UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

División de Estudios de Posgrado de la
Facultad de Ingeniería Eléctrica

GENERACIÓN DE PROTOTIPOS PARA LA CLASIFICACIÓN DE PATRONES USANDO PROGRAMACIÓN GENÉTICA

TESIS

Que para obtener el grado de

MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

Opción Sistemas Computacionales

Presenta

José María Valencia Ramírez

Ingeniero en Computación

Jaime Cerda Jacobo

Doctor en Ingeniería Eléctrica y Electrónica

Director de Tesis

Mario Graff Guerrero

Doctor en Ciencias de la Computación

Co-Director de Tesis

Febrero 2015

Lista de Publicaciones

- Valencia-Ramirez, J. M., Raya, J. A., Cedeno, J. R., Suarez, R. R., Escalante, H. J., and Graff, M. (2014). Comparison between genetic programming and full model selection on classification problems. In *Power, Electronics and Computing (ROPEC), 2014 IEEE International Autumn Meeting on*, pages 1−6.

- Suarez, R. R., Valencia-Ramirez, J. M., and Graff, M. (2014). Genetic programming as a feature selection algorithm. In *Power, Electronics and Computing (ROPEC), 2014 IEEE International Autumn Meeting on*, pages 1−5.

- Valencia-Ramirez, J. M., Graff, M., Escalante, H. J., y Cerda-Jacobo, J. An Iterative Genetic Programming Approach to Prototype Generation. *Genetic Programming and Evolvable Machines*. Enviado para revisión el 28 de Enero de 2015.

Resumen

La regla de k-vecinos más cercanos (kNN, k-Nearest Neighbors) es una de las técnicas de clasificación de patrones más populares y exitosas. A pesar de su alta precisión de clasificación, se sabe que kNN tiene algunas desventajas; principalmente, los requerimientos para almacenar todas las instancias en el conjunto de entrenamiento, $\mathcal{T}$. Por otro lado, cuando un nuevo patrón debe ser clasificado por kNN, se compara con todas las instancias en $\mathcal{T}$. Evidentemente, este es un aspecto crucial cuando se tiene un conjunto $\mathcal{T}$ grande y se quiere clasificar a muchos objetos.

En este trabajo, se propone un enfoque de Programación Genética (GP, Genetic Programming) al problema de la Generación de Prototipos (PG, Prototype Generation) para una clasificación basada en la regla NN. La idea de PG es representar a todos los elementos en $\mathcal{T}$ con un número de instancias generadas, con el objetivo de reducir los requerimientos de almacenamiento y el coste computacional inherente en clasificadores NN. Para generar las instancias artificiales, se propone una técnica de GP iterativo, la cual usa una combinación no lineal de las muestras disponibles en $\mathcal{T}$.

Los experimentos son presentados en un conjunto de bases de datos de referencia para la evaluación de métodos PG. Los resultados experimentales muestran que el enfoque propuesto es muy competitivo con técnicas del estado del arte, obteniendo la mejor precisión reportada hasta el momento para este conjunto de bases. También el método propuesto es muy competitivo en términos de rendimiento de reducción.

Palabras clave: Programación Genética, k-Vecinos más Cercanos, Generación de Prototipos, Clasificación de Patrones.

Abstract

k-Nearest Neighbor (kNN) is one of the most popular and successful pattern classification techniques. Despite its high classification accuracy, it is known that kNN has some disadvantages; most notably, the requirement to store all the instances in the training set, $\mathcal{T}$. On the other hand, when a new pattern must be classified by kNN, it is compared to all of the instances in $\mathcal{T}$. Obviously, this is a crucial aspect when one has a large set $\mathcal{T}$ and wants to classify many objects.

In this work, it proposes a Genetic Programming (GP) approach to the problem of Prototype Generation (PG) for NN based classification. The idea of PG is to represent all the elements in $\mathcal{T}$ with a number of generated instances, with the goal of reducing the storage requirements and the computational cost inherent in NN classifiers. To generate artificial instances, an iterative GP technique is proposed, which uses a non-linear combination of instances available in $\mathcal{T}$.

Experiments are reported in a suite of benchmark for evaluation of PG methods. Experimental results show that the proposed approach is very competitive with state of the art techniques, obtaining the best accuracy reported so far for this benchmark. The proposed method is very competitive in terms of reduction performance as well.

Keywords: Genetic Programming, k-Nearest Neighbors, Prototype Generation, Pattern Classification.

Contenido

Resumen v
Abstract vii
Contenido viii
Lista de Figuras xi
Lista de Tablas xiii
Lista de Algoritmos xv
Lista de Símbolos xvii

1. Introducción 1
1.1. Planteamiento del Problema 1
1.1.1. Definición formal de generación de prototipos 2
1.2. Reconocimiento de Patrones 3
1.3. Aprendizaje y Adaptación 3
1.3.1. Aprendizaje supervisado 4
1.3.2. Aprendizaje no supervisado 4
1.4. Regla de k-vecinos más cercanos 6
1.5. Esquemas de reducción mediante prototipos 7
1.5.1. Dirección de búsqueda 8
1.5.2. Mecanismos de generación 9
1.5.3. Criterios para comparar métodos de generación de prototipos 10
1.6. Trabajos relacionados 12
1.6.1. Edición Generalizada utilizando Vecinos más Cercanos 12
1.6.2. Algoritmo de Selección Clonal para la Selección de Prototipos 13
1.6.3. Algoritmos evolutivos para generar prototipos 14
1.7. Objetivos de la Tesis 16
1.7.1. Objetivo general 16
1.7.2. Objetivos particulares 16
1.8. Descripción de Capítulos 17

2. Generación de Prototipos mediante Programación Genética 19
2.1. Función de distancia 19
2.2. Programación Genética 20
2.3. Método propuesto 21
2.3.1. Operadores Genéticos 23

2.3.2. Función de aptitud . . . 25
2.3.3. Resumen de parámetros utilizados . . . 26
2.4. Algoritmo GP propuesto para generar prototipos . . . 26
2.5. Generando prototipos usando Programación Genética . . . 29
2.5.1. Ejemplo 1 . . . 29
2.5.2. Ejemplo 2 . . . 32
2.6. Conclusiones . . . 35

3. Algoritmos Implementados 39
3.1. GP^{BAL} . . . 39
3.2. GP^{UNB} . . . 40
3.3. GP^{UNB2} . . . 43
3.4. Conclusiones . . . 46

4. Resultados 47
4.1. Conjunto de bases de datos de referencia utilizado . . . 47
4.2. Experimentos realizados . . . 49
4.3. Visualización de prototipos . . . 49
4.4. Resultados obtenidos en bases de datos pequeñas . . . 50
4.5. Resultados obtenidos en bases de datos grandes . . . 53
4.6. Comparación de los resultados obtenidos con técnicas del estado del arte . . . 58
4.7. Conclusiones . . . 60

5. Conclusiones 65
5.1. Trabajos Futuros . . . 66

Referencias 67

Glosario 71

Lista de Figuras

1.1. Procedimiento utilizado para realizar reconocimiento de patrones. 5
1.2. Ejemplo de generación de prototipos 8

2.1. Representación de los programas usando un bosque, donde se utiliza el nodo vacío salida para unir los árboles en el bosque. 22
2.2. Cruza de sub-árboles. La línea punteada indica el punto de cruza. 24
2.3. a) Conjunto de entrenamiento; donde las instancias de la clase 0 se muestran con cuadros negros, instancias de la clase 1 con círculos verdes e instancias de la clase 2 con triángulos azules, b) Segmentación del espacio de búsqueda utilizando el conjunto de prototipos generados y la distancia euclidiana. . . 31
2.4. Genotipo del mejor individuo encontrado al utilizar el Algoritmo 3. 32
2.5. a) Conjunto de entrenamiento; donde las instancias de la clase 0 se muestran con cuadros negros e instancias de la clase 1 con círculos verdes, b) - f) Segmentación del espacio de búsqueda utilizando $1, 2, 3, 4, 5$ prototipo(s) por clase y distancia euclidiana respectivamente. 36
2.6. 5 Genotipos de los mejores individuos encontrados al ejecutar el Algoritmo 3, 5 veces iterativamente. 37

4.3. a) Precisión en $\mathcal{V}$, b) reducción de $\mathcal{T}$ y c) precisión en $\mathcal{V}$ (ρ) por el porcentaje de reduccción (γ); usando los métodos propuestos con diferentes umbrales τ en bases de datos pequeñas. La línea gris en 73.48 en a) indica la precisión obtenida por el clasificador NN. 51
4.4. Rendimiento obtenido en términos de precisión por GP^{BAL}, GP^{UNB} y GP^{UNB2} en el conjunto de entrenamiento en bases de datos pequeñas 52
4.5. Rendimiento obtenido en términos de precisión por obtenido por GP^{BAL}, GP^{UNB} y GP^{UNB2} en el conjunto de validación en bases de datos pequeñas 52
4.6. a) Precisión en $\mathcal{V}$, b) reducción de $\mathcal{T}$ y c) precisión en $\mathcal{V}$ (ρ) por el porcentaje de reduccción (γ); usando los métodos propuestos con diferentes umbrales τ en bases de datos grandes. La línea gris en 80.60 en a) indica la precisión obtenida por el clasificador NN. 55
4.7. Rendimiento obtenido en términos de precisión por GP^{BAL}, GP^{UNB} y GP^{UNB2} en el conjunto de entrenamiento en bases de datos grandes 55

4.8. Rendimiento obtenido en términos de precisión por GP^{BAL}, GP^{UNB} y GP^{UNB2} en el conjunto de validación en bases de datos grandes 56
4.9. a) Precisión en $\mathcal{V}$, b) reducción de $\mathcal{T}$ y c) precisión en $\mathcal{V}$ (ρ) por el porcentaje de reduccción (γ); usando los métodos propuestos con diferentes umbrales τ en todas las bases de datos utilizadas. La línea gris en 77.04 en a) indica la precisión obtenida por el clasificador NN. 57
4.1. Diagrama de bloques del procedimiento utilizado para evaluar la precisión y la reducción de los PG. 61
4.2. Prototipos generados para el conjunto de datos Banana. a) Prototipos generados con $\tau = 1$, notar que los tres métodos propuestos generan los mismos prototipos en este caso. b) Prototipos generados usando GP^{BAL} con $\tau = 3$. c) Prototipos generados usando GP^{BAL} con $\tau = 5$. d) Prototipos generados usando GP^{BAL} con $\tau = 10$. e) Prototipos generados usando GP^{BAL} con $\tau = 50$. f) Prototipos generados usando GP^{UNB} con $\tau = 50$. 62
4.10. Precisión en $\mathcal{V}$ contra Reducción de $\mathcal{T}$. Se consideran 25 métodos PG reportados en Triguero et al., GPPG y EMOPG en bases de datos pequeñas. . . 63
4.11. Precisión en $\mathcal{V}$ contra Reducción de $\mathcal{T}$. Se consideran 20 métodos PG reportados en Triguero et al., GPPG y EMOPG en bases de datos grandes. . . . 63

Lista de Tablas

2.1. Parámetros de GP 26

4.1. Descripción del conjunto de las bases de datos utilizadas para el estudio experimental. Para cada base de datos se muestra: el número de instancias, el número de atributos y el número de clases. 48
4.2. Rendimiento promedio y la desviación estándar, así como el porcentaje de reducción promedio obtenida por los métodos propuestos para pequeñas bases de datos utilizando diferentes umbrales τ. 51
4.3. Rendimiento promedio y desviación estándar, así como el porcentaje de reducción promedio obtenida por los métodos propuestos para bases de datos grandes utilizando diferentes umbrales τ 54
4.4. Rendimiento promedio y desviación estándar, así como el porcentaje de reducción promedio obtenida por los métodos propuestos para todas las bases de datos de referencia usando diferentes umbrales τ 57
4.5. Precisión media y desviación estándar en los conjuntos de datos con similar tipo de características. 58
4.6. Media y desviación estandar del rendimiento y porcentaje de reducción obtenida por GP^{UNB2} para bases de datos pequeñas, grandes, y todas. Para la comparación se muestra el rendimiento obtenido por EMOPG, GPPG, GENN, PSCSA y 1NN. Los mejores resultados se muestran en negrita. . . . 59

Lista de Algoritmos

1. Pseudocódigo utilizado por PSCSA 14
2. Programación Genética 21
3. Algoritmo propuesto de GP para generar prototipos 27
4. Algoritmo utilizado por GP^{BAL} 41
5. Algoritmo utilizado por GP^{UNB} 44
6. Algoritmo utilizado por GP^{UNB2} 45

Lista de Símbolos

$\mathcal{T}$	Conjunto de entrenamiento.
$\mathcal{P}$	Conjunto de prototipos.
$O(N)$	Notación O, para representar un algoritmo de orden lineal.
N	Cardinalidad del conjunto de entrenamiento.
x_i	Instancia del conjunto de entrenamiento.
y_i	Etiqueta de una instancia.
C	Conjunto de clases contenidas en un conjunto de entrenamiento.
K	Número de clases contenidas en un conjunto de entrenamiento.
w_i	Prototipo generado.
M	Cardinalidad del conjunto de prototipos.
d	Número de características de las instancias.
$\mathbb{R}^d$	Espacio real de dimensión d.
$E(\vec{x_1}, \vec{x_2})$	Distancia euclidiana de dos vectores.
$\mathbb{T}$	Conjunto de terminales.
$\mathbb{F}$	Conjunto de funciones.
Φ	Contienen las características del conjunto de entrenamiento.
δ	Parámetros de GP.
g	Número de generaciones.
s	Tamaño de la población.
t	Objetivo de GP.
$best\_ind$	Variable donde se almacena el mejor programa de la población.
$vector\_c^*$	Arreglo donde se almacenan las etiquetas de los prototipos generados por GP.
$vector\_c$	Arreglo temporal donde se almacenan las etiquetas de los prototipos que se estan generando mediante GP.
f^*	Aptitud del mejor programa de la población.
f	Aptitud de la población.
bal	Variable utilizada para saber si se estan generando prototipos de manera balanceada.
$\mathbb{P}$	Población de programas.
$\mathcal{P}_j$	Conjunto de prototipos generados por el j-ésimo programa de la población.
$\mathcal{D}$	Matriz de distancias euclidianas.
mod	Resto de una división entre enteros.
$\mathcal{P}^*$	Conjunto de prototipos óptimo.

