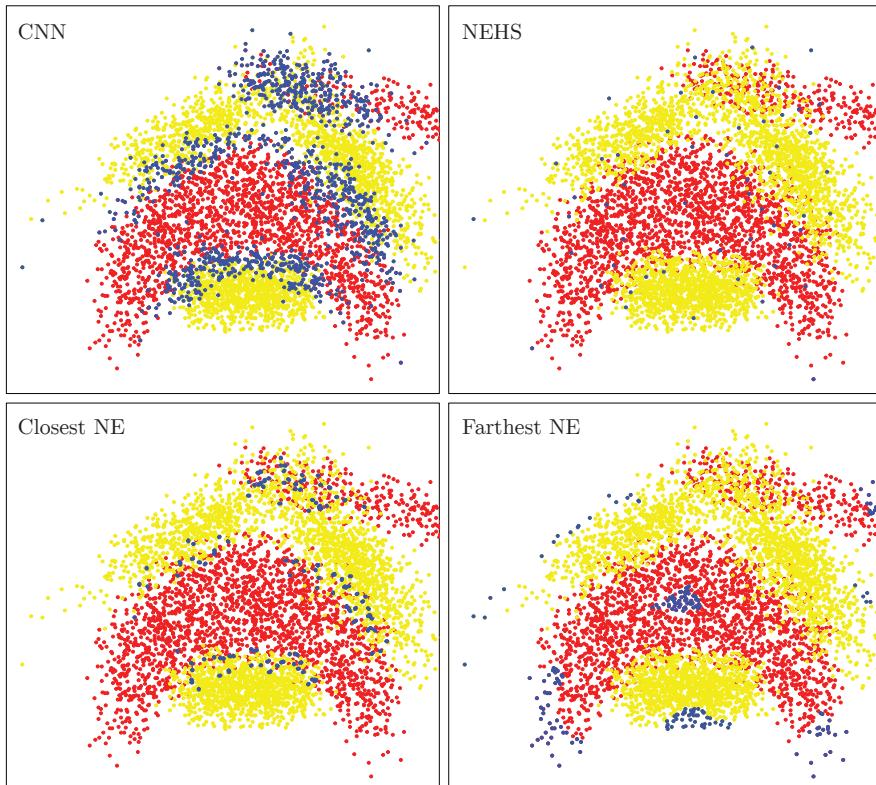


FIGURA 3.1: Subconjunto de instancias seleccionadas por CNN, NEHS, Closest NE y Farthest NE, para el conjunto de instancias BANANA. Las instancias están pintadas en amarillo y rojo según su clase en el conjunto de datos, y en azul aquellas instancias pertenecientes a la selección correspondiente.

### 3.1.3.1. Condensed Nearest Neighbor

En 1968 *Hart* [Har68] fue el primero en proponer un método de reducción de instancias a ser usadas por la regla NN; este método se conoce como *Condensed Nearest Neighbor* (CNN). El objetivo de CNN es realizar un proceso de selección previo al entrenamiento del clasificador 1-NN, consiguiendo un conjunto  $R \subseteq T$  que mantenga la efectividad del conjunto de datos original  $T$  para clasificar instancias desconocidas.

Se trata de un algoritmo incremental en el que se incluyen en  $R$  aquellas instancias en  $T$  que son mal clasificadas usando el conjunto reducido como conjunto de entrenamiento en un clasificador 1-NN. Dicho proceso se repite hasta que no se incluyan nuevas instancias en el conjunto  $R$  luego de una iteración completa sobre las instancias en  $T$ ; ver algoritmo 3.2.

---

#### Algoritmo 3.2 Condensed Nearest Neighbor

---

**Input:**  $T$  conjunto de instancias inicial

**Output:** Conjunto de instancias  $R \subseteq T$

```

1:  $R \leftarrow \{\text{Una instancia cualquiera } t \in T\}$ 
2: repeat
3:    $R' \leftarrow R$ 
4:   for all  $t \in T$  do
5:     if  $t$  es mal clasificada usando  $R$  con un clasificador 1-NN then
6:        $R \leftarrow R \cup \{t\}$ 
7: until  $R = R'$ 
8: return  $R$ 

```

---

CNN logra una reducción considerable sobre el conjunto de datos original, y asegura un conjunto consistente con  $T$ . Sin embargo, al ser dependiente del orden de revisión de las instancias, CNN no asegura un conjunto consistente *mínimo*. Este algoritmo tiende a seleccionar instancias cercanas a los bordes de decisión, viéndose particularmente afectado por datos ruidosos [AR11, JG].

