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

Título

Por:
Antonio Álvarez

Realizado con la asesoría de:
Emely Arráiz 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 2018

Resumen

Hola mundo

Índice general

Índice de figuras

Índice de Tablas

Índice de algoritmos

Acrónimos y Símbolos

KDD	Knowledge Discovery in Databases
DM	Data Mining
IS	Instance Selection
PS	Prototype Selection
NN	Nearest Neighbor
NE	Nearest Enemy
CNN	Condensed Nearest Neighbor
ENN	Edited Nearest Neighbor
RSS	Relaxed Selective Subset
GGA	Generational Genetic Algorithm
SGA	Steady-State Genetic Algorithm
CHC	CHC Adaptive Search Algorithm
MEM	Memetic Algorithm

$\in$	Relación de pertenencia, «*es un elemento de*»
$\subseteq$	Subconjunto
$\setminus$	Diferencia de conjuntos

Introducción

Introdución

Capítulo 1

Marco teórico

1.1. Descubrimiento de Conocimiento y preprocesamiento de datos

Hoy en día, existe una creciente necesidad de procesar grandes volúmenes de datos, estos datos son producto de la recolección de información de procesos y actividades de distintas índoles y se vuelven un material valioso para extraer información sobre posibles tendencias que puedan existir en dichos procesos. Es aquí donde entra el descubrimiento de conocimiento en bases de datos (KDD por su siglas en inglés) como disciplina encargada del procesamiento de datos para la extracción de información.

KDD es definida por *Smyth, P. et al.* [FSS96] como "el proceso no trivial de identificar patrones en los datos que sean válidos, novedosos, potencialmente útiles y finalmente entendibles". Para este fin, KDD se subdivide en distintas etapas a llevar a cabo para lograr el fin último de identificar patrones, éstas son [GLH16]: especificación del problema, entendimiento del problema, preprocesamiento de los datos, minería de datos, evaluación de los resultados y explotación de los resultados. En este trabajo es de especial interés la etapa de preprocesamiento de datos.

El preprocesamiento de datos consiste en el conjunto actividades destinadas a preparar los datos para ser usado por un algoritmo de minería de datos (DM por sus siglas en inglés). Las actividades realizadas en el preprocesamiento pueden ser clasificadas como actividades para la preparación de los datos y la reducción de los mismos [GLH16].

La preparación de datos es un paso obligatorio en el preprocesamiento, ya que transforma los datos, que inicialmente no se pueden utilizar para el algoritmo de DM por asuntos como la presencia de atributos faltantes en instancias, datos erróneos y atributos con formatos no aceptables para el algoritmo a utilizar [GLH16]. Dependiendo del enfoque dado, estas actividades pueden clasificarse en:

- **Limpieza de datos [GLH16, KCH+03]:** incluye el tratamiento de los atributos faltantes y los datos erróneos, que si se dejan sin tratar resulta en un modelo de minería de datos poco confiable. Un atributo faltante en una instancia resulta de no haberlo introducido al momento del registro o por la pérdida en el proceso de almacenamiento. Los datos con atributos faltantes pueden tratarse de 3 maneras [FKP07]: la eliminación de las instancias que presenten el problema, utilizar métodos de estimación de máxima verosimilitud para calcular promedios y variancias y utilizar algoritmos del repertorio de *machine learning* como k-nn, k-means o *Suport Vector Machine* para estimar el valor de los atributos faltantes.

  Por su parte, los datos erróneos (también conocidos como datos ruidosos) pueden venir de dos formas [CAB11]: ruido de clase cuando la instancia está mal clasificada y ruido de atributo cuando uno o más valores de los atributos en una instancia están distorsionados y no representan la realidad. Para tratar los datos ruidosos se puede usar 3 métodos: construir algoritmos de DM que no se vean afectados en cierta medida ante el ruido (sean robustos), pulir los datos [Ten99] de tal manera que se corrijan los errores y por último se puede identificar los datos ruidosos para eliminarlos del conjunto y así quedarse sólo con datos correctos [BF99].

- **Transformación de datos [GLH16]:** se centra en aplicar fórmulas matemáticas a los valores de los atributos para así obtener valores sintéticos que pueden proporcionar más información respecto a la instancia y al conjunto que pertenecen, las transformaciones más comunes son la lineal y la cuadrática.

- **Integración de los datos [GLH16, BLN86]:** consiste en la unión de los conjuntos de datos provenientes de distintas fuentes en un único conjunto. La integración tiene que tomar en cuenta algunos aspectos que se pueden presentar

durante el proceso, entre ellos están la redundacia de atributos, la cual sucede cuando 2 atributos están fuertemente correlacionados. La redundancia de atributos puede traer consigo un sobre ajuste (*overfitting* en inglés) de los modelos predictivos, además de aumentar el tiempo de cómputo de los mismos, es por eso que se debe eliminar esta redundancia y para ello se usa una prueba de correlación χ^2 con el fin de identificar los atributos redundantes y así decidir con cual quedarse.

Continuando, con los problemas que se pueden presentar al momento de la integración, se tiene también la duplicación de instancias, problema que normalmente trae consigo la inconsistencia en los valores de los atributos, debido a las diferencias con las que se registran los valores. Para solucionar este asunto primero se tiene que identificar las instancias duplicadas usando técnicas que midan la similitud entre ellas, como la propuesta de *Fellegi, I. & Sunter, A.* [FS69] que lo modela como un problema de inferencia bayesiana o como en [CKLS01] donde se usan árboles de clasificación y regresión (CART por sus siglas en inglés) para cumplir este trabajo.

- **Normalización de datos [GLH16]:** busca cambiar la distribución de los datos originales de tal manera que se acoplen a las necesidades de los algoritmos predictivos. Dos de los tipos de normalización más usadas son la normalización min-max y la normalización *z-score*.

Pasando a la reducción de los datos, se tiene que engloba todas las técnicas que reducen el conjunto de datos original para obtener uno representativo con el cual trabajar en los modelos predictivos. La reducción de datos cobra especial importancia cuando se tienen conjuntos muy grandes que tienden a elevar en gran medida el tiempo de cómputo de los algoritmos que los van a usar. Las técnicas de reducción de datos son [GLH16]:

- **Discretización de datos [GLH16, GLS$^+$13]:** es el proceso de transformar datos numéricos en datos categóricos, definiendo un número finito de intervalos que representan rangos entre distintos valores consecutivos con el fin de poder tratarlos como valores nominales. Es de especial importancia conseguir el número correcto de intervalos que mantengan la información original de los datos, ya que

muy pocos intervalos puede llegar a ocultar la relación existente entre un rango en específico y una clase dada y muchos intervalos puede llevar a un sobre ajuste [CPSK07]. El principal atractivo de la discretización es que permite utilizar un algoritmo de DM que trabaje principalmente con datos nominales como *Naïve Bayes* [YW09] a partir de datos numéricos. Para un estudio más completo de la discretización se referencia a [GLS$^+$13].

- **Selección de características [GLH16, LM12]:** busca eliminar atributos que sean redundantes o irrelevantes de tal manera que el subconjunto de características restantes mantenga la distribución original de las clases. El proceso de selección de características tiene ventajas, como mantener e incluso mejorar la precisión de los modelos predictivos, reducir los tiempos de cómputo y reducir la complejidad de los modelos resultantes. La búsqueda de un subconjunto de atributos puede realizarse de 3 maneras: búsqueda exhaustiva, búsqueda heurística y métodos no determinísticos. La búsqueda exhaustiva cubre todo el espacio de soluciones, normalmente van probando todas las combinaciones posibles de atributos para conseguir el que mejor se acople a la métrica a optimizar, entre los métodos exhaustivos están *Focus* [AD91], *Automatic Branch & Bound* [LMD98], *Best First Search* [XYC88], entre otros. Por su parte, la búsqueda heurística busca una solución aproximada a la óptima en poco tiempo, entre sus métodos están los propuestos en [DL97, KS96, Bat94]. Por último, están los métodos no determinísticos, de entre los que destacan los algoritmos géneticos, recocido simulado y *Las Vegas Filter* [LS$^+$96].
- **Selección de instancias [GLH16]:** consiste en elegir un subconjunto de las instancias totales manteniendo las características del conjunto original. Es el problema a tratar en este trabajo y se elabora más sobre el mismo en la siguiente sección.

1.2. Selección de Instancias y Selección de Prototipos

La selección de instancias (IS por sus siglas en inglés) consiste en reducir el conjunto de datos dado a un conjunto reducido que va a ser utilizado con un algoritmo

de clasificación o regresión, manteniendo el desempeño del algoritmo como si se usara el conjunto original.

Definición 1. Dado un conjunto de datos X, se tiene que una instancia $X_i = (X_i^1, X_i^2, \ldots, X_i^p)$ donde X_i^j es el atributo j para la instancia X_i con $X_i \in X$ y siendo p el número de atributos. La instancia X_i es de clase Y_j donde $Y_j \in Y$, siendo Y el conjunto de todas las clases definidas con $j \in (1 \ldots q)$ donde q es el número de clases totales. Se divide el conjunto X en un conjunto TR de entrenamiento y un conjunto TS de prueba. El problema de **Selección de Instancias** consiste en conseguir un conjunto $S \subseteq TR$ con el cual, al usarse con el clasificador T se obtengan los mismos valores de precisión o mejores que al usar T con TR [GLH16].

La respuesta óptima de un método de selección de instancias es un conjunto *consistente* y de cardinalidad mínima.

Definición 2. "Un conjunto R es **consistente** con T, *si y solo si* toda instancia $t \in T$ es clasificada correctamente mediante el uso de un clasificador M y las instancias en R como conjunto de entrenamiento." [Ale14]

Sin embargo, conseguir la respuesta óptima es un problema NP-Duro (*NP-Hard*) como lo demuestra *Zukhba, A.* en [Zuk10]. Por lo tanto, la mayoría de los métodos propuestos hasta la fecha se enfocan en obtener una solución aproximada.

El problema de selección de instancias se puede enfocar como un problema de selección de prototipos (PS por sus siglas en inglés). PS es en esencia IS con el detalle de que el clasificador T usado es un clasificador basado en instancias [GLH16], de los cuales K Vecinos más Cercanos (KNN por sus siglas en inglés) es el más conocido. En este trabajo se usa 1-NN como clasificador.