τ	Máximo número de prototipos por clase.
N_s	Número de semillas.
enc	Variable que define si se han encontrado prototipos que ayuden a la clasificación.
p_f	Umbral de balanceo de clasificación.
p_g	Umbral de discriminación.
fit_class	Aptitud de clasificación por clase.
$\mathcal{V}$	Conjunto de validación.
ρ	Precisión en $\mathcal{V}$.
γ	Reducción de $\mathcal{T}$.

Capítulo 1

Introducción

Este capítulo describe los principales problemas que se tienen al utilizar clasificadores basados en similitud y se define formalmente como se pueden resolver utilizando generación de prototipos. Posteriormente se definen algunos conceptos básicos utilizados en el reconocimiento de patrones, así como el clasificador popular k-vecinos más cercanos (kNN, por sus siglas en inglés) y los principales mecanismos y direcciones de búsquedas que se utilizan en métodos de generación de prototipos. Por último, presenta algunos enfoques que se han utilizado en el estado del arte para generar prototipos y se plantean los objetivos de la tesis.

1.1. Planteamiento del Problema

Entre los métodos de clasificación de patrones más populares se encuentran los basados en similitud [Fayed et al., 2007, Escalante et al., 2013]. Este tipo de métodos se basan en medidas de similitud para asignar etiquetas a los nuevos objetos, un clasificador representante de esta metodología es kNN. Los métodos basados en la similitud han reportado excelentes resultados en tareas de clasificación de patrones. Sin embargo, a pesar de su rendimiento aceptable, éstos requieren computar las estimaciones de similitud con el conjunto de entrenamiento $\mathcal{T}$ cuando una nueva instancia tiene que ser clasificada, que puede ser costoso computacionalmente. Además, este tipo de métodos requie-

ren grandes recursos para el almacenamiento y pueden ser sensibles a instancias ruidosas [Wilson and Martinez, 2000, Garain, 2008, Triguero et al., 2012].

Los clasificadores basados en prototipos[1] tienen como objetivo reducir el costo computacional mediante el uso de un conjunto de casos representativos $\mathcal{P}$ para la clasificación en lugar de todo el conjunto de entrenamiento $\mathcal{T}$. El problema clave en la clasificación basada en prototipos es determinar cuáles son los prototipos que se utilizarán para la clasificación. Hay dos alternativas principales para resolver este problema: selección y generación de prototipos. La selección de prototipos (PS, por sus siglas en inglés) consiste en seleccionar un subconjunto de $\mathcal{T}$ para formar $\mathcal{P}$ [Garcia et al., 2012]. La generación de prototipos (PG, por sus siglas en inglés) consiste en generar $\mathcal{P}$ mediante el uso de la información en $\mathcal{T}$ [Triguero et al., 2012].

PS y PG han demostrado ser eficaces, sin embargo, PS se puede ver como un caso especial de PG. Razón por la cual este documento se centra en PG. A continuación se define formalmente el problema de generación de prototipos.

1.1.1. Definición formal de generación de prototipos

Una definición formal al problema de PG es la siguiente: Sea $\mathcal{T} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ un conjunto de entrenamiento de instancias etiquetadas, con $x_i \in \mathbb{R}^d$ y $y_i \in C = \{1, 2, \ldots, K\}$, donde $\mathbb{R}^d$ es el espacio real de dimensión d, C es el conjunto de clases que contiene el problema y K es el número de clases. En este trabajo, el objetivo es obtener un conjunto de prototipos generados $\mathcal{P} = \{(w_1, y_1), (w_2, y_2), \ldots, (w_M, y_M)\}$; tal que $M \ll N$, donde $w_i \in \mathbb{R}^d$ y $\forall_c \in C\ \exists_i$ tal que $y_i = c$ para $1 \leq i \leq M$. Los prototipos en $\mathcal{P}$ son determinados para representar eficientemente las distribuciones de las clases, y discriminar bien cuando se utilizan para clasificar los elementos de $\mathcal{T}$ por medio de un clasificador kNN. Por lo tanto, M debe ser lo suficientemente pequeña para reducir tanto el almacenamiento como el tiempo de evaluación utilizado por un clasificador kNN.

[1] Un prototipo es un patrón típico de una clase específica [Fayed et al., 2007].

Este trabajo se enfoca en el uso de un clasificador kNN, con $k = 1$, para clasificar las instancias de los conjuntos de entrenamiento y validación usando $\mathcal{P}$ como referencia. Se seleccionó $k = 1$ por que es muy común en la comunidad de clasificación de patrones [Triguero et al., 2012].

1.2. Reconocimiento de Patrones

El término reconocimiento de patrones abarca una amplia gama de problemas de procesamiento de información de gran importancia práctica. Por ejemplo: reconocimiento de voz, reconocimiento de imágenes, categorización de texto, detección de fallas en maquinaria, diagnóstico médico, identificación de huellas dactilares y muchas más [Bishop, 1995, Espejo et al., 2010]. A menudo se trata de problemas que muchos humanos resuelven de una manera aparentemente sin esfuerzo. Sin embargo, sus soluciones utilizando computadoras, en muchos casos, han demostrado ser inmensamente difíciles. Para tener una mejor oportunidad de desarrollar soluciones efectivas, es importante adoptar un enfoque basado en principios basados en conceptos teóricos.

1.3. Aprendizaje y Adaptación

En general, cualquier método que incorpora información de muestras de entrenamiento en el diseño de un clasificador emplea la etapa de aprendizaje. Debido a que casi todos los problemas prácticos o interesantes de reconocimiento de patrones son tan difíciles que no podemos adivinar la decisión de clasificación con anticipación, se pasa la gran mayoría del tiempo aquí tratando de "aprender". Por lo tanto, la creación de clasificadores implica asumir alguna forma general del modelo o la forma del clasificador, y el uso de patrones de entrenamiento para aprender o estimar los parámetros desconocidos del modelo [Duda et al., 2000]. Se distinguen principalmente dos enfoques para el aprendizaje

automático: supervisado y no supervisado.

1.3.1. Aprendizaje supervisado

En el aprendizaje supervisado [Espejo et al., 2010], atributos de instancias de datos se dividen en dos tipos: las entradas o variables independientes y salidas o variables dependientes. El objetivo del proceso de aprendizaje consiste en predecir el valor de las salidas a partir del valor de las entradas. Con el fin de lograr este objetivo, un conjunto de datos de entrenamiento (instancias de datos que incluyen los valores de ambas variables de entrada y de salida con valores conocidos) se emplea para guiar el proceso de aprendizaje. Clasificación y regresión son dos tipos de tareas de aprendizaje supervisado.

1.3.2. Aprendizaje no supervisado

En el aprendizaje no supervisado [Espejo et al., 2010], no hay distinción de tipo entre las variables de las instancias de datos. Como consecuencia, no se puede hablar de datos de entrenamiento ya que no se tiene un conjunto de datos con una salida conocida. El objetivo de aprendizaje no supervisado es encontrar la estructura intrínseca, relaciones o afinidades presentes en los datos. Ejemplos de tareas de aprendizaje no supervisado son agrupamiento (clustering) y el descubrimiento de asociaciones.

La Figura 1.1 muestra el procedimiento general que se utiliza para realizar reconocimiento de patrones, desde la captura de los datos ya sea por sensores, cámaras o simples mediciones, cuyos datos se pre-procesan para posteriormente hacer la extracción de características que se utilizarán en la etapa de clasificación. Una vez que se realiza la etapa de clasificación alguna decisión es tomada [Duda et al., 2000]. En cada etapa hay varios problemas a los que se les tiene que hacer frente, sin embargo este trabajo se centra únicamente en la etapa de clasificación.

Clasificación es uno de los problemas más estudiados en el aprendizaje automático

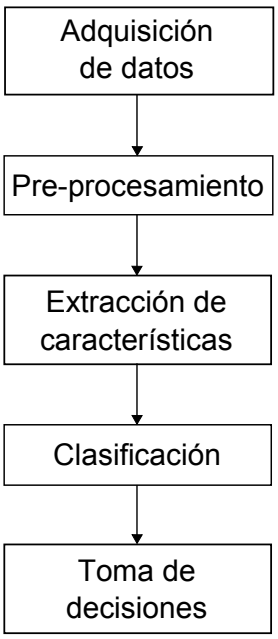

Figura 1.1: Procedimiento utilizado para realizar reconocimiento de patrones.

y minería de datos [Espejo et al., 2010]. Consiste en predecir el valor de un atributo categórico (la clase) en base a los valores de otros atributos (predicción de atributos). Un algoritmo de búsqueda se utiliza para inducir a un clasificador de un conjunto de instancias de datos clasificados correctamente llamado conjunto de entrenamiento $\mathcal{T}$. Otro conjunto de instancias de datos clasificados correctamente, conocido como conjunto de validación $\mathcal{V}$, se utiliza para medir la calidad del clasificador obtenido.

Muchos métodos de clasificación y de predicción han sido propuestas por los investigadores en el aprendizaje automático y reconocimiento de patrones [Bishop, 1995, Duda et al., 2000, Han et al., 2005, Espejo et al., 2010]. Anteriormente los algoritmos eran diseñados residentes en memoria, suponiendo típicamente un tamaño pequeño en los datos. Recientes investigaciones de minería de datos se han basado en este tipo de trabajos, desarrollando técnicas de clasificación y predicción escalables capaces de manejar grandes cantidades de datos [Han et al., 2005].

1.4. Regla de k-vecinos más cercanos

kNN [Koplowitz and Brown, 1981], es uno de los procedimientos más populares para los problemas de reconocimiento y clasificación de patrónes mediante el examen de las muestras más cercanas del conjunto de datos. Esta técnica se incluye en un campo más específico de la minería de datos conocida como aprendizaje "lazy"[Han et al., 2005], este tipo de métodos solo almacenan los datos de entrenamiento haciendo un procesamiento de datos mínimo, sin obtener el modelo de aprendizaje. Cuando llega una instancia de prueba, realiza la generalización con el fin de clasificar la instancia en base a su similitud con las muestras de entrenamiento almacenados.

A pesar de su alta precisión de clasificación [Triguero et al., 2012], se sabe que kNN tiene algunas desventajas que afectan de manera directa el costo computacional y la predicción de nuevas instancias. Los problemas que afectan el rendimiento del método de kNN son los siguientes:

- **Alto espacio en almacenamiento.** El método de kNN necesita almacenar todas las instancias del conjunto de entrenamiento, $\mathcal{T}$. Claramente, el almacenamiento se convierte en un problema cuando $|\mathcal{T}|$ y las características que lo representan es grande [Triguero et al., 2012, Wilson and Martinez, 2000].

- **Alto costo computacional.** Cuando una nueva muestra debe ser clasificada por kNN, se compara con todas las instancias en $\mathcal{T}$; lo cual implica la computación de distancias (con una complejidad $O(N)$, con $N = |\mathcal{T}|$). Este es un aspecto crucial cuando uno tiene un gran conjunto de datos y se quiere clasificar muchos objetos [Triguero et al., 2012, Wilson and Martinez, 2000].

- **Alta sensibilidad al ruido.** Se considera toda la información contenida en $\mathcal{T}$, incluso cuando $\mathcal{T}$ puede contener datos incorrectos o mal clasificados. Lo cual reduce la precisión de clasificación considerablemente [Triguero et al., 2012, Wilson and Martinez, 2000].

Diversos enfoques se han sugerido y estudiado con el fin de superar las desventajas anteriormente mencionadas. La investigación sobre medidas de similitud para mejorar la eficacia de NN (y otras técnicas similares basándose en similitudes) [Wilson and Martinez, 1997]. Otras técnicas reducen la superposición entre las clases basado en los centros de probabilidad local, aumentando así la tolerancia al ruido. También se ha investigado sobre funciones de distancia que sean adecuadas para su uso bajo condiciones de alta dimensionalidad [Triguero et al., 2012].

1.5. Esquemas de reducción mediante prototipos

Una técnica exitosa que resuelve los problemas de un clasificador kNN se basa en la reducción de datos [Wilson and Martinez, 2000]. Estas técnicas pretenden sustituir el conjunto de entrenamiento original $\mathcal{T}$ con un conjunto de prototipos $\mathcal{P}$, donde el número de prototipos es mucho más pequeño que el número de instancias en $\mathcal{T}$, es decir, $|\mathcal{P}| \ll |\mathcal{T}|$ [Triguero et al., 2012]. El problema de este enfoque es determinar el conjunto de prototipos $\mathcal{P}$ que se utilizará para la clasificación de forma tal que la precisión no se vea afectada. Hay dos alternativas para la solución de este problema, las cuales son:

- La selección de prototipos consiste en encontrar un subconjunto óptimo de objetos representativos de $\mathcal{T}$, es decir, $\mathcal{P} \subseteq \mathcal{T}$ [Garcia et al., 2012].
- La generación de prototipos también conocidos como métodos de extracción de prototipos, consiste en generar instancias artificiales y sustituir los datos originales $\mathcal{T}$ con estos [Triguero et al., 2012].