### 3.1.3.2. Nearest Enemy Hypersphere Selection

La idea tras el uso de la regla NN es que instancias “cercanas” en un espacio cualquiera (*i.e.* con atributos similares) comparten su clasificación; esto conlleva a la división de dicho espacio en regiones con instancias de igual clase. La idea

fundamental de *Nearest Enemy Hypersphere Selection* (NEHS) es seleccionar instancias en los centros de dichas regiones, con la finalidad de reducir la cantidad de instancias necesarias para generalizar el espacio descrito por el conjunto original.

Para esto hace uso de la distancia del *enemigo más cercano*: cada instancia  $t \in T$  tiene un enemigo más cercano  $\text{NE}(t) \in T$ , cuya distancia  $\varphi(t, \text{NE}(t))$  define el radio de una hiperesfera en el espacio  $m$ -dimensional de atributos con centro en  $t$ , dentro de la cual toda instancia comparte la clase  $\omega_t$ .

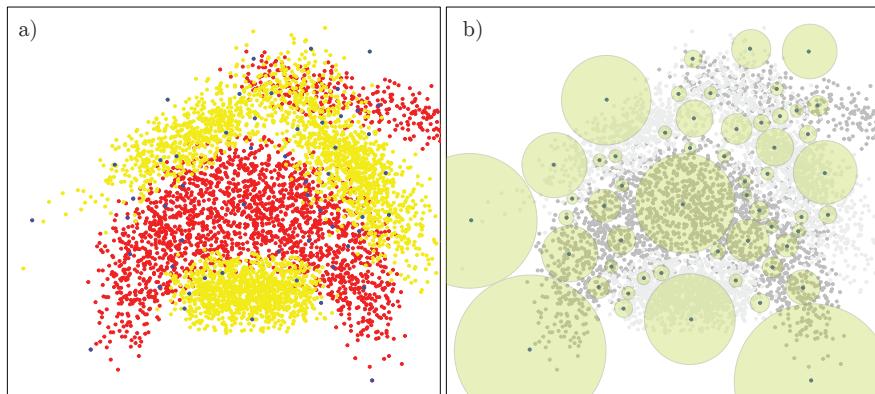


FIGURA 3.2: a) Subconjunto de instancias seleccionadas por NEHS, e  
b) Hiperesferas correspondientes a la selección de NEHS, con radio en  
función a la distancia del enemigo más cercano.

NEHS implementa una estrategia *greedy* que busca seleccionar las mayores hiperesferas que no intersecten entre sí. Realiza la selección de hiperesferas comenzando con las de mayor tamaño, con el objetivo de cubrir el mayor espacio posible (ver el algoritmo 3.3). Sin embargo, para evitar la selección de puntos cercanos a los bordes de decisión (y posiblemente ruidosos), NEHS ignora un porcentaje  $\Delta$  de instancias en  $T$  con menor distancia NE, *i.e.* las hiperesferas de menor radio. Esto implica una selección con bordes de decisión mucho más “suaves” que los del conjunto original.

---

### Algoritmo 3.3 Nearest Enemy Hypersphere Selection

---

**Input:**  $T$  conjunto de instancias inicial,  $\Delta$  porcentaje de instancias a excluir

**Output:** Conjunto de instancias  $R \subseteq T$

- 1:  $R \leftarrow \emptyset$
  - 2: **for all**  $t \in T$  en orden descendiente de distancia NE  
excluyendo las últimas  $\Delta|T|$  instancias **do**
  - 3:     **if**  $\neg\exists r \in R$  tal que  $\varphi(t, r) < \varphi(r, \text{NE}(r)) + \varphi(t, \text{NE}(t))$  **then**
  - 4:          $R \leftarrow R \cup \{t\}$
  - 5: **return**  $R$
-