1.2.1. Regla de K vecinos más cercanos

Inicialmente propuesta por *Fix, E. & Hodges, J.* en [FHJ51]. La regla KNN clasifica instancias a partir de los datos adyacentes; esto viene dado bajo el razonamiento de

que una instancia probablemente comparta la misma clase que sus vecinos. Formalmente, el algoritmo de clasificación usando KNN se puede definir como:

Definición 3. Sea X un conjunto de datos con $X_i \in X$ una instancia del conjunto, con clase $Y_{X_i} \in Y$ la clase a la cual pertenece, siendo Y el conjunto de las clases presentes en los datos. Sea $\pi_1(X_i) \dots \pi_n(X_i)$ un reordenamiento de las n instancias que conforman el conjunto X de acuerdo a la distancia a la que se encuentren de la instancia X_i, usando una métrica de distancia dada $\rho : \chi \times \chi \to \mathbb{R}$, donde χ es el dominio de las instancias en X, tal que $\rho(X_i, \pi_k(X_i)) \leq \rho(X_i, \pi_{k+1}(X_i))$. Para clasificar una instancia X_j se usa la clase de la mayoría perteneciente al conjunto $\left\{Y_{\pi_i(X_j)} \mid i \leq k\right\}$ siendo k el número de vecinos que se toma en consideración. [SSBD14]

Lo simple del algoritmo ha impulsado KNN a ser uno de los algoritmos de DM más usados y ha inspirado numerosos estudios sobre el comportamiento de convergencia y acotaciones sobre el error en la clasificación. Entre dichos trabajos se encuentra el de *Cover, T. & Hart, P.* [CH67] donde muestran que la probabilidad de error R del clasificador NN está acotada por debajo por la probabilidad de error de Bayes R* y acotada por arriba por $R^*(\frac{2-MR^*}{M-1})$ cuando el número de instancias tiende al infinito y además la regla NN es admisible en la clase de reglas KNN, esto quiere decir que no hay $k \neq 1$ para el cual la probabilidad de error R sea menor que para $k = 1$. Para un estudio más formal de las propiedades de convergencia se refiere a [DGL13].

Una implementación ingenua de KNN dado una instancia a clasificar q, consta de calcular la distancia de todos los puntos con respecto a q y reportar los k puntos más cercanos; esto tiene una complejidad de $O(dn)$ donde d es el número de atributos en una instancia y n es el total de instancias [SDI06]. Es por eso que mucho de los esfuerzos de la investigación de KNN es encontrar estructuras de ordenado y almacenamiento que lleven la complejidad a un orden sublineal o inclusive logarítmico; entre ellas están:

- **Árboles KD [SDI06, Ben75]:** también conocidos como *KD Trees* en inglés, se construyen de la siguiente manera: dado n puntos en un conjunto P en un espacio d-dimensional, primero se calcula la mediana M de los valores del i-ésimo atributo de los n puntos (inicialmente i = 1) y con este valor M se particiona el conjunto P en P_L como el conjunto con puntos cuyo valor del i-ésimo atributo es menor a M y P_R como el conjunto de puntos cuyo valor del i-ésimo atributo

es mayor o igual a M. En la siguiente iteración se elige otro atributo i y se particionan P_L y P_R en dos cada uno. El proceso se repite hasta que el conjunto de puntos en un nodo del árbol construido llegue a tener cardinalidad 1. El tiempo de construcción del árbol es de O(nlog(n)) y el tiempo de búsqueda en G(d)log(n) dado una función G la cual es exponencial en d, cabe destacar que el tiempo de búsqueda es al lo sumo O(dn).

- **Árboles de esfera [SDI06, Omo89, Uhl91]:** los árboles de esfera (*balltrees* en inglés) son árboles binarios donde las hojas corresponden a las instancias y cada nodo interior del árbol corresponde a una esfera en el espacio de los datos, cada esfera requiere ser la más pequeña que contenga las esferas asociadas a los nodos hijos. En contraste con los árboles KD, las regiones asociadas entre nodos vecinos en los árboles de esfera pueden intersectarse y no tienen que cubrir la totalidad del espacio, lo que permite una cobertura más flexible que refleje la estructura inherente a los datos.

- **Hashing sensitivo a la localidad [SDI06, Ind04]:** la idea principal detrás del Hashing Sensitivo a la localidad (LSH por sus siglas en inglés) es realizar un hashing con los datos usando varias funciones de hash de tal manera que la probabilidad de colisión entre dos puntos sea mayor mientras más cerca estén uno del otro usando una métrica de distancia. Entonces, una vez construida la tabla de hash, se puede determinar los vecinos más cercanos retornando los elementos en el contenedor correspondiente a su valor calculado por la función de hash.

Un concepto que está presente al momento de clasificar usando KNN es la llamada maldición de dimensionalidad, ésta afecta la clasificación de dos maneras: la primera estipula que un pequeño incremento en las dimensiones de los datos trae consigo un gran aumento en el número de instancias necesarias para mantener la misma precisión del clasificador. La segunda, por su parte, menciona que para métodos de almacenamiento como los árboles KD y los árboles esfera un aumento en las dimensiones tiende a degradar su desempeño a una búsqueda lineal como la implementación ingenua [KM17]. Lo cual afecta la aplicabilidad del algoritmo a datos de muy grandes dimensiones y abre un campo de estudio continuo a formas de optimización del ordenamiento y almacenamiento de las instancias.

1.2.2. Taxonomía del problema de selección de prototipos

En este trabajo se adopta la taxonomía propuesta por *García, S. et al.* en [GDCH12]. En ella se definen unas propiedades comunes de todos los algoritmos de PS con los que se pueden comparar y así establecer la taxonomía. Sea TR el conjunto de entrenamiento y S el conjunto reducido, las propiedades son las siguientes:

1.2.2.1. Dirección de búsqueda

- **Incremental:** se empieza con un conjunto vacío S y se va añadiendo instancias de TR si cumple con cierto criterio. El orden de presentación de las instancias puede llegar a afectar el resultado final para muchos algoritmos, por eso se acostumbra a presentar los datos de manera aleatoria. Una búsqueda incremental tiene la ventaja de que puede seguir agregando instancias una vez finalizado un proceso de selección inicial, lo cual lo hace bastante atractivo para el aprendizaje continuo
- **Decremental:** la búsqueda empieza con S = TR y se va seleccionando instancias para remover de S. El orden de presentación sigue siendo importante, pero a diferencia de los métodos incrementales, se tiene todo el conjunto desde el inicio. Los algoritmos decrementales tienden a presentar un mayor costo computacional que los incrementales y el aprendizaje no puede continuar luego de terminar el lote inicial.
- **Por lote:** se elige un grupo y se evalúan todos los elementos del mismo para su eliminación, los que no pasen la prueba seleccionada son desechados a la vez. El proceso se repite con distintos lotes hasta terminar.
- **Mixto:** S empieza como un subconjunto preseleccionado (puede ser de manera aleatoria o usando un proceso incremental/decremental) e iterativamente puede añadir o remover instancias que cumplan con criterios en específico.
- **Fijo:** el número final de instancias en S se fija al principio de la fase de aprendizaje y se aplica una búsqueda mixta hasta cumplir con dicha cuota.

1.2.2.2. Tipo de selección

- **Condensación:** se busca mantener los puntos bordes (aquellos que están cercas de las fronteras entre las clases). El razonamiento es que son los puntos bordes los que realmente determinan las fronteras, siendo más útiles al momento de clasificar una nueva instancia. Estos métodos tienden a reducir bastante el conjunto original ya que hay menos puntos bordes que interiores.
- **Edición:** los métodos de edición en cambio buscan remover los puntos bordes, suavizando las fronteras bajo la idea de que es el lugar donde se concentran la mayor cantidad de puntos ruidosos. Tienden a disminuir en menor medida el conjunto TR en comparación a los métodos de condensación.
- **Híbridos:** su principal objetivo es mantener la precisión del clasificador usando un conjunto lo más reducido posible. Para esto eliminan tanto puntos internos como los ruidosos en el borde, tomando las ideas principales de los métodos de condensación y edición.

1.2.2.3. Evaluación de la búsqueda

- **Filtro:** son los métodos que usan un conjunto parcial de datos para decidir cuáles remover o añadir sin usar un esquema de validación, donde se deja uno por fuera para probar con el resto de los datos en cada iteración del algoritmo. La simplificación en la validación cruzada y el uso de un subconjunto de TR agiliza los cálculos realizados por el algoritmo a costa de precisión.
- **Envolventes:** usan todo el conjunto TR en un proceso de validación cruzada, donde en cada iteración se va excluyendo cada instancia para evaluar si vale la pena eliminarla. Son métodos más costosos que los filtros, pero tienden a obtener una precisión mayor al momento de generalizar usando un algoritmo de DM.

A continuación se presenta en la la figura 1.1 la clasificación que se le puede dar a los algoritmos. Para un estudio más extenso sobre los distintos algoritmos se recomienda leer [GLH16]

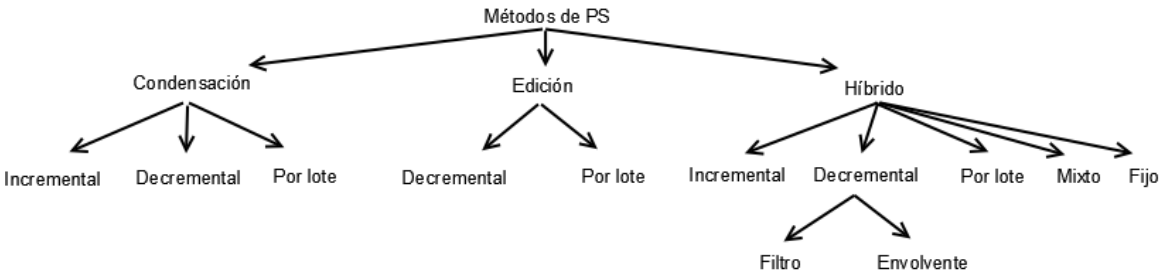

FIGURA 1.1: Taxonomía para los métodos de selección de prototipos

1.2.3. Heurísticas

En esta sección se exponen las heurísticas utilizadas en este trabajo. Citando a *Pearl, J.* en [Pea84]: "Una heurística es un criterio, método o principio para decidir cual, de entre varias alternativas de acciones a seguir, promete ser la más efectiva para alcanzar un objetivo". Para el caso de PS, dicho objetivo es alcanzar un buen aproximado del conjunto de cardinalidad mínima y máxima precisión en la clasificación. Sea $S \subseteq TR$ el conjunto reducido a devolver y TR el conjunto de entrenamiento:

1.2.3.1. Condensed Nearest Neighbor (CNN)

Propuesto inicialmente por *Hart, P.* en [Har68],CNN es un método de condensación incremental. El conjunto S se construye de tal manera que cada elemento de TR está más cerca de un miembro de S de la misma clase que un miembro de S de clase distinta. El algoritmo empieza seleccionando una instancia aleatoria x y se coloca en S (inicialmente vacío), acto seguido se empieza a clasificar todas las instancias de TR sólo usando como referencia las pertenecientes S, si una instancia es clasificada incorrectamente, se agrega a S, asegurando así que en la siguiente vuelta sea clasificada correctamente. Una vez aumentado S se vuelve a probar cada instancia de TR y se agregan las que sean mal clasificadas. El proceso se repite hasta que no existan instancias en TR que se encuentren mal clasificadas. El algoritmo se presenta en 1.1.