La Figura 1.2 muestra 2000 datos sintéticos de entrenamiento que están divididos en dos clases (azul y rojo), para los cuales se generan dos prototipos por cada clase (círculos negros para la clase roja y triángulos negros para la clase azul), los cuales se utilizarán para realizar la clasificación en lugar de todos los datos de entrenamiento, lo cual elimina posibles muestras con ruido y claramente aumenta el tiempo de cómputo para clasificar

nuevas instancias, y reduce el espacio de almacenamiento considerablemente.

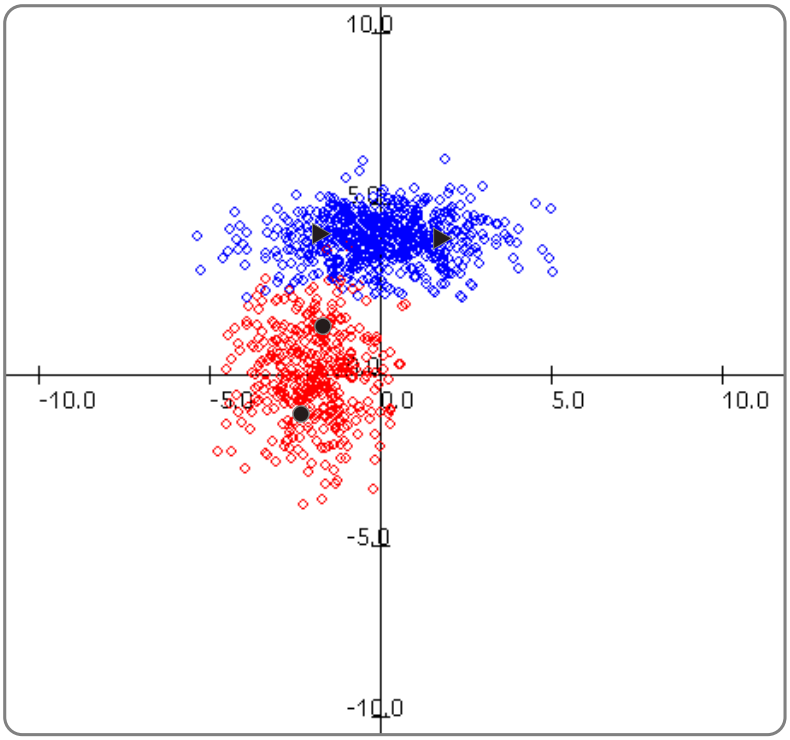

Figura 1.2: Ejemplo de generación de prototipos

1.5.1. Dirección de búsqueda

Los métodos PG buscan un conjunto reducido $\mathcal{P}$ de prototipos para representar el conjunto de entrenamiento $\mathcal{T}$; hay una gran variedad de direcciones de búsqueda que se pueden establecer, entre los que destacan:

- **Incremental.** Una búsqueda incremental empieza con un conjunto vacío $\mathcal{P}$. Entonces, se produce una sucesión de adiciones de prototipos representativos por cada clase a $\mathcal{P}$ [Wilson and Martinez, 2000, Triguero et al., 2012]. Una ventaja importante de esta clase de reducción es que pueden ser rápidas y necesitan menos almacenamiento durante la fase de entrenamiento que algoritmos no incrementales. Además, este tipo

de búsqueda permite establecer adecuadamente el número de prototipos necesarios para cada conjunto de datos. Sin embargo, esto podría obtener resultados adversos debido a la exigencia de un alto número de prototipos para ajustar a $\mathcal{T}$, produciendo así sobreajuste [Triguero et al., 2012].

- **Decremental.** La búsqueda decremental empieza con $\mathcal{P} = \mathcal{T}$, entonces el algoritmo empieza a reducir o modificar los prototipos en $\mathcal{P}$ [Wilson and Martinez, 2000]. Una de las ventajas observadas en los esquemas de decrecimiento es que todos los ejemplos de entrenamiento están disponibles para tomar una decisión. Por otro lado, un inconveniente de este tipo de métodos es que por lo general presentan un alto costo computacional [Wilson and Martinez, 2000].

- **Fija.** Es común utilizar un número fijo de prototipos en PG. Estos métodos establecen el número final de prototipos para $\mathcal{P}$, previamente definido por el usuario, que representarán a $\mathcal{T}$. Este es el principal inconveniente de este enfoque, aparte del hecho de que es muy dependiente de cada conjunto de datos abordado. Sin embargo, estas técnicas sólo se centran en aumentar la precisión de la clasificación [Triguero et al., 2012].

- **Mixta.** Este tipo de búsqueda combina las ventajas de lo visto anteriormente, lo que permite varias rectificaciones para resolver el problema de la búsqueda fija. Sin embargo, estas técnicas son propensas a sobreajustar los datos, y por lo general tienen alto costo computacional [Triguero et al., 2012].

1.5.2. Mecanismos de generación

Esta sección describe los diferentes mecanismos adoptados por los métodos de generación de prototipos [Triguero et al., 2012].

- **Reetiquetado de clase:** Este mecanismo de generación consiste en cambiar las etiquetas de clase de las muestras de $\mathcal{T}$, que podrían tener ruido, y que realmente pertenecen a otras clases diferentes. Su finalidad es hacer frente a instancias mal etiquetadas, ruidosas y casos atípicos en el conjunto de entrenamiento. El efecto obtenido está estrechamente relacionado con la mejora en la precisión de la generalización de los datos

de prueba, aunque la tasa de reducción se mantiene fija.

- **Basado en centroides:** Estas técnicas se basan en la generación de prototipos artificiales mediante la fusión de un conjunto de ejemplos similares. Por lo general el proceso de fusión se realiza desde el cálculo de los valores de los atributos promediados sobre un conjunto seleccionado, produciendo los llamados centroides. La identificación y selección del conjunto de ejemplos son las principales preocupaciones de los algoritmos que pertenecen a esta categoría. Estos métodos pueden obtener una tasa de reducción muy alta, pero por lo general obtienen tasas de precisión bajas.

- **División del espacio:** Este conjunto incluye las técnicas basadas en diferentes heurísticas para particionar el espacio de características, junto con varios mecanismos para definir nuevos prototipos. La idea consiste en dividir $\mathcal{T}$ en algunas regiones, que serán reemplazadas con ejemplos representativos que establecen los límites de decisión. Las capacidades de reducción de estas técnicas por lo general dependen de la cantidad de regiones que se necesitan para representar $\mathcal{T}$.

- **Ajuste de posición:** Los métodos que pertenecen a esta familia tienen como objetivo corregir la posición de un subconjunto de los prototipos de la configuración inicial mediante el uso de un procedimiento de optimización. Nuevas posiciones de los prototipos se pueden obtener mediante el uso de la idea de movimiento en el espacio d-dimensional.

En este trabajo se utiliza el mecanismo de ajuste de posición, y como técnica de optimización Programación Genética.

1.5.3. Criterios para comparar métodos de generación de prototipos

Al comparar métodos PG, hay una serie de criterios que pueden ser utilizados para comparar los puntos fuertes y débiles de cada algoritmo. Estos incluyen la reducción de $\mathcal{T}$, la precisión de clasificación, tolerancia al ruido y los requisitos de tiempo

[Triguero et al., 2012].

- **Reducción de $\mathcal{T}$:** Uno de los principales objetivos de los métodos PG es reducir los requisitos de almacenamiento y también acelerar la clasificación de muestras nuevas. Una reducción en el número de instancias almacenadas dará lugar a una reducción proporcional en el tiempo que se necesita para buscar a través de estas instancias y clasificar una nueva muestra de entrada. El porcentaje de reducción se calcula utilizando la Ecuación 1.1.

$$\text{Porcentaje de reducción} = \frac{|\mathcal{T}| - |\mathcal{P}|}{|\mathcal{T}|} \times 100 \tag{1.1}$$

  donde $|\mathcal{T}|$ es la cardinalidad del conjunto de entrenamiento, y $|\mathcal{P}|$ es cardinalidad del conjunto de prototipos.

- **Precisión:** Es una medida que indica la clasificación correcta de las muestras en un algoritmo. Para medir la precisión se utiliza la tasa de clasificación. Esta medida se calcula como el número de muestras clasificadas correctamente relativo al total de muestras (ver Figura 4.1). La tasa de clasificación ha sido, por mucho, la métrica más utilizada para evaluar el rendimiento de los clasificadores por años [Triguero et al., 2012].

- **Tolerancia al ruido**: Pueden ocurrir dos problemas principalmente en presencia de ruido. El primero es que muy pocos casos serán removidos porque se necesitan muchos casos para mantener los límites de decisión ruidosos. El segundo es que la precisión es afectada, especialmente si los casos ruidosos se retienen en lugar de los casos sin ruido, o éstos no se re-etiquetan con la clase correcta.

- **Requisitos de tiempo:** Por lo general, el proceso de aprendizaje se lleva a cabo sólo una vez en un conjunto de entrenamiento; por lo tanto, no parece ser un método de evaluación muy importante. Sin embargo, si la fase de aprendizaje toma demasiado tiempo, puede llegar a ser poco práctico para aplicaciones reales.

1.6. Trabajos relacionados

PG ha sido un campo de investigación activo en los últimos años, esto puede ser debido al gran número de aplicaciones que tienen que hacer frente a grandes volúmenes de datos en la actualidad. Una revisión reciente en este campo ha sido realizado por Triguero et al. [Triguero et al., 2012], en su estudio más de 30 métodos PG son considerados y 25 de ellos son comparados en un un extenso estudio experimental. En este estudio, el mejor algoritmo en términos de precisión fue Edición Generalizada utilizando Vecinos más Cercanos (GENN, Generalized Editing using Nearest Neighbor) [Koplowitz and Brown, 1981]. Por otra parte, en términos de reducción el mejor fue el algoritmo de Selección Clonal para la Selección de Prototipos (PSCSA, Prototype Selection Clonal Selection Algorithm) [Garain, 2008]. A pesar de ser muy eficaces en términos de precisión o reducción, estos métodos no ofrecen una compensación aceptable entre ambos objetivos.

1.6.1. Edición Generalizada utilizando Vecinos más Cercanos

Edición Generalizada utilizando Vecinos más Cercanos (GENN, por sus siglas en inglés) es un método que consiste en cambiar las etiquetas de clase a instancias de $\mathcal{T}$, que podrían tener errores y que pertenecen a otras clases diferentes, mediante la propuesta de la regla $kk'-$NN [Koplowitz and Brown, 1981].

La regla $kk'-$NN se define de acuerdo a lo siguiente:

1. Las instancias de $\mathcal{T}$ se colocan en grupos de k.

2. Si no hay una mayoría de k' de una clase, se elimina el grupo de k instancias. De lo contrario, todas las instancias se etiquetan como pertenecientes a la clase de la mayoría.

3. El conjunto $\mathcal{T}$ editado se emplea como $\mathcal{P}$, para clasificar el conjunto de validación $\mathcal{V}$ utilizando la regla NN.

1.6.2. Algoritmo de Selección Clonal para la Selección de Prototipos

El algoritmo de Selección Clonal para la Selección de Prototipos (PSCSA, por sus siglas en inglés) se basa en un sistema inmunológico artificial [de Castro and Timmis, 2002], utilizando el algoritmo de selección clonal para encontrar la posición más apropiada para un conjunto de prototipos [Garain, 2008]. El principio o teoría de selección clonal, es el mecanismo utilizado por el sistema inmunológico para describir las características básicas de una respuesta inmune a un estímulo antigénico, y se basa en la idea de que sólo aquellas células que reconocen los antígenos proliferan.

Este algoritmo implica tres etapas: inicialización de la memoria inmune, generación de clones, y selección de clones para actualizar la memoria inmune. Estas etapas se describen brevemente a continuación.

- *Inicialización de la memoria inmune:* La memoria inmune almacena en sus células los mejores antígenos que se encuentran en el proceso de búsqueda. El algoritmo de selección Clonal se inicializa mediante la representación de las instancias de $\mathcal{T}$ como antígenos, y la elección de un antígeno por cada clase para llenar la memoria inmune.

- *Generación de clones:* Después de inicializar la memoria, el proceso de búsqueda evolutiva comienza. La primera etapa consiste en un proceso de hiper-mutación, donde para cada antígeno en $\mathcal{T}$, se selecciona el antígeno más estimulado en la memoria inmune. La medida de estimulación se basa en lo cerca que ambos antígenos son (por medio de la distancia de Hamming o Euclidiana). El antígeno seleccionado de la memoria se utiliza como una matriz para el proceso de hiper-mutación, que genera un determinado número de descendientes. En el siguiente paso, se llama al procedimiento de asignación de recursos, que equilibra el número total de clones presentes en el sistema, dando la mitad de los recursos a los clones de la misma clase del antígeno actual. La otra mitad se divide por igual entre los clones de otras clases. Mientras que la precisión de la clasificación se mejora mediante la generación de clones, otros procesos de mutación (y los procedimientos de asignación de recursos) se llevan a cabo sobre los clones sobrevivientes. Esta mutación produce un menor número de clones

que depende del valor de la estimulación de cada clon progenitor.

- *Selección de clones:* Cuando no se logra ninguna mejora en la precisión de la clasificación, el mejor clon encontrado se inserta en la memoria inmune, si la memoria inmune está llena, se realiza un reemplazo con el peor antígeno presente. Luego se selecciona un nuevo antígeno del conjunto de entrenamiento para llevar a cabo la siguiente generación.