### 3.1.3.3. Closest Nearest Enemy

La distancia del *enemigo más cercano* (NE) de una instancia cualquiera  $t \in T$  (*i.e.*  $\varphi(t, \text{NE}(t))$ ), indica su cercanía a los bordes de decisión establecidos mediante un clasificador 1-NN. La selección por *Closest NE* escoge aquellas instancias con *menor* distancia NE entre las instancias en  $T$ , *i.e.* instancias pertenecientes –o cercanas– a los bordes de decisión de los datos. En el presente trabajo, *Closest NE* selecciona el 5 % ( $\delta = 0.05$ ) de instancias con menor distancia NE.

$$R_{\text{ClosestNE}} = \{\delta|T| \text{ instancias con menor distancia NE}\} \quad (3.3)$$

### 3.1.3.4. Farthest Nearest Enemy

Similar a la selección por *Closest NE*, *Farthest NE* se basa en el uso de la distancia NE para incluir instancias en el conjunto reducido  $R \subseteq T$ . Sin embargo, *Farthest NE* selecciona aquellas instancias con *mayor* distancia NE, *i.e.* instancias alejadas de los bordes de decisión y cercanas a los centros de sus respectivas regiones. Este método también usa un porcentaje de selección igual al 5 % ( $\delta = 0.05$ ) de las instancias en  $T$ .

$$R_{\text{FarthestNE}} = \{\delta|T| \text{ instancias con mayor distancia NE}\} \quad (3.4)$$

## 3.2. Modificaciones particulares

### 3.2.1. Adaptación de GGA, SGA y CHC

Para aplicar algún algoritmo genético al problema de *SI*, es necesario describir las estrategias de selección, recombinación, mutación y reemplazo que seguirá el algoritmo durante el proceso evolutivo.

GGA, SGA y CHC requieren de un operador que seleccione individuos de la población para participar en el proceso reproductivo. Se emplea el método de *selección por torneo* en su versión “determinista”: se escoge al azar un conjunto de individuos entre la población original, y es seleccionado el más apto entre ellos. El

tamaño de dicho conjunto debe ser pequeño; generalmente se usa un “tamaño de torneo” entre 2 y 5 [MG95]. En este trabajo se usan torneos de tamaño 3.

Debido a que CHC aplica un operador de recombinación particular (HUX) y no requiere de un operador de mutación, dichos operadores deben ser definidos solo para GGA y SGA. El operador de *crossover* aplicado en ambos algoritmos es el de *recombinación en un punto*. Dados dos cromosomas “padres”, se elige aleatoriamente un punto de corte en la longitud de ambos cromosomas; los cromosomas “hijos” resultan de combinar la sección izquierda del corte de un padre, con la sección derecha del corte del otro parente. El operador de mutación sigue –en ambos casos– el esquema estándar de modificación de cada bit con una cierta probabilidad.

Finalmente, es necesario describir los criterios de reemplazo. Al tratarse de estrategias evolutivas generacionales, GGA y CHC reemplazan la población de forma incondicional en favor de la descendencia. En cambio, el criterio de reemplazo usado en SGA es “elitista”: se sustituyen los padres por la descendencia generada, solo si dicha descendencia es mejor.

### 3.2.2. Adaptación de PBIL

La estrategia evolutiva de PBIL no requiere modificaciones adicionales para su aplicación al problema de selección de instancias. PBIL está diseñado para encontrar soluciones con representación binaria, lo que permite su aplicación directa al problema. La única modificación realizada al esquema estándar de PBIL es en el método de generación del vector de probabilidades iniciales, reflejando lo descrito en la sección 3.1.3.

### 3.2.3. Adaptación de PSO

La adaptación de PSO al problema de *SI* es mayor, debido a que PSO está pensado para problemas con representación en espacios euclidianos, *i.e.* es necesario el mapeo de vectores en  $\mathbb{R}^l$  a soluciones con representación binaria. En este sentido, se adopta el esquema poblacional de PBIL en el que el genotipo de la población se representa de forma probabilística.