Algoritmo 1.1 CNN

Input: `TR` conjunto de entrenamiento, `k` número de vecinos a ser considerado en la clasificación
Output: `S` conjunto reducido

```
1: S ← instancia aleatoria x
2: flag ← false
3: while ¬flag do
4:     for all x ∈ TR do
5:         Y ← k vecinos más cercanos a x pertenecientes a S // Si | S |< k elegir | S | elementos
6:         Clasificar x con la misma clase que sea mayoría en Y
7:         if x está mal clasificada then
8:             S ← S ∪ {x}
9:             Retornar a 3
10:    if Todas las instancias en TR fueron bien clasificadas then
11:        flag ← true
12: return S
```

1.2.3.2. Edited Nearest Neighbor (ENN)

Propuesto por *Wilson, D.* en [Wil72] ENN es un método de edición decremental. Empieza con S = TR y se va iterando sobre las instancias de S, removiendo aquellas que no concuerdan con la clase de la mayoría de sus k vecinos más cercanos. El algoritmo se presenta en 1.2

Algoritmo 1.2 ENN

Input: `TR` conjunto de entrenamiento, `k` número de vecinos a ser considerado en la clasificación
Output: `S` conjunto reducido

```
1: S ← TR
2: for x ∈ S do
3:     Y ← k vecinos más cercanos a x pertenecientes a S
4:     if la clase de x es distinta a la clase mayoritaria en Y then
5:         Se elimina x de S
6: return S
```

1.2.3.3. Relaxed Selective Subset (RSS)

Propuesto por *Flores, A. & Mount, D.* en [FM17] se tiene que RSS es un algoritmo híbrido incremental con la particularidad de que no es sensible al orden de

presentación de las instancias, porque realiza un ordenamiento inicial de las mismas. El método primero ordena las instancias según la distancia que tengan a su enemigo más cercano (la instancia más cercana con clase distinta) de manera incremental (de la distancia más corta a la más larga). Luego, empezando con un conjunto S vacío, se van presentando las instancias en el orden establecido anteriormente y se agrega a S aquellas para las cuales no exista un punto $r \in S$ que esté a una distancia menor que la distancia que tiene r a su enemigo más cercano. Sea $d_{NE}(p)$ la distancia del punto p a su enemigo más cercano y sea $d(p_i, r)$ la distancia de un punto p_i a un punto r. El algoritmo se presenta en 1.3

Algoritmo 1.3 RSS

Input: `TR` conjunto de entrenamiento
Output: `S` conjunto reducido
1: $S \leftarrow \emptyset$
2: Sea $\{p_i\}_{i=1}^{n}$ los puntos en TR ordenados de manera ascendente respecto a $d_{NE}(p_i)$
3: **for all** $p_i \in TR$ **do**
4: **if** $\neg\exists r \in S$ tal que $d(p_i, r) < d_{NE}(r)$ **then**
5: $S \leftarrow S \cup \{p\}$
6: **return** S

1.3. Meteheurísticas

Las metaheurísticas son métodos de optimización que buscan una solución aproximada. Según *Dorigo, M. et al.* en [DBS17] una metaheurística es un algoritmo que pueden ser utilizado para resolver una gran variedad de problemas de optimización sin necesitar muchos cambios estructurales al adaptarlo a cada problema. Al momento de diseñar una metaheurística se debe tomar en cuenta dos conceptos: intensificación y diversificación [Tal09]. En un proceso de intensificación, las regiones en el espacio de soluciones prometedoras son exhaustivamente revisadas con la esperanza de conseguir mejores soluciones. En un proceso de diversificación, las regiones no exploradas son visitadas para poder abarcar distintos lugares en el espacio de soluciones y así evitar que la exploración se estanque en una región específica. Las metaheurísticas se pueden clasificar como metaheurísticas basadas en una única solución o metaheurísticas basadas en una población [Tal09]. Para estudiar los distintos métodos, primero se necesita

definir una serie de conceptos que son comunes para todos:

Definición 4. La **representación del problema** es la manera de codificar las soluciones pertenecientes al espacio de soluciones. Debe ser acorde al problema de tal manera que cumpla con las siguientes características: debe ser completo, es decir, todas las soluciones del espacio deben poder ser codificadas; debe ser conexo, lo que se traduce a que debe haber un camino entre dos cualesquiera soluciones y por último, debe ser eficiente, de tal manera que la manipulación por los operadores de búsqueda tenga un costo en tiempo y espacio mínimo [Tal09].

Definición 5. La **función objetivo** (también conocida como función de costo o de utilidad) $\mathcal{F}$ asocia a cada solución un valor real que describe la calidad de la solución: $\mathcal{F} : S \to \mathbb{R}$, donde S es el espacio de soluciones. Con la función objetivo se guía la búsqueda hacia "buenas" soluciones en el espacio [Tal09].

Definición 6. La **vecindad** es el conjunto de soluciones N(s) que se le asocia a una solución s por una función de vecindad $N : S \to 2^S$. Donde S es el espacio de soluciones. Una solución $s' \in N(s)$ se conoce como **vecina** de s y se obtiene de realizar una pequeña perturbación a s con un operador de movimiento [Tal09].

1.3.1. Metaheurísticas basadas en una única solución

También conocidas como metaheurísticas de trayectoria, se centran en mejorar una única solución que van cambiando a lo largo del curso del algoritmo; se pueden ver como trayectorias de búsqueda en el espacio de soluciones, dichas trayectorias son trazadas por procesos iterativos que se mueven de una solución a otra dependiendo del criterio de aceptación particular de la metaheurística utilizada. Esta clase de metaheurísticas se enfocan principalmente en la explotación del espacio de soluciones e introducen el componente de exploración de distintas formas. Entre ellas se encuentra la búsqueda local [Tal09, AL03], el recocido simulado [Tal09, KGV83], la búsqueda tabú [Tal09, Glo89], Búsqueda Local Iterada (ILS) [LMS03], Búsqueda de Vecindad Variable (VNS) [MH97],Búsqueda Local Guiada (GLS) [Vou98], GRASP [FR95], entre

otros. De especial importancia está la búsqueda local, la cual juega un papel importante en la mayoría de las metaheurísticas de trayectoria y en algunas metaheurísticas poblacionales:

Definición 7. Búsqueda Local [Tal09, AL03]: es la base para la mayoría de las metaheurísticas basadas en una única solución. Dado una solución inicial, en cada iteración el método reemplaza la solución actual por un vecino que mejore el valor de la función objetivo, la búsqueda termina cuando todos los vecinos candidatos a evaluarse son peores que la solución actual, lo cual significa que un óptimo local ha sido alcanzado. Para vecindades muy grandes se puede restringir el conjunto a un porcentaje dado para agilizar las operaciones. Otro de los factores que se debe elegir al momento de implementar búsqueda local es si en cada iteración se elige el mejor elemento de la vecindad, el primer vecino encontrado que mejore la solución actual o inclusive un vecino aleatorio al momento de reemplazar la respuesta actual. El principal problema de este método es que se queda atrapado en un óptimo local y no hay forma de salir del mismo, esto se debe a que la búsqueda local es un método únicamente de intensificación; sin embargo, los métodos que se derivaron de éste utilizan técnicas de diversificación para escapar del óptimo local.

1.3.2. Metaheurísticas basadas en una población

Estas metaheurísticas empiezan con una población inicial de soluciones, que puede ser elegida de manera aleatoria o con heurísticas que introduzcan "buenas" soluciones, e iterativamente generan nuevos elementos que pueden llegar a suplantar los de la población actual según un criterio de selección. El proceso de generación y selección se repite hasta que se cumpla un criterio de parada, el cual puede ser un número de iteraciones fijas o hasta que la población converga a una región sin mejoras pasado un número de iteraciones. Dichos procesos de generación y selección pueden ser sin memoria, es decir, solo dependen de la población actual, como el caso de los algoritmos genéticos tradicionales o pueden ser con memoria y usar información adquirida durante el proceso de búsqueda para dirigir la generación y selección a mejores resultados.[Tal09]. Un esquema general para los algoritmos basados en una población es el presentado en 1.4.

Algoritmo 1.4 Plantilla para las metaheurísticas basadas en una población

1: $P \leftarrow P_0$ // se genera la población inicial
2: $t = 0$
3: **repeat**
4: Generar(P'_t) // se genera una nueva población
5: P_{t+1} = seleccionar nueva población entre $P_t \cup P'_t$
6: $t = t + 1$
7: **until** Se satisface criterio de parada
8: **return** mejor solución encontrada

Entre las metaheurísticas basadas en una población se encuentran: *Scatter Search* [Tal09, Glo77], colonia de hormigas [Tal09, Dor92], optimización de enjambre de partículas [Tal09, ESK01], algoritmos de estimación de distribución [Tal09, LLIB06], Evolución Diferencial [Tal09, PSL06], Algoritmos evolutivos [Tal09], entre otros.

1.3.2.1. Algoritmos evolutivos

Los algoritmos evolutivos están basadas en la competencia entre individuos de una población llamados cromosomas; la población se inicializa con cromosomas elegidos aleatoriamente o a través de heurísticas. Con esto, dado una función objetivo, se evalúa cuán bueno es cada cromosoma y con esta información se decide por medio de un proceso de selección cuáles serán los cromosomas que se van a cruzar, dando como resultado uno o más hijos que comparten características de sus padres. Luego del cruce, viene la mutación de los nuevos individuos con un operador definido que perturba ligeramente al cromosoma. Por último viene un proceso de reemplazo donde se decide si los hijos suplantan algún elemento de la población (esquema estacionario) o si se construye una nueva población con los hijos que va a suplantar totalmente a sus padres (esquema generacional) [Tal09].