Por último, cuando el algoritmo cumple un criterio de terminación global, los antígenos contenidos en la memoria inmune se emplean como el conjunto de prototipos $\mathcal{P}$ para clasificar las muestras de prueba, utilizando el clasificador NN. El Algoritmo 1 muestra el pseudocódigo utilizado por PSCSA.

Algoritmo 1 Pseudocódigo utilizado por PSCSA

```
Inicialización de la memoria inmune.
mientras criterio de terminación no se cumpla hacer
    Proliferación I (hyper-mutación).
    Asignación de recursos.
    mientras precisión de clasificación mejore hacer
        Proliferación II (mutación).
        Asignación de recursos.
    fin mientras
    Introducir mejor antígeno en la memoria inmune.
fin mientras
regresa: La memoria inmune.
```

1.6.3. Algoritmos evolutivos para generar prototipos

En los últimos años, los métodos PG utilizan algoritmos inspirados en la naturaleza y han demostrado ser muy eficaces, en particular los algoritmos basados en algoritmos evolutivos. Fernandez et al. [Fernández and Isasi, 2004], introdujeron un enfoque para el diseño evolutivo de clasificadores NN llamado Clasificadores Evolutivos basados en Prototipos más Cercanos (ENPC, Evolutionary Nearest Prototype Classifiers). Este método empieza con una fase de inicialización simple donde el clasificador se compone por un solo prototipo. Después de eso, el clasificador evoluciona aplicando operadores básicos (mutación y

cruza), y operadores diseñados para PG (luchar, mover, morir) que tienen como objetivo proporcionar la capacidad de aprender patrones de otras regiones mediante la creación, combinación, división y eliminación de prototipos [Fernández and Isasi, 2004]. El método determina automáticamente el número óptimo de prototipos en $\mathcal{P}$. La precisión de ENPC es competitiva pero en términos de reducción no lo es.

Rosales-Perez et al. propusieron un enfoque evolutivo multi-objetivo para la generación de prototipos (EMOPG, Evolutionary Multi-Objective approach for Prototype Generation) [Rosales-Pérez et al., 2014]. Este enfoque tiene como objetivo ajustar el posicionamiento de prototipos en el espacio de entrada mediante el uso de Pareto Archived Evolution Strategy (PAES). Una solución se inicializa con un conjunto de casos potencialmente buenos (prototipos). Entonces, PAES optimiza el posicionamiento de estos casos con el objetivo de maximizar explícitamente el rendimiento de clasificación y la reducción.

Cordelia et al. fue uno de los primeros trabajos en utilizar GP para generar prototipos [Cordelia et al., 2005] y posteriormente presentaron una mejora a este método [Cordella et al., 2006]. La idea de estos trabajos, es que cada prototipo sea representante de un grupo de muestras en $\mathcal{T}$, y consiste en una expresión matemática que contiene operadores aritméticos y variables que representan características. Este enfoque es capaz de encontrar automáticamente el número de expresiones necesarias para representar todas las posibles subclases presentes en el conjunto de datos. Con el fin de generar expresiones sintácticamente correctas -prototipos- se define una gramática no determinista, donde cada prototipo se codifica como un árbol de derivación.

Otro enfoque, llamado Genetic Programming for Prototype Generation (GPPG), el cual utiliza GP para generar prototipos fue presentado en [Escalante et al., 2013]. GPPG considera operadores genéticos tradicionales y propone un nuevo operador especialmente diseñado para PG, llamado mitosis. Este operador está diseñado para corregir prototipos que están muy cerca de las instancias de varias clases. La idea es dividir un prototipo en otros dos prototipos asociados a dos clases diferentes, cuando el prototipo inicial está ha-

ciendo demasiados errores, al hacer esto, este enfoque permite optimizar automáticamente el número de prototipos; sin embargo, tiene el riesgo de sobreajuste.

1.7. Objetivos de la Tesis

1.7.1. Objetivo general

Implementar un enfoque iterativo para la generación de prototipos mediante programación génetica, y comparar su rendimiento en precisión de clasificación y reducción de $\mathcal{T}$ con técnicas del estado del arte como EMOPG [Rosales-Pérez et al., 2014], GPPG [Escalante et al., 2013] y 25 métodos presentados en [Triguero et al., 2012] en un conjunto de bases de datos de referencia.

1.7.2. Objetivos particulares

- Descargar el conjunto de bases de datos de referencia para la evaluación de métodos PG utilizado en Triguero et al. [Triguero et al., 2012].
- Implementar el enfoque propuesto de GP para la generación de prototipos usando una búsqueda incremental.
- Implementar tres métodos GP^{BAL}, GP^{UNB} y GP^{UNB2} para obtener el conjunto de clases para las cuales se generarán prototipos mediante GP.
- Obtener el rendimiento de la precisión de clasificación y reducción de $\mathcal{T}$ de GP^{BAL}, GP^{UNB} y GP^{UNB2} en el conjunto de bases de referencia.
- Mejorar la precisión obtenida por un clasificador 1NN y obtener altas tasas de reducción.
- Comparar la precisión y reducción de $\mathcal{T}$ con técnicas del estado del arte para tener un panorama general del rendimiento de nuestros métodos.

- Apoyar con este documento a personas que incursionan en el tema de clasificación de patrones mediante PG y servir como base para futuros trabajos de investigación.

1.8. Descripción de Capítulos

El capítulo 2 aborda el problema de PG, los conceptos basicos de GP, y expone con detalle el enfoque propuesto para la generación de prototipos mediante Programación Genética, así como los operadores genéticos, parámetros, el algoritmo utilizado para su implementación y se muestran dos ejemplos que ilustran como genera prototipos GP.

El capítulo 3 expone con detalle la implementación de tres algoritmos GP^{BAL}, GP^{UNB} y GP^{UNB2} para obtener el conjunto de clases para las cuales se generarán prototipos mediante GP.

El capítulo 4 muestra los resultados obtenidos en precisión de clasificación y reducción de $\mathcal{T}$ en bases de datos pequeñas, base de datos grandes, así como un promedio de bases de datos pequeñas y grandes. Por último, se presenta una comparación de los algoritmos propuestos con técnicas del estado del arte como es EMOPG, GPPG, entre otros.

Finalmente en el capítulo 5 se exponen las conclusiones, se mencionan algunas mejoras que se pueden realizar al enfoque propuesto y trabajos futuros de investigación.

Capítulo 2

Generación de Prototipos mediante Programación Genética

Este capítulo describe la principal métrica que se utiliza en la clasificación basada en la regla NN. Posteriormente se abordan los conceptos básicos de GP. Finalmente, se describe el enfoque propuesto para la generación de prototipos mediante Programación Genética y se dan ejemplos para ilustrar como GP genera prototipos.

2.1. Función de distancia

La función de distancia (o su complemento, la función de similitud) se utiliza para decidir cuales son los vecinos más cercanos a un vector de entrada y puede tener un efecto dramático en sistemas de aprendizaje basados en instancias [Wilson and Martinez, 2000]. La regla NN y sus derivados suelen utilizar la función de distancia Euclidiana, la cual se define como:

$$E(x_1, x_2) = \sqrt{\sum_{i=1}^{d} (x_{1i} - x_{2i})^2} \tag{2.1}$$

donde x_1 y x_2 son dos vectores de entrada, d es el número de atributos de entrada, x_{1i} y x_{2i} son los valores de entrada para el atributo de entrada i. En este trabajo se utiliza la distancia Euclidiana como función de distancia por ser la más utilizada en la regla NN. Sin embargo, una variedad de otras funciones de distancia también están disponibles

[Wilson and Martinez, 2000].

2.2. Programación Genética

Programación Genética es un conjunto de técnicas de computación evolutiva que permiten a las computadoras resolver problemas de forma automática. Desde su creación hace unos veintisiete años, GP se ha utilizado para resolver una amplia gama de problemas prácticos, produciendo una serie de resultados competitivos e incluso nuevas invenciones patentables. Al igual que en muchas otras áreas de la ciencia de la computación, GP está evolucionando rápidamente, con nuevas ideas, técnicas y aplicaciones que se están proponiendo constantemente [Poli et al., 2008].

En GP se evoluciona una población de programas o individuos. Es decir, generación tras generación, GP transforma estocásticamente poblaciones de programas en poblaciones nuevas (con suerte se obtiene una mejor). GP es un proceso aleatorio, por lo tanto, no se pueden garantizar los resultados. GP ha tenido mucho éxito en la evolución de formas nuevas e inesperadas en la solución de problemas, varios ejemplos se pueden consultar en [Koza, 1992, Poli et al., 2008].

Los pasos básicos en un sistema GP se pueden observar en el Algoritmo 2. Los cuales consisten en crear aleatoriamente una población inicial de programas (línea 1), donde cada programa representa una solución posible del problema a resolver, y modificar esta población mediante la selección de uno o más programas basados en su aptitud (líneas 3-4), a los cuales se les aplicarán operadores genéticos (línea 5); hasta que un criterio de terminación se cumpla (líneas 2-6). Al final del proceso el mejor programa encontrado durante la evolución es considerado la solución al problema (línea 7) [Poli et al., 2008].

Algoritmo 2 Programación Genética

1: Aleatoriamente crear una población inicial de programas con las primitivas disponibles.
2: **repetir**
3: Ejecutar cada programa y determinar su aptitud.
4: Seleccionar uno o dos programa(s) de la población con una probabilidad basada en la aptitud para que participen en las operaciones genéticas.
5: Crear un nuevo programa o más mediante la aplicación de operaciones genéticas con ciertas probabilidades especificadas.
6: **hasta que** una solución aceptable se encuentre o se cumpla alguna otra condición de paro (por ejemplo, se alcanza un número máximo de generaciones)
7: **regresa:** El mejor programa encontrado hasta el momento.

2.3. Método propuesto

Como se mencionó anteriormente, nuestro método de PG se basa en GP. Normalmente los programas de GP usan una estructura de árbol, pero este método toma en cuenta a todas las clases cuando GP está evolucionando. Por esta razón, es necesario una representación multi-árbol de los programas, tal como se representa en la Figura 2.1. Así, un programa es un conjunto de árboles, llamado bosque; donde cada árbol es un prototipo que pertenece a una cierta clase. Por lo tanto, un programa (bosque de árboles) regresa un conjunto de prototipos al evaluarlo. Se decidió implementar este bosque usando un nodo vacío, llamado salida, cuyo único propósito es juntar todos los árboles en el bosque. La restricción es que salida es siempre la raíz y todos los programas en la población lo tienen, como se puede ver en la Figura 2.1.

Cada programa en el espacio de búsqueda es construido a partir de dos conjuntos de primitivas, es decir, el conjunto de terminales $\mathbb{T}$ y el conjunto de funciones $\mathbb{F}$. $\mathbb{T}$ contiene las entradas y las constantes que son necesarias para el tipo de programas que serán evolucionados, mientras que $\mathbb{F}$ contiene las funciones.

El método de GP propuesto recibe una matriz con las características que describen los objetos. Cuando se utiliza regresión simbólica o clasificación basada en funciones descriminantes [Espejo et al., 2010], esta matriz normalmente se define como se muestra en

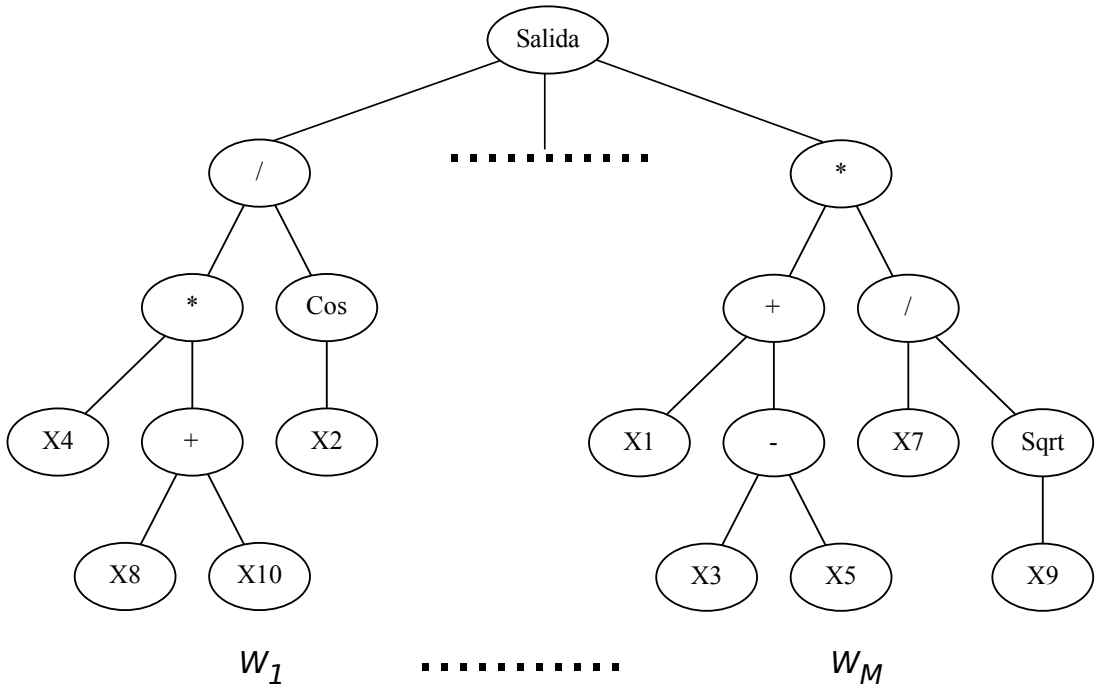

Figura 2.1: Representación de los programas usando un bosque, donde se utiliza el nodo vacío salida para unir los árboles en el bosque.

la Ecuación 2.2.

$$\Phi = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,d} \\ x_{2,1} & x_{2,2} & \dots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \dots & x_{N,d} \end{bmatrix} \tag{2.2}$$

donde N es el número de instancias en $\mathcal{T}$, d es el número de características de las instancias, al evaluar cada programa de la población se obtiene una matriz de $N \times K$, porque GP evalua por cada renglón en Φ, es decir, por cada árbol se obtiene un vector de N valores y se utiliza un bosque de K árboles (un árbol por clase).