A esta adaptación de PSO la llamaremos *Population-Based PSO* (ver algoritmo 3.4), en la cuál el vector de posición  $\vec{X}_i$  de cada partícula es un vector de probabilidades que representa el genotipo de una población particular. Esto permite la generación de soluciones en codificación binaria para su respectiva evaluación, en base al vector de probabilidades que describe dicha población. A diferencia del PSO tradicional, la selección de óptimos locales y globales se hace en base a las soluciones generadas mediante  $\vec{X}_i$ , y no en base al vector  $\vec{X}_i$  por si solo. Sin embargo, la modificación de los vectores de probabilidad y velocidad ( $\vec{X}_i$  y  $\vec{V}_i$  respectivamente) sigue la estrategia de actualización estándar en base a la mejor solución local y la mejor solución global.

---

**Algoritmo 3.4** Population-Based PSO

---

**Input:** pop tamaño de la población, part número de partículas, vmax velocidad máxima, w peso de inercia, c1 peso del mejor local, c2 peso del mejor global  
**Output:** Una solución al problema

```

1: for  $i \in [1 \dots \text{part}]$  do
2:    $\vec{X}_i \leftarrow$  Vector de probabilidades de tamaño  $l$ 
3:    $\vec{V}_i \leftarrow$  Vector de velocidades aleatorias entre  $[-\text{vmax}, \text{vmax}]$ 
4:    $p_i \leftarrow$  Generar una solución a partir de  $\vec{X}_i$ 
5:    $s^* \leftarrow$  La mejor solución  $p_i, i \in [1 \dots \text{part}]$ 
6: while  $\neg$  Condición de parada do
7:   for  $i \in [1 \dots \text{part}]$  do
8:      $\vec{X}_i \leftarrow \vec{X}_i + \vec{V}_i$ 
9:     Limitar valores en  $\vec{X}_i$  entre  $[0, 1]$ 
10:     $\vec{V}_i \leftarrow w\vec{V}_i + c1 \text{ Unif}(0, 1)(p_i - \vec{X}_i) + c2 \text{ Unif}(0, 1)(s^* - \vec{X}_i)$ 
11:    Limitar valores en  $\vec{V}_i$  entre  $[-\text{vmax}, \text{vmax}]$ 
12:     $P \leftarrow$  Generar población de tamaño pop a partir de  $\vec{X}_i$ 
13:    if El mejor individuo en  $P$  es mejor que  $p_i$  then
14:       $p_i \leftarrow$  El mejor individuo en  $P$ 
15:      if  $p_i$  es mejor que  $s^*$  then
16:         $s^* \leftarrow p_i$ 
17: return  $s^*$ 
```

---

Esta versión de PSO puede clasificarse como una metaheurística de la clase de *Algoritmos de Coevolución Cooperativa* [DGH09] (puesto que mantiene diferentes poblaciones que evolucionan de forma colaborativa) y *Algoritmos de Estimación de Distribución* (debido a que adopta las ideas de representación descritas por PBIL).

# Capítulo 4

## Evaluación Experimental

Este capítulo consiste en la descripción de la metodología usada durante la evaluación de los métodos introducidos en los capítulos anteriores, así como el análisis de los resultados obtenidos de dicha evaluación. El objetivo de este estudio radica en la comparación empírica de cinco metaheurísticas (*i.e.* GGA, SGA, CHC, PBIL y PSO) adaptadas para encontrar soluciones al problema de selección de instancias. Adicionalmente, se pretende estudiar el impacto de la aplicación de estrategias alternativas de generación de soluciones iniciales, usando como semilla los subconjuntos de instancias generados por diferentes algoritmos heurísticos al problema en cuestión (*i.e.* CNN, NEHS, ClosestNE y FarthestNE).