El diseño de un algoritmo evolutivo viene dado con la toma de decisiones respecto a algunos componentes. Algunos comunes a todas las metaheurísticas como la representación del problema, el cual puede ser un vector de valores binarios, enteros, reales, una permutación, entre otros; la inicialización de la población, que puede ser por medio de heurísticas o aleatoria y la elección de una función objetivo que represente cuán buena es un cromosoma y el criterio de parada. Por otro lado, hay unos

componentes que son propios de los algoritmos evolutivos como el criterio de selección para reproducirse, el operador de cruce, el operador de mutación y la estrategia de reemplazo.

Cano, J. en [dA04] hace un estudio comparativo de varias algoritmos evolutivos con respecto a las heurísticas tradicionales de PS. En este trabajo compara un algoritmo genético generacional (GGA), un algoritmo genético estacionario (SSGA), CHC y aprendizaje incremental basado en población (PBIL); aquí obtuvo que los algoritmos evolutivos obtiene buenos resultados y CHC en especial supera a los métodos tradicionales cuando se evalúa tanto el nivel de reducción como de precisión del clasificador. Por otra parte, *Cano, J. et al.* en [GCH08], plantean un algoritmo memético estacionario que obtiene también muy buenos resultados con respecto a otros algoritmos evolutivos en cuestión de reducción (CHC estando por encima) y con precisión comparable al resto.

Los algoritmos evolutivos que fueron implementados en este trabajo fueron:

1.3.2.1.1 Algoritmo Genético Generacional (GGA)

Generational Genetic Algorithm (GGA) en inglés, es el esquema tradicional de algoritmos genéticos; los algoritmos genéticos fueron desarrollados por *Holland, H.* en [Hol75]. La versión generacional usa una estrategia de reemplazo en la cual se genera una población nueva de hijos en cada ciclo del algoritmo y ésta suplanta a la generación anterior. El algoritmo genera una población inicial aleatoria y empieza a crear poblaciones nuevas en cada generación hasta que se cumpla una condición de parada. En medio del proceso está actuando un operador de cruce que mezcla los elementos seleccionados como padres y una operación de mutación que modifica la nueva generación. El algoritmo se presenta en 1.5 [Ale14].

1.3.2.1.2 Algoritmo Genético Estacionario (SSGA)

Steady State Genetic Algorithm (SSGA) en inglés, es otra variación de los algoritmos genéticos. En este caso, la estrategia de reemplazo consiste en generar uno o dos hijos por iteración y decidir al momento si va a suplantar algún elemento de la

Algoritmo 1.5 Algoritmo Genético Generacional

Input: `pop` tamaño de la población, `cp` probabilidad de cruce, `mp` probabilidad de mutación
Output: Una solución al problema

$P \leftarrow$ Generar población aleatoria de `pop` individuos
$s^* \leftarrow$ el *mejor* individuo en P
while ¬ Condición de parada **do**
    $P' \leftarrow \emptyset$
    while $\mid P' \mid <$ `pop` **do**
        $p_1 \leftarrow$ Seleccionar un individuo en P
        $p_2 \leftarrow$ Seleccionar un individuo en P
        $c_1, c_2 \leftarrow$ recombinar p_1 y p_2 con probabilidad `cp`
        Mutar c_1 y c_2 con probabilidad `mp`
        $P' \leftarrow P' \cup \{c_1, c_2\}$
    $P \leftarrow P'$
    if El *mejor* individuo en P es *mejor* que s^* **then**
        $s^* \leftarrow$ el *mejor* individuo en P
return s^*

población; puede suplantar a uno de los padres si es mejor que uno de ellos o puede suplantar al peor elemento de la población. Al igual que GGA, se tiene que definir un operador de mutación y cruce. El algoritmo se presenta en 1.6 [Ale14].

Algoritmo 1.6 Algoritmo Genético Estacionario

Input: `pop` tamaño de la población, `cp` probabilidad de cruce, `mp` probabilidad de mutación
Output: Una solución al problema

$P \leftarrow$ Generar población aleatoria de `pop` individuos
$s^* \leftarrow$ el *mejor* individuo en P
while ¬ Condición de parada **do**
    $p_1 \leftarrow$ Seleccionar un individuo en P
    $p_2 \leftarrow$ Seleccionar un individuo en P
    $c_1, c_2 \leftarrow$ recombinar p_1 y p_2 con probabilidad `cp`
    Mutar c_1 y c_2 con probabilidad `mp`
    Seguir algún criterio de reemplazo de individuos en P por c_1 y c_2
    if El *mejor* individuo en P es *mejor* que s^* **then**
        $s^* \leftarrow$ el *mejor* individuo en P
return s^*

1.3.2.1.3 Algoritmo Memético (MA)

Memetic Algorithm en inglés, es un algoritmo evolutivo basado en los algoritmos genéticos que tiene la peculiaridad de tener un proceso de optimización interno llamado "meme", el cual es aplicado a todos o algunos cromosomas de la población en cada iteración; el meme más común es una búsqueda local [NC12]. El esquema clásico se basa en los GGA y primero genera una población nueva con los cruces y mutaciones propios de un GGA, para luego pasar a una fase de intensificación donde aplica el meme a todas las soluciones y se genera una nueva población optimizada que suplanta la generación anterior. Otro esquema se basa en los SSGA y en cada iteración se cruzan una serie de padres para generar uno o dos hijos que, luego de mutar con cierta probabilidad dada, se decide si pasan a un proceso de optimización con el meme y el resultado se decide si se incorpora a la población. En 1.7 se presenta el algoritmo para la versión estacionaria de los algoritmos meméticos (SSMA).

Algoritmo 1.7 Algoritmo Memético Estacionario

Input: `pop` tamaño de la población, `cp` probabilidad de cruce, `mp` probabilidad de mutación, `mem` meme usado
Output: Una solución al problema

```
1: P ← Generar población de pop individuos
2: s* ← el mejor individuo en P
3: while ¬ Condición de parada do
4:     p1 ← Seleccionar un individuo en P
5:     p2 ← Seleccionar un individuo en P
6:     c1, c2 ← recombinar p1 y p2 con probabilidad cp
7:     Mutar c1 y c2 con probabilidad mp
8:     Determinar si c1 y c2 van a ser optimizados con mem y almacenar el resultado en c'1 y c'2
9:     Seguir algún criterio de reemplazo de individuos en P por c'1 y c'2
10:    if El mejor individuo en P es mejor que s* then
11:        s* ← el mejor individuo en P
12: return s*
```

1.3.2.1.4 CHC Adaptative Search Algorithm

Propuesto inicialmente por *Eshelman, L.* en [Esh91], es un algoritmo evolutivo generacional con la diferencia de que es totalmente elitista, ya que elige los mejores

n elementos de entre la vieja y nueva población para conformar la nueva generación (n es el número de cromosomas en la población). También tiene la particularidad de que implementa un operador de cruce llamado HUX en el cual, dado dos padres, intercambia la mitad de los genes que no coincidan entre ellos de manera aleatoria con el fin de crear hijos lo más distinto posible de los padres. Además CHC tiene un mecanismo de prevención de incesto en el cual se usa la distancia de Hamming entre los dos posibles candidatos a ser padres para determinar si son lo suficientemente distintos para cruzarse, para esto usa un umbral que inicialmente es L/4 donde L es la longitud del cromosoma. Por último, no existe una operación de mutación y en cambio, cuando pasa una generación sin individuos nuevos, se disminuye el umbral de incesto en 1, hasta que llega a 0 y se toma la decisión de reinicializar la población, preservando el mejor cromosoma encontrado hasta el momento y poblando los cromosomas restantes con variaciones del mejor, donde se perturban hasta un 35 % de los genes asociados al cromosoma. El algoritmo se presenta en 1.8, donde t es la generación actual, d es el umbral de incesto, P(t) es la población de la generación t, L es la longitud del cromosoma.

Algoritmo 1.8 CHC

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

1: $t = 0$
2: $d = L/4$
3: $P(t) \leftarrow$ Generar población de `pop` individuos
4: $s^* \leftarrow$ el *mejor* individuo en P
5: **while** ¬ condición de parada **do**
6: $t = t + 1$
7: $C(t) \leftarrow P(t-1)$
8: **recombinar** los cromosomas en C(t) para formar C'(t)
9: evaluar los cromosomas en C'(t) con la función objetivo
10: seleccionar P(t) de C'(t) y P(t-1) sólo con los mejores cromosomas
11: **if** El *mejor* cromosoma en P es *mejor* que s^* **then**
12: $s^* \leftarrow$ el *mejor* cromosoma en P
13: **if** $P(t) = P(t-1)$ **then**
14: $d \leftarrow d - 1$
15: **if** $d < 0$ **then**
16: **Reinicializar** P(t)
17: **return** s^*

Algoritmo 1.9 Recombinar

Input: C(t) candidatos a padre, d umbral de incesto
Output: C'(t) hijos
  for all par de instancias en C(t) x_1 y x_2 **do**
    $ham \leftarrow$ distancia de hamming entre x_1 y x_2
    if $ham/2 > d$ **then**
      cambiar la mitad de elementos que difieran entre x_1 y x_2 de forma aleatoria para generar x'_1 y x'_2
      $C'(t) \leftarrow C'(t) \cup \{x'_1, x'_2\}$
    else
      borrar el par x_1 y x_2 de C(t)
  return C'(t)

Algoritmo 1.10 Reinicializar

Input: P(t-1) población anterior, s^* mejor solución, r porcentaje de genes a cambiar, d umbral de incesto
Output: P(t) población renovada
  Llenar P(t) con copias de s^*
  for all miembros $x_i \in P(t)$ excepto uno **do**
    Cambiar $r * L$ genes de manera aleatoria de x_i
    Evaluar x_i con la función objetivo
  $d = L/4$
  return P(t)

1.4. Criterios para comparar los métodos de selección de prototipos

Al momento de comparar los distintos métodos de PS, se usan los siguiente criterios para evaluar las fortalezas y debilidades relativas de cada algoritmo [GLH16]:

- **Reducción en el espacio de almacenamiento:** se asocia con la cantidad de instancias que permanecen al final de proceso. La reducción de las instancias trae consigo una disminución en los tiempos de cómputo al tener que revisar menos individuos en cada iteración para clasificar una nueva instancia.

- **Precisión en la generalización:** se espera que aún con el conjunto reducido, se mantenga las tasas de acierto de 1-NN o inclusive, lleguen a mejorar. Un

algoritmo de PS debe poder mantener la precisión al momento de ser evaluado con el conjunto de prueba. La precisión se calcula dividiendo el número de clasificaciones hechas correctamente entre el total de clasificaciones.