Si lo que se quiere es clasificación de patrones basada en funciones descriminantes[1], se puede utilizar Φ, y al evaluar los programas se selecciona por cada fila de la matriz ob-

[1]Una nueva muestra de datos se clasifica como perteneciente a la clase que es la más cercana al valor de salida calculado por la función discriminante.

tenida, el elemento con el valor más cercano a la clase. Pero lo que se quiere es clasificación basada en similitud, es decir, generar prototipos. Porque se reduce el costo computacional de clasificación y ha reportado un rendimiento aceptable [Wilson and Martinez, 2000, Hastie et al., 2009, Triguero et al., 2012].

Para la clasificación basada en prototipos se necesita obtener una matriz de tamaño $d \times K$; es decir, un prototipo de dimensión d por cada clase. Por esta razón se decidió usar la transpuesta de Φ como conjunto de terminales (ver Ecuación 2.3).

$$\Phi^T = \begin{bmatrix} x_{1,1} & x_{2,1} & \cdots & x_{N,1} \\ x_{1,2} & x_{2,2} & \cdots & x_{N,2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1,d} & x_{2,d} & \cdots & x_{N,d} \end{bmatrix} \tag{2.3}$$

De esta forma, al evaluar cada programa en la población se obtiene una matriz de tamaño $d \times K$, porque cada árbol regresa d valores (uno por cada renglón en Φ^T) y un programa es un bosque de K árboles. La distancia Euclidiana se utiliza como función de distancia (Ecuación 2.1), entre las columnas de la matriz (cada columna representa un prototipo) y la muestra a clasificar, y la muestra es etiquetada con la columna con la cual se obtiene la distancia más pequeña.

Por otra parte, $\mathbb{F}$ esta formada por funciones aritméticas (por ejemplo, $+$, $-$, $\times$, $\div$), funciones trascendentales y min. La función min está implementada usando operadores aritméticos y la función exponencial; como se muestra en la Ecuación 2.4.

$$min(x, y) = \frac{y - x}{1 + e^{-100*(x-y)}} + x \tag{2.4}$$

2.3.1. Operadores Genéticos

Durante el proceso evolutivo se utilizaron los siguientes operadores genéticos:

- **Cruza de sub-árbol:** La creación de un programa hijo mediante la combinación de los sub-árboles seleccionadas aleatoriamente a partir de dos programas padres [Koza, 1992, Poli et al., 2008].

- **Mutación de sub-árbol:** La creación de un hijo mediante la selección aleatoria de un sub-árbol en un padre, el cual es reemplazado por un árbol generado aleatoriamente [Koza, 1992, Poli et al., 2008].

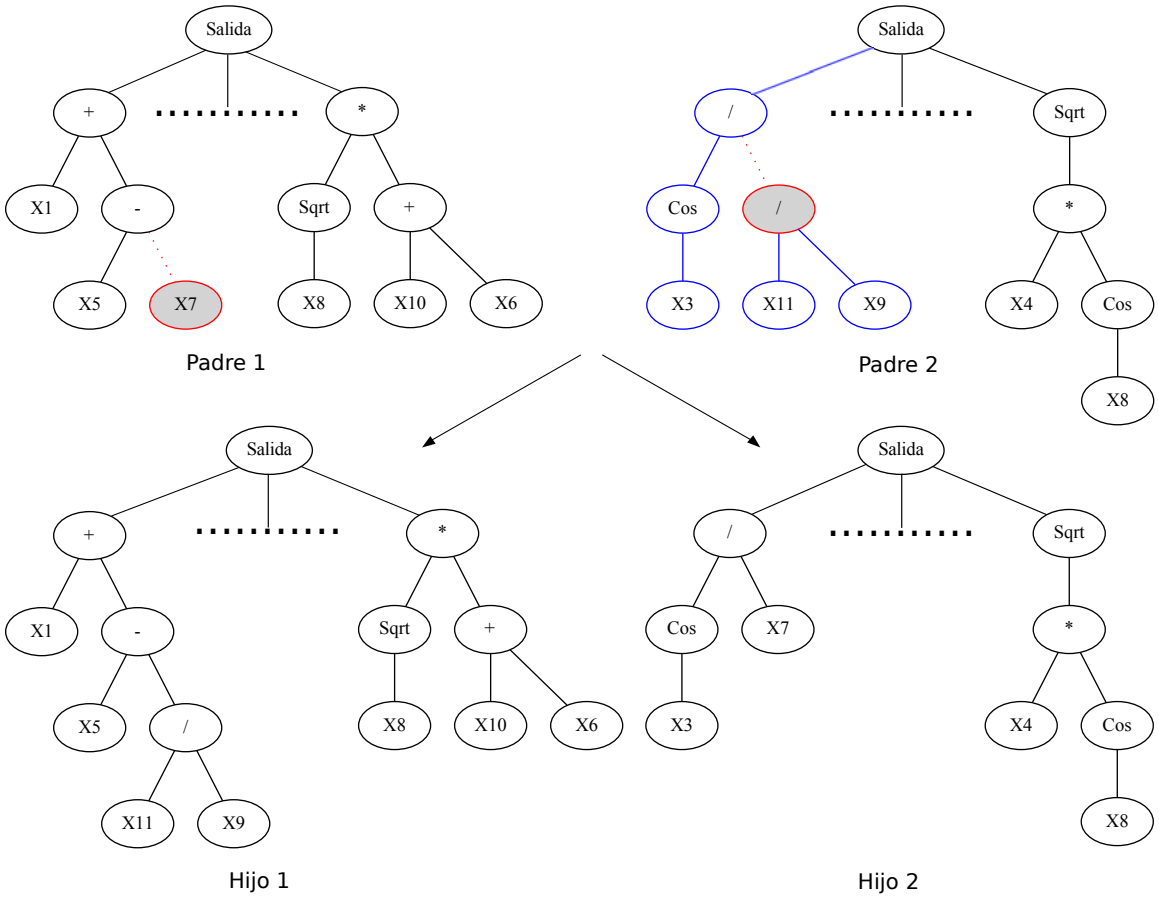

Figura 2.2: Cruza de sub-árboles. La línea punteada indica el punto de cruza.

La Figura 2.2 muestra el proceso de cruza de sub-árboles. Hay dos padres 1 y 2, en padre 1 un nodo del árbol es seleccionado aleatoriamente como punto de cruza (línea roja punteada), el nodo "Salida"que se utiliza para unir los árboles no puede ser seleccionado como punto de cruza; una vez que este punto es seleccionado, en padre 2 otro nodo es

seleccionado aleatoriamente como punto de cruza (línea roja punteada), pero asegurándose que el árbol donde se selecciona el nodo pertenezca a la misma clase que al punto de cruza seleccionado del padre 1 (para el ejemplo de la Figura 2.2, el segundo punto de cruza sólo se puede seleccionar a partir del árbol de color azul). Los sub-árboles debajo del punto de cruza son intercambiados creando dos hijos. Un procedimiento similar es realizado para la mutación, la diferencia es que el padre 2 es un programa aleatorio y el punto de cruza siempre es la raiz del árbol.

Al igual que con la mayoría de los algoritmos evolutivos, los operadores genéticos en GP se aplican a los programas que probabilísticamente se seleccionan de acuerdo a una función de aptitud. Es decir, los mejores programas son más propensos a tener más programas hijos que programas inferiores. El método más comúnmente empleado para la selección de los individuos en GP es la selección por torneo [Poli et al., 2008]. Por esta razón, en este trabajo se optó por utilizar este método de selección, así como utilizar solo los operadores genéticos estándar (cruza y mutación).

En el proceso evolutivo se utilizó una población de estado estable, es decir, una "generación"se concluye cuando han pasado tantos eventos de cruza/mutación como el tamaño de la población. El resto del proceso evolutivo es similar a algoritmos genéticos.

2.3.2. Función de aptitud

La población inicial de programas creados al azar, por lo general son muy pobres en la solución de cualquier problema. Sin embargo, incluso en la poblacion creada al azar, algunos programas son mejores que otros. Para ello es necesario contar con una función de aptitud que evalue los programas para saber cual es más apto. En este trabajo se utiliza como función de aptitud, la precisión de clasificación obtenida mediante la regla kNN (con k=1); clasificando las instancias de $\mathcal{T}$ usando el conjunto de prototipos generados $\mathcal{P}$ mediante la distancia Euclidiana (Ecuación 2.1).

2.3.3. Resumen de parámetros utilizados

Como se ha mencionado anteriormente, los parámetros utilizados en GP son los estándar que se han utilizado en trabajos anteriores [Koza, 1992, Poli et al., 2008]. Claramente, si tratamos de maximizar la precisión en el conjunto de entrenamiento $\mathcal{T}$ terminaríamos con un conjunto de prototipos $\mathcal{P}$ sobre-entrenado, por esta razón hemos decidido utilizar un pequeño número de generaciones en el proceso evolutivo y limitar la longitud máxima de los árboles a 1024. En la Tabla 2.1, se muestra un resumen de los parámetros que se utilizan en GP.

Tabla 2.1: Parámetros de GP

Parámetro	**Valor**
Tamaño de la Población	1000
Número de Generaciones	50
Conjunto de Funciones ($\mathcal{F}$)	$\{+, -, \times, \div, \lvert\cdot\rvert, \sin, \cos, \sqrt{}, \text{mín}\}$
Probabilidad de Cruza	90 %
Probabilidad de Mutación	10 %
Profundidad de mutación	Aleatorio $\in [1, 5]$
Tamaño máximo	1024
Método de selección	Torneo de tamaño 2
Tipo de población	Estado estable

2.4. Algoritmo GP propuesto para generar prototipos

La modificación principal que se hace al algoritmo de GP, Algoritmo 3 es que se diseña para generar prototipos solo para las clases que lo necesitan (los enfoques propuestos para definir cuales son las clases que necesitan más prototipos se describen en el capítulo 3). El algoritmo tiene parámetros obligatorios que se requieren para la generación de prototipos y parámetros que se necesitan de ejecuciones previas, y un parámetro opcional, el cual se utiliza para inicializar la población con el mejor individuo de ejecuciones previas.

El Algoritmo 3 empieza verificando si GP se va a inicializar con el mejor individuo encontrado en previas ejecuciones (líneas de 1 - 6). Una vez que la población se inicializa,

Algoritmo 3 Algoritmo propuesto de GP para generar prototipos

Entrada: Φ^T : Matriz con las características que describen los objetos;
g : Número de generaciones;
s : Número de programas en la población;
$\mathcal{P}$: Prototipos generados en ejecuciones anteriores;
t : Objetivo;
$vector\_c^*$: Vector de etiquetas de los prototipos generados en ejecuciones anteriores;
$vector\_c$: Vector de clases para las cuales se generarán prototipos;
Parámetros opcionales: $best\_ind$: mejor individuo de ejecuciones anteriores;
$i \leftarrow 0$;
$f^* \leftarrow -1$;

si $best\_ind$ está definido **entonces**
  $\mathbb{P} \leftarrow$ Inicializar la población con $s-1$ programas;
  $\mathbb{P}[s] \leftarrow best\_ind$
si no
  $\mathbb{P} \leftarrow$ Inicializar la población con s programas;
  {Cada programa es un bosque con $|vector\_c|$ árboles}
fin si
mientras $i \leq (g \times |\mathbb{P}|)$ **hacer**
  $i \leftarrow i + 1$;
  para $j = 1$ hasta s **hacer**
    $\mathcal{P}_j \leftarrow evaluar(\mathbb{P}[j])$
    $\mathcal{P}_j \leftarrow concatenar(\mathcal{P}, \mathcal{P}_j)$
    $\mathcal{D} \leftarrow distancia\_euclideana(\Phi^T, \mathcal{P}_j)$
    $etiquetas \leftarrow concatenar(vector\_c^*, vector\_c)$
    $c \leftarrow etiquetas$[(argumento mínimo de $\mathcal{D}$ por renglón)]
    $f[j] \leftarrow -1 + (suma(c == t)/|t|)$
    {Se calcula la tasa de clasificación.}
    si $f[j] \geq f^*$ **entonces**
      $f^* \leftarrow f[j]$
      $best\_ind \leftarrow \mathbb{P}[j]$
      $\mathcal{P}^* \leftarrow \mathcal{P}_j$
    fin si
  fin para
  Se aplica el operador de selección;
  $\mathbb{P} \leftarrow$ Se aplica el operador de cruza con una cierta probabilidad;
  $\mathbb{P} \leftarrow$ Se aplica el operador de mutación con una cierta probabilidad;
fin mientras
regresa: $[\mathcal{P}^*\ f^*\ best\_ind]$

cada individuo en la población se evalua para obtener el conjunto de prototipos que genera (línea 10), los cuales se concatenan con los prototipos generados en ejecuciones previas (línea 11). Posteriormente, se aplica la distancia euclidiana entre Φ^T y $\mathcal{P}$, las columnas de Φ^T se etiquetan con la etiqueta del prototipo con el cual se obtiene la menor distancia (líneas 12 - 14). Para calcular la aptitud del individuo se utiliza la tasa de clasificación (línea 15) y se actualiza al mejor individuo generado hasta el momento así como su aptitud y el conjunto de prototipos que genera (líneas 16 - 20). En el siguiente paso, la población se modifica aplicando los operadores genéticos de cruza y mutación (líneas 23 - 24). Este proceso se repite hasta que el número de aplicar cruza/mutación a la población es mayor al número de generaciones por el tamaño de población, porque se utiliza población de estado estable (líneas 7 - 25).