### 4.1. Diseño Experimental

La metodología seguida durante la evaluación experimental permite establecer la efectividad de las metaheurísticas descritas en función de *a)* la reducción del conjunto de instancias usado para el entrenamiento del clasificador (en este caso un clasificador 1-NN), y *b)* la precisión de dicho clasificador –una vez entrenado– al clasificar instancias previamente desconocidas. La metodología experimental debe permitir la generalización del comportamiento de las diferentes metaheurísticas, así como las estrategias de generación de soluciones iniciales, con la finalidad de comparar los resultados obtenidos y establecer los métodos más efectivos frente al problema de selección de instancias.

### 4.1.1. Conjuntos de datos

Un factor esencial en la evaluación de métodos de selección de instancias es el conjunto de datos utilizado; la distribución de los datos, el número de instancias y atributos, y la cantidad de datos ruidosos, son solo algunos de los elementos que modifican el espacio de búsqueda del problema y por ende la efectividad de los algoritmos heurísticos para encontrar buenas soluciones. Los conjuntos de datos usados en este trabajo pertenecen al *UCI Machine Learning Repository* [BL13] y al *KEEL Data-Mining Software Tool* [AFL<sup>+</sup>10].

Los 14 conjuntos seleccionados fueron separados en 3 grupos en función del número de instancias de cada conjunto. En la tabla 4.1 se presentan 9 conjuntos con menor cantidad de instancias.

CONJUNTO	INSTANCIAS	ATRIBUTOS	CLASES
Cleveland	297	13	2
Glass	214	9	6
Iris	150	4	3
LED7Digit	500	7	10
Monk	432	6	2
Pima	768	8	2
WDBC	569	30	2
Wine	178	13	3
Wisconsin	683	9	2

CUADRO 4.1: Conjuntos de datos pequeños

En la tabla 4.2 se describen dos conjuntos de datos caracterizados como de tamaño medio.

CONJUNTO	INSTANCIAS	ATRIBUTOS	CLASES
Banana	5300	2	2
Segmentation	2100	19	7

CUADRO 4.2: Conjuntos de datos medianos

Finalmente, los 3 conjuntos con mayor número de instancias se presentan en la tabla 4.3.

CONJUNTO	INSTANCIAS	ATRIBUTOS	CLASES
Pen-Based	10992	16	10
Satimage	6435	36	6
Thyroid	7200	21	3

CUADRO 4.3: Conjuntos de datos grandes

### 4.1.2. Particiones y ejecuciones

Los conjuntos de datos considerados en la sección anterior son particionados usando la estrategia de validación cruzada en 10 iteraciones (*10-fold cross-validation*). Dado el conjunto inicial de instancias  $T$ , es dividido aleatoriamente en 10 conjuntos disjuntos de igual tamaño  $T_0, T_1, \dots, T_{10}$ . Adicionalmente, cada conjunto  $T_i$  mantiene la proporción de distribución de las clases del conjunto original  $T$ . En función de esta partición, se definen un par de conjuntos  $(T'_i, T_i)$ , con  $i = 1 \dots 10$ , donde  $T'_i = T \setminus T_i$ .

El conjunto  $T'_i$  (el conjunto de “entrenamiento”) es usado por las metaheurísticas durante el proceso de búsqueda para evaluar soluciones intermedias; en vez de usar  $T$ , se usa el conjunto  $T'_i$  como el conjunto de instancias inicial. Las instancias restantes, pertenecientes al conjunto  $T_i$ , son usadas como conjunto de “validación”; la mejor solución encontrada por las metaheurísticas es usada para clasificar las instancias –previamente desconocidas– de  $T_i$ .

Cada metaheurística es evaluada usando los 10 pares de conjuntos  $(T'_i, T_i)$ . Por cada par de conjunto entrenamiento-validación se realizaron 3 repeticiones, para un total de 30 ejecuciones de una metaheurística para un conjunto de datos y un tipo de inicialización particular.