- **Tiempo de cómputo:** involucra cuánto tiempo le lleva al algoritmo realizar la reducción de los datos. Un factor importante al momento de escalar los métodos a conjuntos muy grandes, ya que pueden ser poco prácticos si tardan demasiado. En este trabajo el tiempo de cómputo se mide en segundos.

- ***Cohen's Kappa*:** es una métrica que originalmente mide el nivel de acuerdo o desacuerdo entre dos clasificadores. Sin embargo se han hecho adaptaciones de esta métrica para ser usada por un sólo clasificador [GDCH12], ya que es más robusta que la precisión por tomar en cuenta la posibilidad de que una clasificación sea hecha aleatoriamente. *Cohen's kappa* se calcula a partir de la matriz de confusión como se muestra en la ecuación (1.1). Donde y_{ii} es el conteo de las celdas de la diagonal principal, N el número de instancias revisadas, Ω el el número de clases presentes, $y_i.$ es la suma de las celdas de la fila i y $Y_{\cdot i}$ es la suma de las celdas de la columna i.

$$kappa = \frac{N * \sum_{i=1}^{\Omega} y_{ii} - \sum_{i=1}^{\Omega} y_i. * y_{\cdot i}}{N^2 - \sum_{i=1}^{\Omega} y_i. * y_{\cdot i}} \tag{1.1}$$

Capítulo 2

Marco metodológico

En este capítulo se detalla la representación utilizada para los cromosomas, la función objetivo, las adaptaciones particulares que se hizo a cada algoritmo evolutivo usado en el experimento, el proceso de validación cruzada, la técnica de estratificación, se presenta los conjuntos de datos usados para el experimento y se explica el método de entonación utilizado para ajustar los algoritmos evolutivos.

2.1. Representación del cromosoma

Sea T el conjunto de instancias a reducir de tamaño `n`, la representación usada para modelar el problema de selección de prototipos es el de un mapa de bits de tamaño `n`, donde cada bit representa una instancia dentro de T; si el valor del bit i es 1, entonces la instancia $t_i \in T$ está en el conjunto reducido S, si el bit i es 0, t_i se encuentra por fuera de S. En este sentido, el conjunto S_M, que representa el conjunto reducido S dado por el mapa de bits M (que se le conoce también como cromosoma M y a los bits se le conoce como genes), se define como en la ecuación (2.1). Además un ejemplo se presenta en 2.1.

$$S_m = \{t_i \in T \mid i = 1 \ldots n \wedge s_i = 1\} \tag{2.1}$$

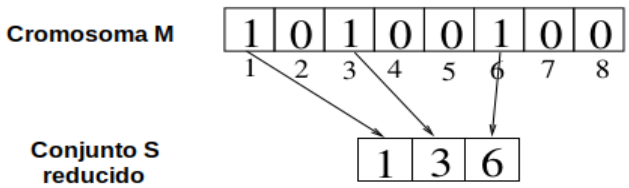

FIGURA 2.1: Representación de un cromosoma y su respectivo conjunto reducido

2.2. Función objetivo

Se necesita una función con la cual los algoritmos evolutivos puedan evaluar cuán buena es una solución dada, además de que dicha función debe permitir establecer una relación de orden entre las soluciones con el fin de decidir cuál cromosoma es mejor que otro. Como se explicó anteriormente, los algoritmos evolutivos buscan aproximarse al óptimo global, que en este caso es el conjunto reducido S con menor cardinalidad posible y mayor precisión en la clasificación de instancias nuevas. Es por eso que se adopta una función objetivo derivada del trabajo de *Cano, J.* en [dA04], la cual se presenta a continuación:

$$\mathcal{F}(S) = \alpha * error(S) + (1 - \alpha) * \left(100 * \frac{\mid S \mid}{TR}\right) \tag{2.2}$$

Donde $\mathcal{F} : W \rightarrow \mathbb{R}$ es la función objetivo, S es el conjunto reducido a evaluar, W es el espacio de cromosomas asociados a los distintas posibilidades de conjuntos reducidos, α es un parámetro que controla cuánta importancia se le da al error asociado a S con respecto a la tasa de reducción del segundo término de la ecuación (2.2), TR es el conjunto de entrenamiento original del cual se realizó la reducción y error(S) es el porcentaje de error al clasificar un conjunto de prueba TS usando 1-KNN con S como conjunto de referencia. El α usado es 0.5 como lo establecen en [dA04] para darle la misma importancia a la reducción de datos como a mantener bajo los porcentajes de error en la clasificación.

Dado esta función objetivo, la meta de todas las metaheurísticas implementadas se vuelve minimizar $\mathcal{F}(S)$, lo cual quiere decir que se busca tanto reducir $\mid S \mid$, como reducir error(S). Una conjunto S_i es mejor que un conjunto S_j si $\mathcal{F}(S_i) < \mathcal{F}(S_j)$.

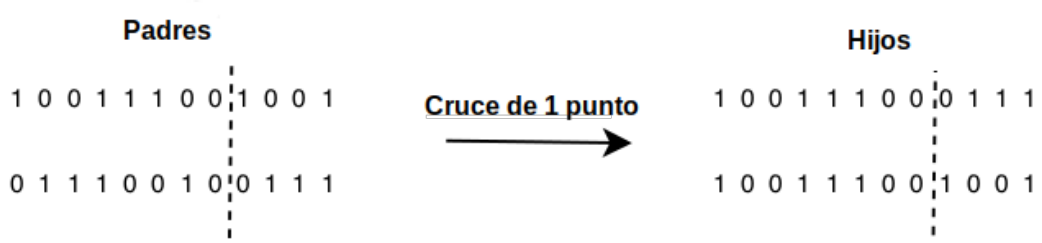

FIGURA 2.2: Cruce de un punto

2.3. Adaptaciones de los algoritmos evolutivos

Para aplicar los distintos algoritmos evolutivos implementados para este trabajo, es necesario determinar los operadores de cruce y mutación, el método de selección de los cromosomas que van a cruzarse, el criterio de selección de los cromosomas sobrevivientes y en caso del algoritmo memético el meme interno utilizado.

Para el caso del algoritmo genético estacionario y el algoritmo memético, se eligió como método se selección de cromosomas a cruzarse un proceso de torneo [Tal09], el cual consiste en elegir k cromosomas de manera aleatoria y se eliminan k-1 donde el sobreviviente es el mejor dentro de los k. CHC por su parte elige dos cromosomas aleatorios y utiliza su mecanismo de prevención de incesto para elegir a los padres. El algoritmo genético generacional simplemente elige dos elementos aleatorios para realizar el cruce dada una probabilidad.

El operador de cruce utilizado en GGA, SSGA y MA es la recombinación de un punto [Tal09], el cual consiste en definir un punto u en el cual se va dividir los dos cromosomas seleccionados como padres x_1 y x_2, luego se forman dos hijos y_1 y y_2 donde la mitad de sus genes provienen de de un padre y la otra mitad del otro padre. CHC en cambio usa el operador HUX explicado anteriormente. En la figura 2.2 se muestra un ejemplo del cruce de un punto.

El operador de mutación para GGA, SSGA y MA consta de cambiar 5 % de los genes del cromosoma de manera aleatoria. Se elige 5 % basado en [Ale14] para que la mutación represente un cambio real en el conjunto que representa el cromosoma, ya que si sólo se cambia un gen, el conjunto mutado sería para los efectos de la optimización

casi idéntico al original. Sin embargo la probabilidad de que un cromosoma dado mute es baja, basado principalmente en los resultados de *Cano, J.* en [dA04] donde obtienen mejores resultados experimentales con bajas probabilidades de mutación (menor al 1 % por cromosoma), justificándose en que con mayores valores, la búsqueda podría degenerar en una búsqueda aleatoria. CHC por su parte no tiene mutación.

El criterio de reemplazo para GGA es generar una población nueva de hijos P_i que va a suplantar la generación anterior P_{i-1} excepto el mejor elemento en P_{i-1}, el cual toma el lugar del peor elemento de P_i en la nueva generación. Por su parte, el criterio de reemplazo de SSGA es que dado dos padres y los dos hijos producidos por el operador de cruce, se eligen los 2 mejores cromosomas para permanecer dentro de la población. MA, en cambio usa un criterio de reemplazo en el cual los 2 hijos suplantan a los 2 peores elementos de la población y CHC se queda con los `n` mejores cromosomas entre P_i y P_{i-1}, ambos casos son totalmente elitistas.

El algoritmo memético es el que más adaptaciones tiene para adecuarse a PS, se usa una adaptación realizado por *Cano, J. et al.* en [GCH08]. Se basa en el algoritmo memético estacionario presentado anteriormente, con la peculiariadad de que para decidir si los hijos producidos en una iteración van a ser optimizados con el meme, se usa un parámetro P_{LS} que se determina de la siguiente forma:

$$P_{LS} = \begin{cases} 1 & \text{si } \mathcal{F}(S_{nuevo}) < \mathcal{F}(S_{peor}) \\ 0.0625 & \text{en caso contrario} \end{cases} \tag{2.3}$$

Donde $\mathcal{F}$ es la función objetivo, S_{nuevo} es el conjunto reducido representado por uno de los cromosomas hijos y S_{peor} es el conjunto reducido representado por el peor cromosoma de la población. Es así como P_{LS} representa la probabilidad con la cual se va a decidir si se optimiza el cromosoma hijo; P_{LS} debe ser calculado para cada hijo creado en el cruce. La idea es que si el hijo es mejor que el peor cromosoma de la población, entonces vale la pena optimizarlo; en cambio, si es peor, se le da una probabilidad de 6,25 %.

El meme usado en MA es el que se presenta en 2.1. El procedimiento consiste en ir reduciendo progresivamente las instancias que se encuentran en el conjunto S,

representado por el cromosoma M, sin que se pierda la precisión asociada a S. Para esto, se usa una lista U del primer vecino más cercano de cada gen en M, una lista R que contiene los genes que ya han sido puestos en 0 y que no generan una ganancia mayor al umbral de aceptación `t`, clase(i) es la clase asociada a la instancia representada por el gen i del cromosoma M, `ganancia` representa cuánto mejora (en caso de que sea positiva) o cuánto empeora (en caso de ser negativa) la solución dada por el cromosoma M luego de cambiar un gen, $fitness_M$ es el valor de evaluar la función objetivo con el cromosoma M y $fitness_{ganancia}$ se define como en la ecuación (2.4), donde L es el largo del cromosoma:

$$fitness_{ganancia} = \frac{\frac{ganancia}{L} * 100 + \frac{100}{L}}{2} \tag{2.4}$$

El meme empieza a revisar los genes del cromosoma M que estén prendidos (con valor 1) y no se encuentren en la lista de revisados R en la línea 4; entonces, mientras se cumplan estas condiciones, se elige un m_j que esté prendido y no esté en R, se apaga (coloca valor cero), se hace una copia de U a U', se actualiza la lista del primer vecino más cercano U tomando en cuenta que m_j ya no pertenece al conjunto reducido S y por cada vecino de U que es actualizado, se actualiza la ganancia, sumando 1 si el gen i estaba mal clasificado anteriormente y con el nuevo vecino se clasifica correctamente (líneas 13 y 14) o restando 1 si el gen i estaba bien clasificado anteriormente y con el nuevo vecino se clasifica incorrectamente (líneas 11 y 12); acto seguido, se actualiza el *fitness* del cromosoma M si la ganancia está por encima del umbral `t` y se limpia la lista de revisados R (líneas 15, 16 y 17) o en caso contrario, se recupera U con la copia U' y se prende de nuevo el gen m_j.