Al final del proceso evolutivo se regresa al mejor individuo encontrado, así como el conjunto de prototipos que genera concatenados con prototipos de ejecuciones previas y su aptitud (línea 26 del Algoritmo 3).

Como se puede observar GPPG tiene algunas similitudes con la contribución realizada de este trabajo; sin embargo, tiene la siguientes diferencias significantes:

- El enfoque propuesto no requiere usar operadores genéticos novedosos (que son caros computacionalmente), solo usa un sistema GP estándar con operadores genéticos tradicionales.

- Realmente, el método propuesto es muy fácil de implementar y mucho más eficiente que GPPG.

- El enfoque propuesto genera iterativamente prototipos, es decir, con el fin de aumentar el número de prototipos, uno necesita ejecutar una instancia del Algoritmo 3.

- Los resultados experimentales muestran, que el enfoque propuesto supera GPPG significativamente en términos de precisión y sin embargo obtiene un rendimiento de reducción comparable.

2.5. Generando prototipos usando Programación Genética

Esta sección se muestran dos ejemplos con datos artificiales, con el fin de ilustrar como genera prototipos GP y como se interpretan los resultados obtenidos.

2.5.1. Ejemplo 1

Sea un conjunto de entrenamiento de instancias etiquetadas $\mathcal{T}$, donde $\mathcal{T}$ contiene 36 instancias definidas por 2 características divididas en 3 clases (ver Ecuación 2.5). La Figura 2.3.a muestra gráficamente la distribución de $\mathcal{T}$ por clase. El objetivo es reducir el número de muestras mediante la búsqueda de muestras representativas llamadas prototipos mediante GP.

$$\mathcal{T} = \left\{ \begin{array}{cc} (3.3, 3.5) & 2 \\ (-0.0, 2.3) & 0 \\ (2.0, -0.9) & 1 \\ (3.1, 0.0) & 1 \\ (3.0, -0.6) & 1 \\ (3.3, 1.1) & 1 \\ (2.6, -0.6) & 1 \\ (4.6, 4.5) & 2 \\ (2.5, 3.7) & 2 \\ (3.6, 4.0) & 2 \\ (-1.7, 2.4) & 0 \\ (-0.4, 1.5) & 0 \\ (-0.7, 1.9) & 0 \\ (-0.2, 0.3) & 0 \\ (3.7, 0.7) & 1 \\ (2.1, 4.3) & 2 \\ (1.0, 1.8) & 0 \\ (2.7, -0.3) & 1 \\ (1.7, 0.2) & 1 \\ (5.2, 3.5) & 2 \\ (2.5, 3.1) & 2 \\ (3.7, -0.2) & 1 \\ (2.3, 1.9) & 2 \\ (4.1, 2.4) & 2 \\ (3.9, 4.2) & 2 \\ (4.9, 3.3) & 2 \\ (1.6, 1.3) & 1 \\ (1.7, 3.0) & 2 \\ (0.5, 1.7) & 0 \\ (0.9, 1.1) & 0 \\ (3.5, 4.1) & 2 \\ (3.0, 2.5) & 2 \\ (3.4, 1.9) & 2 \\ (-0.9, 2.2) & 0 \\ (2.7, 1.0) & 1 \\ (0.1, 2.9) & 0 \end{array} \right\} \tag{2.5}$$

Solución: Para generar prototipos mediante GP, se necesita definir el conjunto de terminales Φ^T, el cuál queda definido por la Ecuación 2.6, donde las columnas son las instancias en $\mathcal{T}$ y los renglones son las características de las instancias, es decir, Φ^T es una matriz de tamaño 2×36. La meta de GP es clasificar correctamente a todas las instancias con los prototipos generados utilizando el vector objetivo t, definido por la Ecuación 2.7. Los parámetros que se utilizan son los definidos en la Tabla 2.1, pero por la simplicidad de este problema se utiliza una población de 100 individuos, 10 generaciones y un tamaño máximo del individuo de 50.

$$\Phi^T = \left[\begin{array}{cccccccccccc} 3.3 & 0.0 & 2.0 & 3.1 & 3.0 & 3.3 & 2.6 & 4.6 & 2.5 & 3.6 & -1.7 & -0.4 \\ 3.5 & 2.3 & -0.9 & 0.0 & -0.6 & 1.1 & -0.6 & 4.5 & 3.7 & 4.0 & 2.4 & 1.5 \end{array}\right.$$
$$\begin{array}{cccccccccccc} -0.7 & -0.2 & 3.7 & 2.1 & 1.0 & 2.7 & 1.7 & 5.2 & 2.5 & 3.7 & 2.3 & 4.1 \\ 1.9 & 0.3 & 0.7 & 4.3 & 1.8 & -0.3 & 0.2 & 3.5 & 3.1 & -0.2 & 1.9 & 2.4 \end{array} \tag{2.6}$$
$$\left.\begin{array}{cccccccccccc} 3.9 & 4.9 & 1.6 & 1.7 & 0.5 & 0.9 & 3.5 & 3.0 & 3.4 & -0.9 & 2.7 & 0.1 \\ 4.2 & 3.3 & 1.3 & 3.0 & 1.7 & 1.1 & 4.1 & 2.5 & 1.9 & 2.2 & 1.0 & 2.9 \end{array}\right]$$

$$t = [\ 2\ 0\ 1\ 1\ 1\ 1\ 1\ 2\ 2\ 2\ 0\ 0\ 0\ 0\ 1\ 2\ 0\ 1\ 1\ 2\ 2\ 1\ 2\ 2\ 2\ 2\ 1\ 2\ 0\ 0\ 2\ 2\ 2\ 0\ 1\ 0\ ] \tag{2.7}$$

Una vez que se tiene definido el conjunto de terminales, y el resto de los parámetros de GP, se ejecuta el Algoritmo 3. Al final del proceso evolutivo se selecciona el individuo con mejor aptitud. En este caso, el mejor programa que genera GP es el definido por el bosque de 3 árboles (uno por cada clase) de la Figura 2.4, en el cual cada árbol esta definido por solo un terminal.

Evaluación del mejor individuo:

Los individuos se evalúan por cada renglón en Φ^T, para este caso 2 veces. El primer árbol de la Figura 2.4 es la expresión $(X33)$, donde Xi es la columna i de Φ^T. El segundo árbol es la expresión $(X2)$ y el tercer árbol es la expresión $(X8)$. Al evaluar para el primer renglón se obtienen los siguientes valores $(X33) = -0.9$, $(X2) = 2.0$ y $(X8) = 2.5$. Para el segundo renglón se obtiene $(X33) = 2.2$, $(X2) = -0.9$ y $(X8) = 3.7$. Por lo cual, al evaluar el

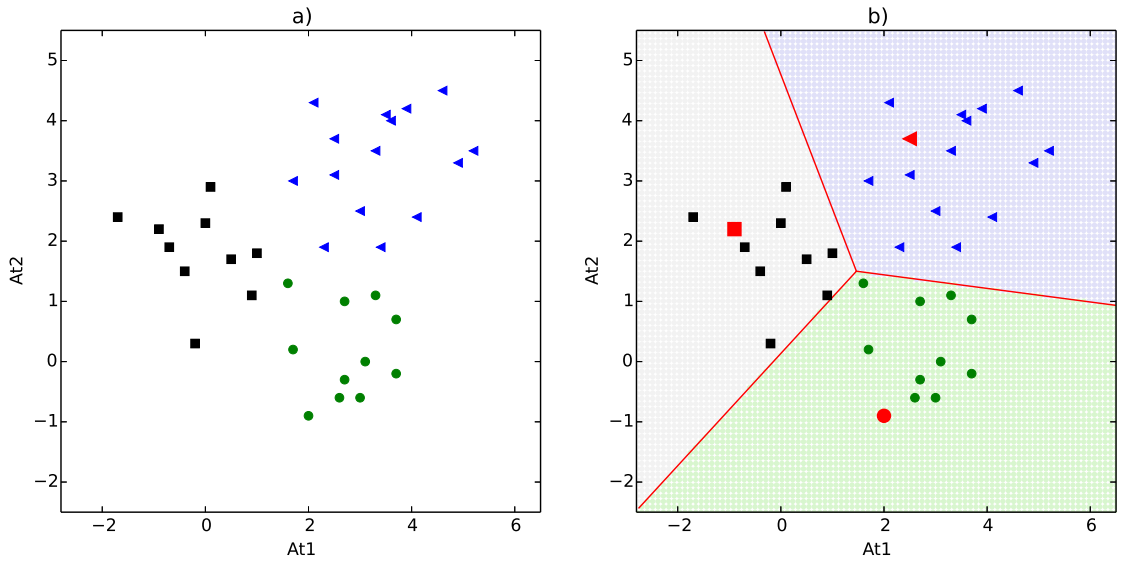

Figura 2.3: a) Conjunto de entrenamiento; donde las instancias de la clase 0 se muestran con cuadros negros, instancias de la clase 1 con círculos verdes e instancias de la clase 2 con triángulos azules, b) Segmentación del espacio de búsqueda utilizando el conjunto de prototipos generados y la distancia euclidiana.

individuo de la Figura 2.4 nos regresa la matriz de la Ecuación 2.8. Donde cada columna de la matriz representa un prototipo.

$$\text{Evaluación del mejor individuo} = \begin{bmatrix} -0.9 & 2.0 & 2.5 \\ 2.2 & -0.9 & 3.7 \end{bmatrix} \tag{2.8}$$

Para este ejemplo, con solo un prototipo por clase se logra clasificar correctamente a $\mathcal{T}$ (ver Figura 2.3.b), por lo cual no es necesario buscar más prototipos. En la Figura 2.3.b se muestra gráficamente la posición de los prototipos encontrados por GP, así como la segmentación realizada del espacio de búsqueda cuando se utiliza la distancia euclidiana, donde se aprecia claramente que cada prototipo clasifica correctamente las muestras de su clase.

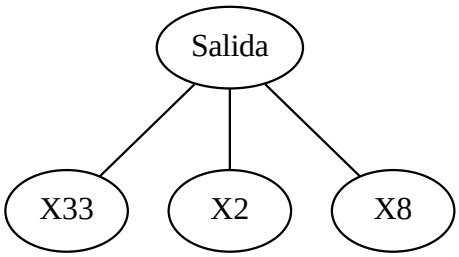

Figura 2.4: Genotipo del mejor individuo encontrado al utilizar el Algoritmo 3.

2.5.2. Ejemplo 2

El problema a resolver es similar al presentado en la sección 2.5.1, la diferencia es que se utiliza un conjunto de entrenamiento diferente. El conjunto de entrenamiento utilizado en este ejemplo contiene 130 instancias definidas por 2 características divididas en 2 clases. La Figura 2.5.a muestra gráficamente la distribución de $\mathcal{T}$ por clase. Como se puede apreciar el problema es más difícil que el anterior, por que hay traslape entre clases y algunas muestras contienen ruido, además que la forma de las clases es más difícil de modelar. Aunque obviamente es fácil comparado con problemas reales.

La solución es igual al problema de la sección 2.5.1, se tiene que definir Φ^T y t, posteriormente se ejecuta el Algoritmo 3 con los mismos parámetros del ejemplo anterior. Sin embargo, por la distribución de las clases el algoritmo se ejecutó 5 veces para mejorar la aptitud. La primera vez que se ejecuta el mejor individuo que se obtiene se muestra en la Figura 2.6.1, las Figuras 2.6.2 - 2.6.5 muestran los genotipos de los mejores individuos obtenidos por las ejecuciones de la 2-5 respectivamente. En cada ejecución se van almacenando los prototipos generados y estos son utilizados en futuras ejecuciones del Algoritmo 3.