### 4.1.3. Parámetros

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

PARÁMETROS	ALGORITMOS				
	GGA	SGA	CHC	PBIL	PSO
Iteraciones	1000	1000	1000	1000	1000
Población	50	30	30	40	5
Prob. de Cruce	0.9	1.0	-	-	-
Prob. de Mutación	0.001	0.001	-	0.001	-
Mutation Shift	-	-	-	0.01	-
Learning Rate	-	-	-	0.1	-
Neg. Learning Rate	-	-	-	0.01	-
Partículas	-	-	-	-	5
Velocidad Máxima	-	-	-	-	0.2
Inercia	-	-	-	-	0.9
c1	-	-	-	-	0.1
c2	-	-	-	-	0.1

CUADRO 4.4: Parámetros usados en cada metaheurística

## 4.2. Resultados

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

# Conclusiones y Recomendaciones

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut

metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

1. First itemtext
2. Second itemtext
3. Last itemtext
4. First itemtext
5. Second itemtext

# Bibliografía

- [AFL<sup>+</sup>10] J Alcalá, A Fernández, J Luengo, J Derrac, S García, L Sánchez, and 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*, 17:255–287, 2010.
- [AR11] MILOUD-AOUIDATE Amal and BABA-ALI Ahmed Riadh. Survey of nearest neighbor condensing techniques. 2011.
- [Bac96] T. Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, USA, 1996.
- [Bal94] Shumeet Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical report, 1994.
- [Bal95] Shumeet Baluja. An empirical comparison of seven iterative and evolutionary function optimization heuristics. Technical report, 1995.
- [BC95] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. pages 38–46. Morgan Kaufmann Publishers, 1995.
- [BDT99] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, Inc., New York, NY, USA, 1999.
- [Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.
- [BL97] Avrim Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artif. Intell.*, 97(1-2):245–271, 1997.

- [BL13] K. Bache and M. Lichman. UCI machine learning repository, 2013.
- [BS12] S Siva Sathya Binitha S. A survey of bio inspired optimization algorithm. *International Journal of Soft Computing and Engineering*, 2(2):137–151, 2012.
- [BT12] J. Bien and R. Tibshirani. Prototype selection for interpretable classification. *ArXiv e-prints*, February 2012.
- [CF01] Vicente Cerveron and Francesc J Ferri. Another move toward the minimum consistent subset: a tabu search approach to the condensed nearest neighbor rule. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 31(3):408–413, 2001.
- [CH67] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, January 1967.
- [CHL03] José Ramón Cano, Francisco Herrera, and Manuel Lozano. Using evolutionary algorithms as instance selection for data reduction in kdd: an experimental study. *Evolutionary Computation, IEEE Transactions on*, 7(6):561–575, 2003.
- [DGH09] Joaquín Derrac, Salvador García, and Francisco Herrera. A first study on the use of coevolutionary algorithms for instance and feature selection. In *Proceedings of the 4th International Conference on Hybrid Artificial Intelligence Systems*, HAIS ’09, pages 557–564, Berlin, Heidelberg, 2009. Springer-Verlag.
- [DGKL94] Luc Devroye, Laszlo Gyorfi, Adam Krzyzak, and Gábor Lugosi. On the strong universal consistency of nearest neighbor regression function estimates. *The Annals of Statistics*, pages 1371–1385, 1994.
- [Esh90] Larry J Eshelman. The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. *Foundations of genetic algorithms*, pages 265–283, 1990.
- [FH51] E. Fix and J. L. Hodges. Discriminatory analysis, nonparametric discrimination: Consistency properties. *US Air Force School of Aviation Medicine*, Technical Report 4(3):477+, January 1951.