2.4. Conjunto de datos

Los conjuntos de datos utilizados para validar el experimento provienen de *UCI Machine Learning Repository* [DKT17] y *KEEL Data-Mining Software Tool* [AFFL$^+$11]. Se hace una separación como la establecida en [dA04] donde se considera como conjunto de datos pequeños aquellos con menos de 2000 instancias, los conjuntos medianos los que poseen entre 2000 y 20000 instancias y los conjuntos grandes aquellos con más

Algoritmo 2.1 Meme

Input: `M` cromosoma a optimizar, `t` umbral de aceptación
Output: `M` cromosoma optimizado

1: Sea $\texttt{M} = \{m_1, m_2, \ldots, m_n\}$ el cromosoma a optimizar
2: $R \leftarrow \emptyset$
3: $U = \{u_1, u_2, \ldots, u_n\}$ la lista de vecinos asociados, donde u_i es el vecino más cercano del gen i.
4: **while** $(\exists m_i \in \texttt{M} \mid m_i = 1 \wedge i \notin R)$ **do**
5: elegir j aleatoriamente de `M` tal que $m_j = 1 \wedge j \notin R$
6: $ganancia \leftarrow 0$
7: $m_j \leftarrow 0$
8: Copiar U a U'
9: **for all** $u_i \in U \mid u_i = j$ **do**
10: $u_i \leftarrow$ nuevo vecino más cercano con el nuevo `M`
11: **if** $clase(i) = clase(u_i') \wedge clase(i) \neq clase(u_i)$ **then**
12: $ganancia \leftarrow ganancia - 1$
13: **else if** $clase(i) \neq clase(u_i') \wedge clase(i) = clase(u_i)$ **then**
14: $ganancia \leftarrow ganancia + 1$
15: **if** $ganancia \geq \texttt{t}$ **then**
16: $fitness_M \leftarrow fitness_M + fitness_{ganancia}$
17: $R \leftarrow \emptyset$
18: **else**
19: Recuperar U de U'
20: $m_j \leftarrow 1$
21: $R \leftarrow R \cup j$
22: **return** M

de 20000 instancias. En la tabla 2.1 se detallan los conjuntos pequeños, en 2.2 los medianos y en 2.3 los grandes. Solo se eligió conjunto de datos numéricos para poder utilizar la distancia euclideana para 1-NN sin problemas derivados de convertir datos categóricos.

2.5. Validación cruzada y estratificación

Dado un conjunto de datos D, el proceso de validación cruzada [K+95] consta de dividir D en k subconjuntos mutuamente exclusivos $D_1, D_2, \ldots, D_k$ de aproximadamente el mismo tamaño, donde cada subconjunto mantiene la distribución de las clases como se encuentra en D. Luego se procede a probar el clasificador M, que en este caso

Conjunto	Instancias	Atributos	Clases
Iris	150	4	3
Cleveland	297	13	5
Led7Digit	500	7	10
Pima	768	8	2
WDBC	569	30	2
Monk-2	432	6	2
Wisconsin	683	9	2
Wine	178	13	3
Glass	214	9	7
Banknote	1372	5	2

Tabla 2.1: Conjuntos de datos pequeños

Conjunto	Instancias	Atributos	Clases
Banana	5300	2	2
Cardiotocography	2126	23	3
Eye-state	14980	15	2
Page-blocks	5473	10	5
Penbased	10992	16	10
Satimage	6435	36	7
Thyroid	7200	21	3
Segment	2310	19	7

Tabla 2.2: Conjuntos de datos medianos

Conjunto	Instancias	Atributos	Clases
Credit-card	30000	24	2
Shuttle	58000	9	7

Tabla 2.3: Conjuntos de datos grandes

es 1-NN, k veces, donde en cada prueba $t \in \{1, 2, \ldots, k\}$ se utiliza como conjunto de entrenamiento $TR = D \setminus D_t$, se aplica el algoritmo de selección de prototipos a TR y el conjunto resultante S se valida usando $TS = D_t$ como conjunto de prueba. El porcentaje de aciertos del clasificador se calcula como el promedio de las k pruebas realizadas; pero como las metaheurísticas tienen un componente estocástico, se necesita repetir cada prueba t varias veces. En este trabajo se decide por k = 10 y se repite cada prueba t 3 veces basándose en el trabajo de *Cano, J.* en [dA04]. Este esquema de validación cruzada es la forma clásica del método y se aplica a los conjuntos de tamaño pequeño como lo hacen en el trabajo antes citado. En la figura 2.3 se muestra un esquema de cómo se aplica la validación cruzada para el problema de selección de

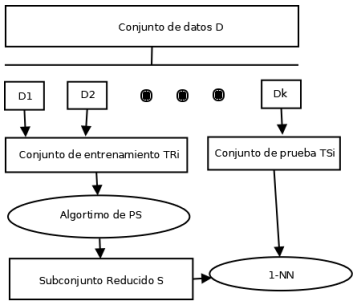

FIGURA 2.3: Validación cruzada clásica

instancias dado que la partición k es seleccionada como conjunto de prueba.

Por otra parte está la estratificación, la cual es una técnica propuesta por *Cano, J. et al.* en [CHL05] para solventar el problema de aplicación de los algoritmos de PS frente a conjunto de datos muy grandes. Dicho problema viene dado porque la mayoría de los algoritmos de PS y metaheurísticas utilizadas son $O(n^2)$, siendo n la cantidad de instancias del conjunto a procesar, y por lo tanto, para grandes volúmenes de datos estos algoritmos empiezan a tardar mucho en computar una solución, lo cual los vuelve poco útiles al momento de hacer preprocesamiento de datos. Es así que la estratificación se adopta como una técnica que lleva a tiempos aceptables el cómputo con conjuntos de muchas instancias.

Dado un conjunto de datos D, la estratificación empieza dividiendo D en k subconjuntos, llamados estratos, mutuamente exclusivos $D_1, D_2, \ldots, D_k$ de aproximadamente el mismo tamaño y preservando la distribución de clases presente en D. Luego, a diferencia de la validación cruzada clásica que forma un conjunto TR a partir de k-1 subconjuntos D_i, para luego aplicar el algoritmo de PS, en la estratificación se aplica el algoritmo de PS directamente a cada uno de los $k-1$ subconjuntos seleccionados para el entrenamiento, formando entonces subconjuntos reducidos $DS_1, DS_2, \ldots, DS_{t-1}, DS_{t+1}, \ldots, DS_k$, donde D_t es el conjunto seleccionado como conjunto de prueba; acto seguido, se juntan todos los DS_i para formar el conjunto reducido S que va a ser usado por 1-NN para clasificar D_t. La estratificación prueba ser un método efectivo, como lo demuestran en [CHL05], ya que reduce considerablemente la cantidad de instancias que debe tratar el algoritmo de PS a $\frac{N}{k}$, por lo que la elección del número de estratos k se vuelve de especial importancia. Para este trabajo se adopta k = 10 para los conjuntos medianos y k = 50 para los conjuntos grandes,

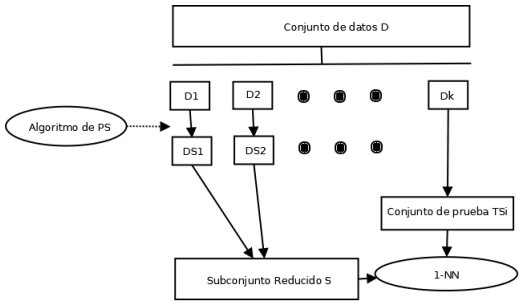

FIGURA 2.4: Estratificación

tal y como se determinan en [CHL05], cuya idea es hacer que cada algoritmo de PS no trabaje con más de 2000 instancias por estrato para reducir la cantidad a un conjunto de tamaño pequeño según la clasificación anteriormente expuesta. Además, al igual que en la validación cruzada clásica, las metaheurísticas utilizadas son estocásticas y por lo tanto, cada una de las k pruebas realizadas se repite 3 veces, regresando el promedio de todas las pruebas realizadas como resultado. En la figura 2.4 se muestra un esquema de cómo se aplica la estratificación, donde el estrato k es seleccionado como conjunto de prueba.

2.6. Entonación de las metaheurísticas

Para la entonación de las metaheurísticas se uso *irace* [LIDLC+16], el cual es un paquete de R que implementa el método de entonación automática conocido como *iterated F-race*, el cual forma parte de los métodos de filtro según la taxonomía propuesta por *Eiben, A. & Smit, S.* en [ES11]. Como método de filtro, su principal objetivo es ir reduciendo los vectores de parámetros (configuraciones) a entonar paulatinamente, usando una competencia continua en la cual, cuando existen suficientes pruebas estadísticas de que una configuración genera una utilidad (evaluación de la función objetivo) con menos valor que el resto de las configuraciones, se elimina de las pruebas. Para mayor información respecto a la taxonomía de los métodos de entonación automática se recomienda leer [ES11].

Iterated F-race en específico trata de un método que consiste en tres pasos: (1) elegir varias configuraciones posibles de acuerdo a una distribución en particular y

así formar una población, (2) elegir las mejores configuraciones de la población por medio de un proceso de carrera y (3) actualizar la distribución de la población de tal manera que se sesgue a favor de las mejores configuraciones. Estos tres pasos son repetidos hasta que un criterio de parada es cumplido. El algoritmo de *iterated F-race* se presenta en 2.2.