Primer Ejecución

Evaluación del mejor individuo:

El mejor individuo de la primera ejecución del algoritmo se muestra en la Figura 2.6.1. El primer árbol de la Figura 2.6.1 es la expresión $(X17 + 0.21)/0.86$. El segundo árbol es la expresión $(X75)$. Al evaluar para el primer renglón de Φ^T se obtienen los siguientes

valores $(X17 + 0.21)/0.86 = 286.31$ y $(X75) = 171.0$. Para el segundo renglón se obtiene $(X17 + 0.21)/0.86 = 190.95$ y $(X75) = 177.0$. Por lo cual, al evaluar el individuo de la Figura 2.6.1 nos regresa la matriz de la Ecuación 2.9.

$$\text{Evaluación del mejor individuo} = \begin{bmatrix} 286.31 & 171.0 \\ 190.95 & 177.0 \end{bmatrix} \tag{2.9}$$

Segunda Ejecución

Evaluación del mejor individuo:

El mejor individuo de la segunda ejecución del algoritmo se muestra en la Figura 2.6.2. El primer árbol de la Figura 2.6.2 es la expresión $(X129)$. El segundo árbol es la expresión $|X116|$. Al evaluar para el primer renglón de Φ^T se obtienen los siguientes valores $(X129) = 225.0$ y $|X116| = 218.0$. Para el segundo renglón se obtiene $(X129) = 114.0$ y $|X116| = 54.0$. Por lo cual, al evaluar el individuo de la Figura 2.6.2 nos regresa la matriz de la Ecuación 2.10. Como se puede observar en la matriz los prototipos que genera el individuo se concatenan con prototipos de previas ejecuciones.

$$\text{Evaluación del mejor individuo} = \begin{bmatrix} 286.31 & 171.0 & 225.0 & 218.0 \\ 190.95 & 177.0 & 114.0 & 54.0 \end{bmatrix} \tag{2.10}$$

Tercera Ejecución

Evaluación del mejor individuo:

El mejor individuo de la tercera ejecución del algoritmo se muestra en la Figura 2.6.3. El primer árbol de la Figura 2.6.3 es la expresión $(X30)$. El segundo árbol es la expresión $(X111)$. Al evaluar para el primer renglón de Φ^T se obtienen los siguientes valores $(X30) = 262.0$ y $(X111) = 225.0$. Para el segundo renglón se obtiene $(X30) = 157.0$ y $(X111) = 71.0$. Por lo cual, al evaluar el individuo de la Figura 2.6.3 nos regresa la matriz de la Ecuación 2.11. Como se puede observar en la matriz los prototipos que genera el individuo se concatenan con prototipos de previas ejecuciones.

$$\text{Eval.} = \begin{bmatrix} 286.31 & 171.0 & 225.0 & 218.0 & 262.0 & 225.0 \\ 190.95 & 177.0 & 114.0 & 54.0 & 157.0 & 71.0 \end{bmatrix} \tag{2.11}$$

Cuarta Ejecución

Evaluación del mejor individuo:

El mejor individuo de la cuarta ejecución del algoritmo se muestra en la Figura 2.6.4. El primer árbol de la Figura 2.6.4 es la expresión $(X26)$. El segundo árbol es la expresión $(X54)$. Al evaluar para el primer renglón de Φ^T se obtienen los siguientes valores $(X26) = 233.0$ y $(X54) = 224.0$. Para el segundo renglón se obtiene $(X26) = 136.0$ y $(X54) = 143.0$. Por lo cual, al evaluar el individuo de la Figura 2.6.4 nos regresa la matriz de la Ecuación 2.12. Como se puede observar en la matriz los prototipos que genera el individuo se concatenan con prototipos de previas ejecuciones.

$$\text{Eval.} = \begin{bmatrix} 286.31 & 171.0 & 225.0 & 218.0 & 262.0 & 225.0 & 223.0 & 224.0 \\ 190.95 & 177.0 & 114.0 & 54.0 & 157.0 & 71.0 & 136.0 & 143.0 \end{bmatrix} \tag{2.12}$$

Quinta Ejecución

Evaluación del mejor individuo:

El mejor individuo de la quinta ejecución del algoritmo se muestra en la Figura 2.6.5. El primer árbol de la Figura 2.6.5 es la expresión $(X19)$. El segundo árbol es la expresión $(X63)$. Al evaluar para el primer renglón de Φ^T se obtienen los siguientes valores $(X19) = 257.0$ y $(X63) = 214.0$. Para el segundo renglón se obtiene $(X19) = 186.0$ y $(X63) = 170.0$. Por lo cual, al evaluar el individuo de la Figura 2.6.5 nos regresa la matriz de la Ecuación 2.13. Como se puede observar en la matriz los prototipos que genera el individuo se concatenan con prototipos de previas ejecuciones.

$$\text{Eval.} = \begin{bmatrix} 286.31 & 171 & 225 & 218 & 262 & 225 & 223 & 224 & 257 & 214 \\ 190.95 & 177 & 114 & 54 & 157 & 71 & 136 & 143 & 186 & 170 \end{bmatrix} \tag{2.13}$$

Al final de ejecutar el algoritmo 5 veces iterativamente se obtiene el conjunto de prototipos de la Ecuación 2.13. En las Figuras 2.5.b - 2.5.f se muestra gráficamente la posición de los prototipos encontrados por GP, así como la segmentación realizada del espacio de búsqueda cuando se utiliza la distancia euclidiana, para 1-5 ejecuciones del Algoritmo 3 respectivamente. En estas figuras se puede observar que para un prototipo por clase la precisión de clasificación en $\mathcal{T}$ es baja, sin embargo al agregar más prototipos se segmentan mejor las clases, por lo tanto, la clasificación mejora considerablemente.

2.6. Conclusiones

Con base al interés por desarrollar un sistema para la clasificación de patrones basados en la regla NN, es que se ha propuesto un nuevo enfoque para generar prototipos mediante GP usando búsqueda incremental por las ventajas que esta presenta [Wilson and Martinez, 2000, Triguero et al., 2012].

El algoritmo propuesto es simple y fácil de implementar debido a que no utiliza operadores genéticos novedosos, al contrario utiliza solo operadores tradicionales.

Como ya se mencionó anteriormente, el algoritmo esta diseñado para generar prototipos de manera iterativa, y puede generar solo prototipos para las clases que lo requieran. Entonces es necesario un mecanismo que encuentre el conjunto de clases para las cuales se van a estar generando prototipos. En el siguiente capítulo se abordan tres diferentes algoritmos que resuelven este problema, así como los procedimientos completos que se utilizan para generar prototipos.

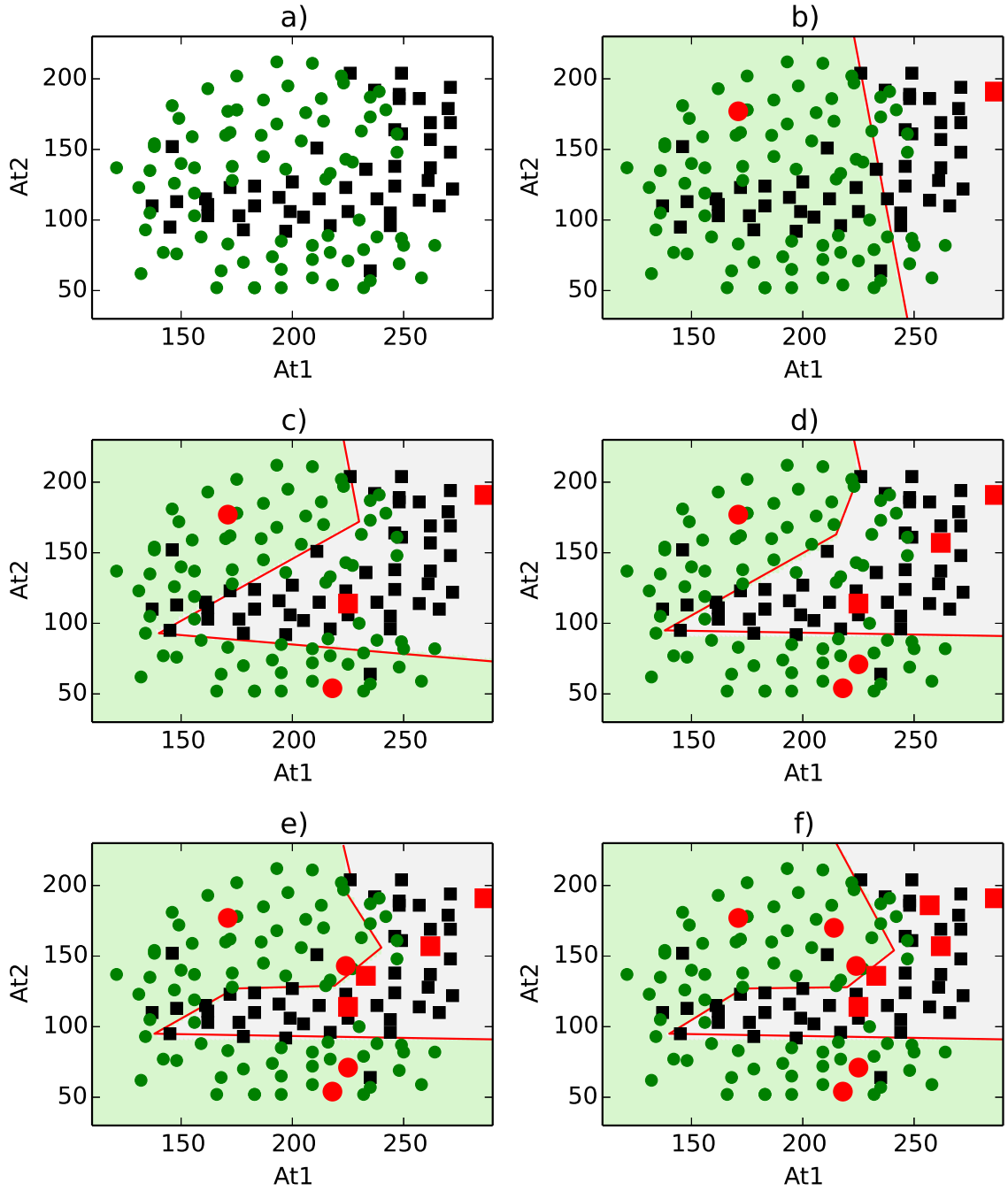

Figura 2.5: a) Conjunto de entrenamiento; donde las instancias de la clase 0 se muestran con cuadros negros e instancias de la clase 1 con círculos verdes, b) - f) Segmentación del espacio de búsqueda utilizando $1, 2, 3, 4, 5$ prototipo(s) por clase y distancia euclidiana respectivamente.

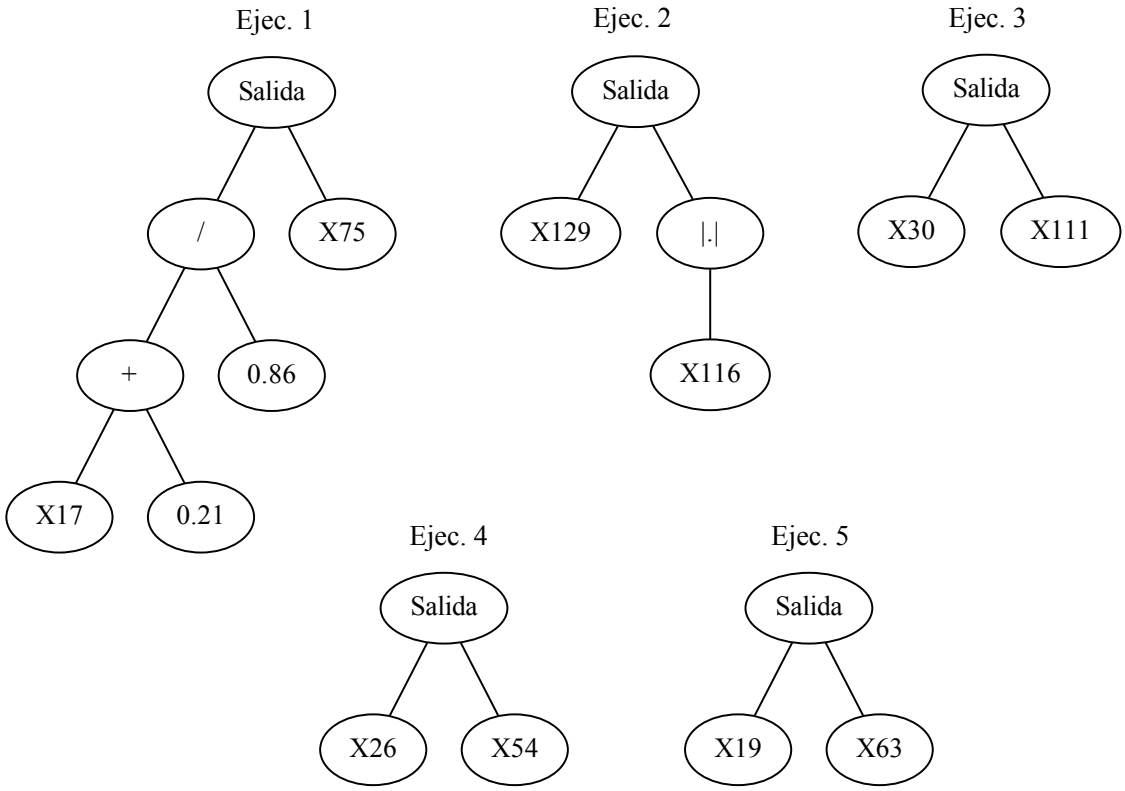

Figura 2.6: 5 Genotipos de los mejores individuos encontrados al ejecutar el Algoritmo 3, 5 veces iterativamente.

Capítulo 3

Algoritmos Implementados

En el capítulo anterior se abordó la metodología propuesta para generar prototipos mediante GP, así como los parámetros utilizados. En este capítulo se abordan tres diferentes enfoques para generar prototipos utilizando una búsqueda incremental con GP. El primer enfoque genera prototipos para todas las clases, el cual se llama GP^{BAL}. El segundo enfoque genera prototipos solo para las clases que tienen muchos errores, el cual se llama GP^{UNB}. El tercer enfoque genera prototipos para las clases que aún no clasifican correctamente a todas sus muestras. Las tres variantes usan la misma representación y función de aptitud, pero difieren en la forma en como el proceso iterativo es realizado.

3.1. GP^{BAL}

El primer método implementado se llama GP^{BAL}, este método siempre genera un prototipo por clase en todas las iteraciones, por esta razón, se utiliza el superíndice BAL de balanceado. El método empieza con un conjunto vacío de prototipos $\mathcal{P}^*$, y se ejecuta iterativamente el Algoritmo 3. Este proceso se muestra en el Algoritmo 4, consiste en tres etapas.