- [FI93] Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In Ruzena Bajcsy, editor, *IJCAI*, pages 1022–1029. Morgan Kaufmann, 1993.
- [Gat72] Geoffrey W. Gates. The reduced nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 18(3):431–433, 1972.
- [GCH08] Salvador García, José Ramón Cano, and Francisco Herrera. A memetic algorithm for evolutionary prototype selection: A scaling up approach. *Pattern Recognition*, 41(8):2693–2709, 2008.
- [GK14] Lee-Ad Gottlieb and Aryeh Kontorovich. Near-optimal sample compression for nearest neighbors. *CoRR*, abs/1404.3368, 2014.
- [GKK13] Lee-Ad Gottlieb, Aryeh Kontorovich, and Robert Krauthgamer. Efficient classification for metric data. *CoRR*, abs/1306.2547, 2013.
- [GPY08] Roberto Gil-Pita and Xin Yao. Evolving edited k-nearest neighbor classifiers. *International Journal of Neural Systems*, 18(06):459–467, 2008.
- [Har68] P. Hart. The condensed nearest neighbor rule (corresp.). *IEEE Trans. Inf. Theor.*, 14(3):515–516, September 1968.
- [HLL02] Shinn-Ying Ho, Chia-Cheng Liu, and Soundy Liu. Design of an optimal nearest neighbor classifier using an intelligent genetic algorithm. *Pattern Recognition Letters*, 23(13):1495–1503, 2002.
- [Hol75] J.H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.
- [JG] Norbert Jankowski and Marek Grochowski. Comparison of instances selection algorithms i. algorithms survey.
- [KE95] J Kennedy and R Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948. IEEE, 1995.
- [KL04] Robert Krauthgamer and James R. Lee. Navigating nets: simple algorithms for proximity search. In J. Ian Munro, editor, *SODA*, pages 798–807. SIAM, 2004.

- [KS98] J Kennedy and WM Spears. Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 78–83. IEEE, 1998.
- [KS04] Mohammad Kolahdouzan and Cyrus Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 840–851. VLDB Endowment, 2004.
- [LHTD02] Huan Liu, Farhad Hussain, Chew Lim Tan, and Manoranjan Dash. Discretization: An enabling technique. *Data Min. Knowl. Discov.*, 6(4):393–423, October 2002.
- [LM98] Huan Liu and Hiroshi Motoda. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.
- [LM02] Huan Liu and Hiroshi Motoda. On issues of instance selection. *Data Min. Knowl. Discov.*, 6(2):115–130, April 2002.
- [MG95] Brad L. Miller and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.
- [Muh91] Heinz Muhlenbein. Evolution in time and space - the parallel genetic algorithm. In *Foundations of Genetic Algorithms*, pages 316–337. Morgan Kaufmann, 1991.
- [SE98] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69–73. IEEE, 1998.
- [Ska94] David B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In William W. Cohen and Haym Hirsh, editors, *ICML*, pages 293–301. Morgan Kaufmann, 1994.
- [SLI<sup>+</sup>01] Basilio Sierra, Elena Lazkano, Iñaki Inza, Marisa Merino, Pedro Larrañaga, and Jorge Quiroga. Prototype selection and feature subset

selection by estimation of distribution algorithms. a case study in the survival of cirrhotic patients treated with tips. In *Artificial Intelligence in Medicine*, pages 20–29. Springer, 2001.

- [SSBD14] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.
- [Tou02] Godfried T. Toussaint. Open problems in geometric methods for instance-based learning. In Jin Akiyama and Mikio Kano, editors, *JCDCG*, volume 2866 of *Lecture Notes in Computer Science*, pages 273–283. Springer, 2002.
- [Vor08] Georges Voronoï. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire. recherches sur les parallolloèdres primitifs. *Journal für die reine und angewandte Mathematik*, 134:198–287, 1908.
- [Wil72] DR Wilson. Asymptotic properties of nearest neighbor rules using edited data. *Institute of Electrical and Electronic Engineers Transactions on Systems, Man and Cybernetics*, 2:408–421, 1972.
- [Wil91] Gordon Wilfong. Nearest neighbor problems. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, SCG ’91, pages 224–233, New York, NY, USA, 1991. ACM.
- [WK88] D. Whitley and J. Kauth. *GENITOR: A Different Genetic Algorithm*. Technical report (Colorado State University. Department of Computer Science). Colorado State University, Department of Computer Science, 1988.
- [WM97] D. Randall Wilson and Tony R. Martinez. Instance pruning techniques. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML ’97, pages 403–411, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.
- [Yan08] Xin-She Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.
- [ZS02] Hongbin Zhang and Guangyu Sun. Optimal reference subset selection for nearest neighbor classification by tabu search. *Pattern Recognition*, 35(7):1481–1490, 2002.

- [Zuk10] A. V. Zuhkba. Np-completeness of the problem of prototype selection in the nearest neighbor method. *Pattern Recognit. Image Anal.*, 20(4):484–494, December 2010.