Algoritmo 2.2 IRACE

Input: `I` conjunto de instancias del problema a entonar, `X` espacio de configuraciones, U función de utilidad, `B` presupuesto con el que se cuenta
Output: θ^{elite} mejores configuraciones encontradas

1: $\theta_1 \leftarrow$ una muestra uniforme de X
2: $\theta^{elite} \leftarrow$ resultado de una **carrera** usando θ_1 como población y con presupuesto B_1
3: $j \leftarrow 1$
4: **while** $B^{usado} \leq B$ **do**
5: $\quad j \leftarrow j + 1$
6: $\quad \theta^{nueva} \leftarrow$ muestra sesgada hacia θ^{elite} de X
7: $\quad \theta_j \leftarrow \theta^{nueva} \cup \theta^{elite}$
8: $\quad \theta^{elite} \leftarrow$ resultado de una **carrera** usando θ_j como población y con presupuesto B_j
9: **return** θ^{elite}

Irace empieza estimando cuántas iteraciones N^{iter} va a ejecutar; este valor está en función al número de parámetros que se piensa entonar, por lo tanto $N^{iter} = \lfloor 2 + log_2 N^{parametros} \rfloor$, donde $N^{parametros}$ representa el número de parámetros, con la idea de que mientras más parámetros se necesite entonar, mayor será la cantidad de iteraciones que una carrera en específico necesita para conseguir las configuraciones élites. El cálculo de cuántas carreras se van a realizar en total, calculado N^{iter}, va en función de cuanto presupuesto B se le asigne a todo el proceso; se calcula la cantidad de carreras como $\lfloor B/N^{iter} \rfloor$. Si la cantidad de carreras es 0 entonces *irace* pide que se introduzca un B mayor. Para este estudio, se asignó para los conjuntos pequeños y medianos $B = 1000$ iteraciones y en cada carrera las metaheurísticas disponían de 1000 iteraciones para hallar el mejor valor posible; en cambio, para los conjuntos grandes se asignó $B = 400$ y cada metaherística en cada carrera disponía de 300 iteraciones.

Cabe acotar que *irace* asume que el problema sobre el cual se va a entonar es un problema de minimización. Por lo tanto, siempre se busca menores valores de la

función de utilidad U para cada configuración; además, luego de cada carrera asigna un rango r_z dependiendo de cómo se compare la configuración z con respecto al resto, a menor rango, mejor es la configuración, por lo tanto, el conjunto de élites θ^{elite} está conformado por los k elementos con menor rango. El número k es calculado al principio de la corrida de *irace* y en este trabajo se deja su cálculo automático por defecto.

Por otra parte,en la primera línea del algoritmo 2.2, se hace un muestreo uniforme del espacio de configuraciones X con una distribución normal truncada. En las siguientes iteraciones se va sesgando el muestreo con las configuraciones élites encontradas en la línea 6, esto se logra usando una probabilidad p_z con la cual se toma una configuración θ^z que se calcula como en la ecuación (2.5), donde N_{j-1}^{elite} es el número de élites de la iteración j-1:

$$p_z = \frac{N_{j-1}^{elite} - r_z + 1}{N_{j-1}^{elite} * \frac{N_{j-1}^{elite}+1}{2}} \tag{2.5}$$

Una vez obtenido un θ^z se genera una nueva configuración tomando una muestra del espacio de parámetros X, un parámetro d a la vez de entre un rango $[d_{inf}, d^{sup}]$ definido por el usuario, usando una distribución normal truncada $\mathcal{N}(\mu_d^z, (\sigma_d^j)^2)$ donde μ_d^z, que representa la media, es el valor θ_d^z y la variancia σ_d^j está definida como $(d^{sup} - d_{inf})/2$ en la iteración j, con una actualización progresiva en cada iteración dada por la ecuación (2.6), donde N_j^{nuevo} es el número de configuraciones en la población que se va a generar en la línea 6 del algoritmo 2.2 en la iteración j, con el fin de acercar cada vez más los nuevos elementos a la élite encontrada en las últimas iteraciones del experimento.

$$\sigma_d^j = \sigma_d^{j-1} * \left(\frac{1}{N_j^{nuevo}}\right)^{\frac{1}{N^{parametros}}} \tag{2.6}$$

El proceso de carrera (*racing* en inglés) para la entonación de metaheurísticas con el cual se realiza el paso (2) (líneas 2 y 8), fue propuesto por *Birattari, M. et al.* en [BSPV02]. La carrera empieza usando una población con configuraciones; luego, a cada iteración de la carrera, las configuraciones son evaluadas en una sola instancia I_j. Después de un número iteraciones dadas por un parámetro T, aquellas configuraciones que estadísticamente tienen un desempeño inferior al resto son descartadas.

Irace establece $T = 1$; además, puede usar una prueba Friedman no paramétrica o una prueba t-test como prueba estadística para hacer las comparaciones entre las configuraciones. Por recomendación de los autores, se usa una prueba t-test con un valor de significancia de 0.05 ya que, según sus criterios, es la más adecuada para la entonación de parámetros de valores continuos.

Irace, además, implementa un método de reinicialización cuando la población de configuraciones converge prematuramente. En este proceso se mantienen las configuraciones élites de la última carrera y empieza de nuevo segun ciertas consideraciones. Aunado a esto, *irace* también implementa un sistema de carrera elitista, el cual evita que las mejores configuraciones encontradas hasta el momento se pierdan en una carrera producto de una serie desfavorable de evaluaciones en un momento dado. En este trabajo se usa ambas funciones como configuración por defecto que tiene *irace*. Para más información de todas las funciones y utilidades que presenta esta herramemienta, se recomienda leer [LIDLC$^+$16].

Capítulo 3

Resultados

3.1. Diseño experimental

El objetivo de este estudio es determinar si existe una mejoría en el desempeño de los algoritmos evolutivos utilizados en este trabajo (GGA, SSGA, MA y CHC), al usar como conjunto inicial los resultados obtenidos por CNN, ENN y RSS, en vez de usar una inicialización aleatoria.

Para lograr este objetivo, lo primero que se hace es entonar los algoritmos evolutivos para que devuelvan el mejor valor posible al probarse con los conjuntos de datos expuestos en la sección 2.4. Para lograr la entonación, se usa *irace*, expuesto en la sección 2.6, con los parámetros de la tabla 3.1. Se utilizan todos las instancias de la sección 2.4 en la entonación, se consigue 3 configuraciones para cada algoritmo evolutivo, una para los conjuntos pequeños, otra para los medianos y la última para los grandes. Se usa 1000 iteraciones para entonar los conjuntos pequeños y medianos y 400 para los conjuntos grandes, la otra modificación es que el número de iteraciones dado a cada metaheurística en cada carrera es 1000 para los conjuntos pequeños y medianos y 300 para los grandes; además, en la entonación de los conjuntos grandes sólo se realiza cada prueba con cada estrato una vez, en vez de las 3 veces decididas en la sección 2.5; esto se debe a que sin estas simplificaciones y reducciones, el tiempo de entonación con los conjuntos grandes sería muy elevado. Por último, se usa 100 iteraciones para entonar solamente el número de iteraciones sin mejorías una vez que se fijan los otros parámetros a su configuración óptima.

Parámetros	irace
Iteraciones	1000, 400 y 100
Número decimales significativos	4
Prueba estadística	t-test
Nivel de confianza para prueba estadística	0.95
Frecuencia de la prueba estadística	1 iteración
Número de configuraciones élites	automática
Reinicialización por convergencia prematura	Sí
Modo elitista	Sí

Tabla 3.1: Parámetros usados para *irace*

Parámetros	Tipo de dato	Rangos
Iteraciones sin mejorías	entero	[10,100]
Población	entero	[10,150]
Probabilidad de cruce	real	[0,1]
Probabilidad de mutación	real	[0,0.01]
Número del torneo	entero	[1,10]

Tabla 3.2: Rangos usados para los parámetros en la entonación

Los parámetros a entonar son: iteraciones sin mejorías para detener el algoritmo, población, probabilidad de cruce, probabilidad de mutación y número de elementos ha considerar en el torneo. Los rangos válidos para cada parámetro se presentan en la tabla 3.2. Se usa una población de entre 10 y 150 porque *Cano, J. et al.* en muchos de sus trabajos como [dA04] usa poblaciones de alrededor de 50; se usa un rango de probabilidad de mutación con cota superior de 1 % haciendo referencia a los mismos trabajos de *Cano, J. et al.* [dA04, GDCH12, GCH08], donde nunca superan esta cota; se coloca un torneo de entre 1 y 10 por las recomendaciones de *Talbi, E.* en [Tal09] donde explican que mientras mayor sea el número del torneo menos oportunidades tienen los peores cromosomas de la población para reproducirse; por lo tanto se elige 10 como cota superior, considerando que si las poblaciones están alrededor de los 50 cromosomas, entonces 1/5 de la población no tiene oportunidad de reproducirse.

Una vez obtenido las distintas configuraciones para cada algoritmo evolutivo. Se procede con el experimento principal, el cual consta de obtener los resultados de reducción, precisión y kappa evaluados en el conjunto de entrenamiento y en el conjunto de prueba y tiempo en segundos de cada heurística y metaheurística. Primero se obtienen los resultados de CNN, RSS y ENN por sí solas. Luego el experimento se divide en dos modalidades, la primera consta de probar las metaheurísticas utilizando como

criterio de parada el paso de un número de iteraciones sin mejorías, de tal manera que cada metaheurística tenga la posibilidad de converger a una solución sin restricciones de tiempo, a esta fase se le nombra como modalidad A; por otra parte la modalidad B prueba las metaheurísticas fijando el número de iteraciones en el cual corren, ésto con el sentido de poder comparar el desempeño de las metaheurísticas dado los mismos recursos de número de iteraciones, en este caso se fijan las iteraciones a 1000 para todas las metaheurísticas.

Entrando en la modalidad A, primero se prueban las metaheurísticas utilizando una población inicial generada aleatoriamente donde cada gen tiene una probabilidad de tener valor 1 de 50 %. Esto se decide en base a los resultados de *Flores, A.* en [Ale14] donde muestran que tienen las menores tasas de error tanto en el entrenamiento como en la validación. Luego se hacen las combinaciones entre heurísticas y metaheurísticas, se usa el conjunto reducido S obtenido de la heurística como cromosoma base para la creación de la población inicial de la metaheurística; esta población se forma heredando aleatoriamente 50 % de los genes con valor 1 del cromosoma base S, de tal manera de que se varíen las soluciones, pero manteniendo por fuera las instancias del conjunto original que fueron eliminadas por la heurística, la idea es usar una población inicial con las características particulares proporcionadas por RSS, CNN y ENN. Acto seguido, se prueban las metaheurísticas usando como cromosoma base S el conjunto combinado por los resultados de CNN con RSS y ENN con RSS; esto con el fin de explorar la posibilidad de que el conjunto hallado por RSS pueda complementar al conjunto hallado por CNN y ENN y mejore el desempeño de las metaheurísticas. Por otra parte la modalidad B es idéntica a la A con el detalle que las metaheurísticas se detienen exactamente a 1000 iteraciones.

Los resultados para cada conjunto de datos individualmente evaluadas se presentan en el anexo A, en la sección 3.3 se presentan tablas con los promedios de reducción, precisión, kappa y tiempo en segundos de las distintas heurísticas, metaheurísticas y sus combinaciones. Además, se presentan tablas en las cuales se le asignan grados a los distintos algoritmos de acuerdo a cuántas veces tuvieron los mejores resultados para cada instancia, estos grados se determinan para cada uno de los criterios mencionados anteriormente y para las combinaciones reducción + precisión y reducción + kappa expresados como combinación lineal con vectores de pesos de 0.5. Por último, para

Parámetros	Algoritmos			
	GGA	SGA	ME	CHC
Iteraciones fijas	1000	1000	1000	1000
Iteraciones sin mejoría	-	-	-	-
Población	70	90	21	33
Prob. de Cruce	0.4837	0.9848	0.9496	-
Prob. de Mutación	0.0001	0.0057	0.0071	-
Número del torneo	-	3	1	-

Tabla 3.3: Parámetros usados para los conjuntos pequeños

comparar si realmente existen diferencias entre los resultados se utiliza una prueba no paramétrica de grado con signo de *Wilcoxon* con un nivel significativo de 1 %; se hacen pruebas para cada criterio y sus combinaciones como las realizadas en las tablas para grados. Luego, se presentan los resultados en unas tablas en las cuales se compara cada par de algoritmo y combinación antes expuesta en cada columna y cada fila, de tal manera que en una casilla hay un (+) si el algoritmo presente en la fila es mejor que el algoritmo presente en la columna, un (-) si es peor y (=) si son iguales y se agregan dos columna al final que cuentan cuántos algoritmos de las columnas son iguales o peores que el algoritmo de la fila y cuántos algoritmos de las columnas son peores que el algoritmo de la fila.

Los experimentos fueron hechos con un procesador Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz, 4 procesadores y 4GB de memoria RAM. El preprocesamiento de los datos fue hecho con Python v.3.6.5, la entonación fue hecha con *Irace* v.2.4 y R v.3.4.4, los experimentos principales fueron realizados con GCC v.7.3.0.

3.2. Resultados de la entonación

Parámetros	Algoritmos			
	GGA	SGA	ME	CHC
Iteraciones fijas	1000	1000	1000	1000
Iteraciones sin mejoría	-	-	-	-
Población	88	132	32	33
Prob. de Cruce	0.5779	0.9859	0.9549	-
Prob. de Mutación	0.0001	0.0001	0.0004	-
Número del torneo	-	1	3	-

Tabla 3.4: Parámetros usados para los conjuntos medianos

Conclusiones y Recomendaciones

Conclusiones

Bibliografía

[AD91] Hussein Almuallim and Thomas G Dietterich. Learning with many irrelevant features. In *AAAI*, volume 91, pages 547–552, 1991.

[AFFL+11] Jesús Alcalá-Fdez, Alberto Fernández, Julián Luengo, Joaquín Derrac, Salvador García, Luciano Sánchez, and Francisco Herrera. Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic & Soft Computing*, 17, 2011.

[AL03] Emile HL Aarts and Jan Karel Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

[Ale14] Flores Alejandro. *Metaheurísticas Bio-Inspiradas para Selección de Instancias*. PhD thesis, Undergraduate thesis,Departamento de Ciencias de la Computación, Universidad Simón Bolívar, Venezuela, 2014.

[Bat94] Roberto Battiti. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on neural networks*, 5(4):537–550, 1994.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[BF99] Carla E Brodley and Mark A Friedl. Identifying mislabeled training data. *Journal of artificial intelligence research*, 11:131–167, 1999.

[BLN86] Carlo Batini, Maurizio Lenzerini, and Shamkant B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM computing surveys (CSUR)*, 18(4):323–364, 1986.

[BSPV02] Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 11–18. Morgan Kaufmann Publishers Inc., 2002.

[CAB11] Cagatay Catal, Oral Alan, and Kerime Balkan. Class noise detection based on software metrics and roc curves. *Information Sciences*, 181(21):4867–4877, 2011.

[CH67] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[CHL05] José Ramón Cano, Francisco Herrera, and Manuel Lozano. Stratification for scaling up evolutionary prototype selection. *Pattern Recognition Letters*, 26(7):953–963, 2005.

[CKLS01] Munir Cochinwala, Verghese Kurien, Gail Lalk, and Dennis Shasha. Efficient data reconciliation. *Information Sciences*, 137(1-4):1–15, 2001.

[CPSK07] Krzysztof J Cios, Witold Pedrycz, Roman W Swiniarski, and Lukasz Andrzej Kurgan. *Data mining: a knowledge discovery approach*. Springer Science & Business Media, 2007.

[dA04] José Ramón Cano de Amo. *Reducción de datos basada en Selección Evolutiva de Instancias para Mineria de Datos*. PhD thesis, PhD thesis, Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada, Spain, 2004.

[DBS17] Marco Dorigo, Mauro Birattari, and Thomas Stützle. *Metaheuristic*, pages 817–818. Springer US, Boston, MA, 2017.

[DGL13] Luc Devroye, László Györfi, and Gábor Lugosi. *A probabilistic theory of pattern recognition*, volume 31. Springer Science & Business Media, 2013.

[DKT17] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[DL97] Manoranjan Dash and Huan Liu. Feature selection for classification. *Intelligent data analysis*, 1(3):131–156, 1997.

[Dor92] Marco Dorigo. Optimization, learning and natural algorithms. *PhD Thesis, Politecnico di Milano*, 1992.

[ES11] Agoston E Eiben and Selmar K Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[Esh91] Larry J Eshelman. The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In *Foundations of genetic algorithms*, volume 1, pages 265–283. Elsevier, 1991.

[ESK01] Russell C Eberhart, Yuhui Shi, and James Kennedy. *Swarm intelligence*. Elsevier, 2001.

[FHJ51] Evelyn Fix and Joseph L Hodges Jr. Discriminatory analysis-nonparametric discrimination: consistency properties. Technical report, California Univ Berkeley, 1951.

[FKP07] Alireza Farhangfar, Lukasz A Kurgan, and Witold Pedrycz. A novel framework for imputation of missing values in databases. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 37(5):692–709, 2007.

[FM17] Alejandro Flores and David M Mount. Nearest neighbor condensation with guarantees. 2017.

[FR95] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

[FS69] Ivan P Fellegi and Alan B Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[FSS96] Usama M Fayyd, Gregory P Shapiro, and Padhraic Smyth. From data mining to knowledge discovery: An overview. 1996.

[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.

[GDCH12] Salvador Garcia, Joaquin Derrac, Jose Cano, and Francisco Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE transactions on pattern analysis and machine intelligence*, 34(3):417–435, 2012.

[GLH16] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Springer, 2016.

[Glo77] Fred Glover. Heuristics for integer programming using surrogate constraints. *Decision sciences*, 8(1):156–166, 1977.

[Glo89] Fred Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[GLS+13] Salvador Garcia, Julian Luengo, José Antonio Sáez, Victoria Lopez, and Francisco Herrera. A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):734–750, 2013.

[Har68] Peter Hart. The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 14(3):515–516, 1968.

[Hol75] John H Holland. Adaptation in natural and artificial systems. an introductory analysis with application to biology, control, and artificial intelligence. *Ann Arbor, MI: University of Michigan Press*, pages 439–444, 1975.

[Ind04] Piotr Indyk. Nearest neighbors in high-dimensional spaces. 2004.

[K+95] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

[KCH+03] Won Kim, Byoung-Ju Choi, Eui-Kyeong Hong, Soo-Kyung Kim, and Doheon Lee. A taxonomy of dirty data. *Data mining and knowledge discovery*, 7(1):81–99, 2003.

[KGV83] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[KM17] Eamonn Keogh and Abdullah Mueen. Curse of dimensionality. In *Encyclopedia of Machine Learning and Data Mining*, pages 314–315. Springer, 2017.

[KS96] Daphne Koller and Mehran Sahami. Toward optimal feature selection. Technical report, Stanford InfoLab, 1996.

[LIDLC+16] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[LLIB06] Jose A Lozano, Pedro Larrañaga, Iñaki Inza, and Endika Bengoetxea. *Towards a new evolutionary computation: advances on estimation of distribution algorithms*, volume 192. Springer, 2006.

[LM12] Huan Liu and Hiroshi Motoda. *Feature selection for knowledge discovery and data mining*, volume 454. Springer Science & Business Media, 2012.

[LMD98] Huan Liul, Hiroshi Motoda, and Manoranjan Dash. A monotonic measure for optimal feature selection. In *European conference on machine learning*, pages 101–106. Springer, 1998.

[LMS03] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

[LS+96] Huan Liu, Rudy Setiono, et al. A probabilistic approach to feature selection-a filter solution. In *ICML*, volume 96, pages 319–327. Citeseer, 1996.

[MH97] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[NC12] Ferrante Neri and Carlos Cotta. Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, 2:1–14, 2012.

[Omo89] Stephen M Omohundro. *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[Pea84] Judea Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.

[PSL06] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[SDI06] Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. *Nearest-neighbor methods in learning and vision: theory and practice (neural information processing)*. The MIT press, 2006.

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[Tal09] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[Ten99] Choh-Man Teng. Correcting noisy data. In *ICML*, pages 239–248. Citeseer, 1999.

[Uhl91] Jeffrey K Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information processing letters*, 40(4):175–179, 1991.

[Vou98] Christos Voudouris. Guided local search—an illustrative example in function optimisation. *BT Technology Journal*, 16(3):46–50, 1998.

[Wil72] Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, (3):408–421, 1972.

[XYC88] Lei Xu, Pingfan Yan, and Tong Chang. Best first strategy for feature selection. In *Pattern Recognition, 1988., 9th International Conference on*, pages 706–708. IEEE, 1988.

[YW09] Ying Yang and Geoffrey I Webb. Discretization for naive-bayes learning: managing discretization bias and variance. *Machine learning*, 74(1):39–74, 2009.

[Zuk10] AV Zukhba. Np-completeness of the problem of prototype selection in the nearest neighbor method. *Pattern Recognition and Image Analysis*, 20(4):484–494, 2010.

Apéndice A

Apéndice A

Apéndice A