- *Conjunto de prototipos a generar:* se define el conjunto de clases para las cuales se generarán prototipos usando el Algoritmo 3 (línea 1).
- *Generación de prototipos:* la población de programas se inicializa aleatoriamente

usando una *semilla* (líneas 3-4). Mientras la aptitud obtenida no supere la obtenida previamente y el número de *semillas* sea menor o igual a cierto número — en esta contribución se utilizaron 2 *semillas* para evitar el problema de sobre entrenamiento — se incrementa el valor de la *semilla* y se vuelve a ejecutar otra instancia del Algoritmo 3, usando la *semilla* actual para obtener otros prototipos, asegurando que la nueva población contenga al mejor individuo de la ejecución anterior (líneas 5-8).

- *Actualización de prototipos:* si se mejora la aptitud, se actualiza el conjunto de prototipos óptimo (líneas 9-14), de lo contrario el algoritmo termina (líneas 14-16). Se vuelven a generar prototipos y se actualizan hasta τ número de veces, donde $\tau \geq 1$, es decir, debe haber mínimo un prototipo por clase para que ésta pueda ser clasificada o hasta que no se encuentre un conjunto de prototipos que mejore la clasificación (líneas 2 - 17).

Al final del proceso evolutivo se regresa el conjunto de prototipos óptimo encontrado así como las etiquetas de las clases a las cuales pertenecen (línea 18). GP^{BAL} siempre genera el mismo número de prototipos por clase, porque es una idea simple e intuitiva, pero se debe notar que puede haber instancias de algunas clases que están correctamente clasificadas mientras que otras instancias de otras clases están clasificadas incorrectamente. Esto significa que unas clases necesitan más prototipos que otras, esto podría deberse a que sus muestras están más dispersas. Por esta razón, se proponen dos métodos para generar prototipos de forma desbalanceada, que se describen a continuación.

3.2. GP^{UNB}

El segundo enfoque propuesto se llama GP^{UNB}, el cual consiste en generar máximo ($\tau \times$ *número de clases*) prototipos; como se puede ver en Algoritmo 5, se utiliza el superíndice UNB del inglés unbalanced que significa desbalanceado, ya que este algoritmo no necesariamente genera el mismo número de prototipos por clase.

Algoritmo 4 Algoritmo utilizado por GP^{BAL}

Entrada: δ : Parámetros de GP;
τ : Máximo número de prototipos;
N_s : Número se semillas;
$i \leftarrow 0$; $\mathcal{P}^* \leftarrow []$; $vector\_c^* \leftarrow []$;
$f^* \leftarrow -1.0$; $enc \leftarrow$ **cierto**;
$C \leftarrow$ Conjunto de clases;
Garantizar: $\tau >= 1$;

1: $vector\_c \leftarrow C$;
2: **mientras** $i < \tau$ **and** enc **hacer**
3: $semilla \leftarrow 0$;
4: $[\mathcal{P}\ f\ best\_ind] \leftarrow GP(\delta, semilla, \mathcal{P}^*, vector\_c^*, vector\_c)$;
5: **mientras** $f^* \geq f$ **and** $semilla \leq N_s$ **hacer**
6: $semilla \leftarrow semilla + 1$;
7: $[\mathcal{P}\ f\ best\_ind] \leftarrow GP(\delta, semilla, \mathcal{P}^*, vector\_c^*, vector\_c, best\_ind)$;
8: **fin mientras**
9: **si** $f > f^*$ **entonces**
10: $\mathcal{P}^* \leftarrow \mathcal{P}$;
11: $f^* \leftarrow f$;
12: $vector\_c^* = concatenar(vector\_c^*, vector\_c)$
13: $i \leftarrow i + 1$;
14: **si no**
15: $enc \leftarrow$ **falso**;
16: **fin si**
17: **fin mientras**
18: **regresa:** [P^*, $vector\_c^*$]

Este método trata de balancear el porcentaje de clasificación por clase (fit_class) usando como umbral a p_f. Para realizar esto es necesario calcular la precisión de clasificación por clase, y así obtener la diferencia entre la aptitud máxima y mínima. Si la diferencia es mayor o igual que p_f, significa que hay clases que tienen muy baja aptitud comparadas con las otras, y esto debería ser compensado mediante la generación de prototipos solo para las clases cuya aptitud es menor o igual a $(\text{mín} + p_g \times (\text{máx} - \text{mín}))$. Donde: p_g es el umbral de discriminación. En los experimentos realizados, se probaron diferentes valores de p_f y p_g, y los mejores resultados se obtuvieron con $p_f = 0.1$ y $p_g = 0.9$. Por otra parte, si todas las clases tienen similar aptitud, los prototipos son solo generados para las clases con instancias mal clasificadas (Algoritmo 5, líneas 10-22).

El método GP^{UNB} consiste en tres etapas, similares a GP^{BAL}.

- *Conjunto de prototipos a generar:* se define el conjunto de clases para las cuales se generarán prototipos (líneas 3-23). Si no se han generado prototipos, genera un prototipo por clase, de lo contrario, genera prototipos solo para las clases que tienen una precisión baja comparada con otras clases. Sin embargo, si todas las clases tienen una precisión similar, genera prototipos para todas las clases que aún no clasifican correctamente todas sus muestras. Si no se encuentra un conjunto de prototipos a generar el algoritmo termina (línea 24).

- *Generación de prototipos:* la población de programas se inicializa aleatoriamente usando una *semilla* (líneas 2 y 25). Mientras la aptitud obtenida no supere la obtenida previamente y el número de *semillas* sea menor o igual a cierto número, se incrementa el valor de la *semilla* y se vuelve a ejecutar otra instancia del Algoritmo 3, usando la *semilla* actual para obtener otros prototipos, asegurando que la nueva población contenga al mejor individuo de la ejecución anterior (líneas 26-29).

- *Actualización de prototipos:* si se mejora la aptitud, se actualiza el conjunto de prototipos óptimo (líneas 30-34), de lo contrario el algoritmo termina (líneas 34-36). Este procedimiento se realiza hasta que el número de prototipos sea mayor a $\tau \times |C|$, donde

$|C|$ es el número de clases y $\tau \geq 1$ (líneas 1 - 40).

Al final del proceso evolutivo se regresa el conjunto de prototipos óptimo encontrado así como las etiquetas de las clases a las cuales pertenecen (línea 41).

3.3. GP^{UNB2}

El tercer método se llama GP^{UNB2}, este método solo genera prototipos para clases que necesitan aumentar su precisión, es decir, clases que aún no clasifican correctamente todas sus instancias. De lo contrario no es necesario generar más prototipos, porque ya no se puede mejorar su precisión.

El método GP^{UNB2} consiste en tres etapas, similares a GP^{BAL} y GP^{UNB}.

- *Conjunto de prototipos a generar:* se define el conjunto de clases para las cuales se generarán prototipos (líneas 4-13). Solo se genera prototipos para las clases que aún no clasifican correctamente todas sus muestras. Si no se encuentra un conjunto de prototipos a generar el algoritmo termina (línea 14).

- *Generación de prototipos:* la población de programas se inicializa aleatoriamente usando una *semilla* (líneas 2 y 15). Mientras la aptitud obtenida no supere la obtenida previamente y el número de *semillas* sea menor o igual a cierto número, se incrementa el valor de la *semilla* y se vuelve a ejecutar otra instancia del Algoritmo 3, usando la *semilla* actual para obtener otros prototipos, asegurando que la nueva población contenga al mejor individuo de la ejecución anterior (líneas 16-19).

- *Actualización de prototipos:* si se mejora la aptitud, se actualiza el conjunto de prototipos óptimo (líneas 20-25), de lo contrario el algoritmo termina (líneas 25-27). Este procedimiento se realiza hasta que el número de prototipos sea mayor a $\tau \times |C|$, donde $|C|$ es el número de clases y $\tau \geq 1$ (líneas 1 - 31).

Algoritmo 5 Algoritmo utilizado por GP^{UNB}

Entrada: δ : Parámetros de GP; τ : Máximo número de prototipos; N_s : Número de semillas;
p_f : Umbral de balanceo de clasificación; p_g : Umbral de discriminación;
$i \leftarrow 0$; $\mathcal{P}^* \leftarrow []$; $vector\_c^* \leftarrow []$;
$f^* \leftarrow -1.0$; $enc \leftarrow$ **cierto**;
$C \leftarrow$ Conjunto de clases;
Garantizar: $\tau >= 1$;

1: **mientras** $|vector\_c^*| < \tau \times |C|$ **and** enc **hacer**
2: $semilla \leftarrow 0$;
3: $vector\_c \leftarrow []$;
4: **si** $vector\_c^*$ esta vacío **entonces**
5: $vector\_c \leftarrow C$;
6: **si no**
7: $fit\_class \leftarrow$ Calcular porcentaje de clasificación por clase;
8: $max = fit\_class.max()$
9: $min = fit\_class.min()$
10: **si** $(max - min) \geq p_f$ **entonces**
11: **para todo** j en C **hacer**
12: **si** $fit\_class[j] \leq (min + p_g * (max - min))$ **entonces**
13: $vector\_c.append(j)$
14: **fin si**
15: **fin para**
16: **si no**
17: **para todo** j en C **hacer**
18: **si** $fit\_class[j] \neq 1.0$ **entonces**
19: $vector\_c.append(j)$
20: **fin si**
21: **fin para**
22: **fin si**
23: **fin si**
24: **si not** ($vector\_c$ esta vacío) **entonces**
25: $[\mathcal{P}\ f\ best\_ind] \leftarrow GP(\delta, semilla, \mathcal{P}^*, vector\_c^*, vector\_c)$;
26: **mientras** $f^* \geq f$ **and** $semilla \leq N_s$ **hacer**
27: $semilla \leftarrow semilla + 1$;
28: $[\mathcal{P}\ f\ best\_ind] \leftarrow GP(\delta, semilla, \mathcal{P}^*, best\_ind, vector\_c^*, vector\_c)$;
29: **fin mientras**
30: **si** $f > f^*$ **entonces**
31: $\mathcal{P}^* \leftarrow \mathcal{P}$;
32: $f^* \leftarrow f$;
33: $vector\_c^* \leftarrow concatenar(vector\_c^*, vector\_c)$;
34: **si no**
35: $enc \leftarrow$ **falso**;
36: **fin si**
37: **si no**
38: $enc \leftarrow$ **falso**;
39: **fin si**
40: **fin mientras**
41: **regresa:** $[\mathcal{P}^*\ vector\_c^*]$

Algoritmo 6 Algoritmo utilizado por GP^{UNB2}

Entrada: δ : Parámetros de GP; τ : Máximo número de prototipos; N_s : Número de semillas;
$i \leftarrow 0$; $\mathcal{P}^* \leftarrow []$; $vector\_c^* \leftarrow []$;
$f^* \leftarrow -1.0$; $enc \leftarrow$ **cierto**;
$C \leftarrow$ Conjunto de clases;
Garantizar: $\tau >= 1$;

1: **mientras** $|vector\_c^*| < \tau \times |C|$ **and** enc **hacer**
2: $semilla \leftarrow 0$;
3: $vector\_c \leftarrow []$;
4: **si** $vector\_c^*$ esta vacío **entonces**
5: $vector\_c \leftarrow C$;
6: **si no**
7: $fit\_class \leftarrow$ Calcular porcentaje de clasificación por clase;
8: **para todo** j en C **hacer**
9: **si** $fit\_class[j] \neq 1.0$ **entonces**
10: $vector\_c.append(j)$
11: **fin si**
12: **fin para**
13: **fin si**
14: **si not** ($vector\_c$ esta vacío) **entonces**
15: $[\mathcal{P}\ f\ best\_ind] \leftarrow GP(\delta, semilla, \mathcal{P}^*, vector\_c^*, vector\_c)$;
16: **mientras** $f^* \geq f$ **and** $semilla \leq N_s$ **hacer**
17: $semilla \leftarrow semilla + 1$;
18: $[\mathcal{P}\ f\ best\_ind] \leftarrow GP(\delta, semilla, \mathcal{P}^*, best\_ind, vector\_c^*, vector\_c)$;
19: **fin mientras**
20: **si** $f > f^*$ **entonces**
21: $\mathcal{P}^* \leftarrow \mathcal{P}$;
22: $f^* \leftarrow f$;
23: $vector\_c^* \leftarrow concatenar(vector\_c^*, vector\_c)$;
24: $i \leftarrow i + 1$;
25: **si no**
26: $enc \leftarrow$ **falso**;
27: **fin si**
28: **si no**
29: $enc \leftarrow$ **falso**;
30: **fin si**
31: **fin mientras**
32: **regresa:** $[\mathcal{P}^*\ vector\_c^*]$

Al final del proceso evolutivo se regresa el conjunto de prototipos óptimo encontrado así como las etiquetas de las clases a las cuales pertenecen (línea 32).

3.4. Conclusiones

Con la implementación de los métodos presentados en este capítulo, se logró generar prototipos de una forma iterativa. Los tres métodos presentados utilizan un procedimiento similar, en lo que difieren es la forma en como se calcula el conjunto de prototipos que serán generados con GP. GP^{BAL} genera prototipos de manera balanceada para todas las clases. GP^{UNB} genera prototipos tratando de equilibrar el rendimiento de clasificación por clase. GP^{UNB2} sólo genera prototipos para las clases que contienen casos mal clasificados.

El rendimiento de los algoritmos propuestos en términos de precisión y reducción; así como la comparación con técnicas del estado del arte, se exponen en el siguiente capítulo.

Capítulo 4

Resultados

Hasta el momento se han descrito los procedimientos utilizados para crear sistemas de clasificación de patrones mediante la generación de prototipos. En este capítulo se reportan los resultados experimentales obtenidos por GP^{BAL}, GP^{UNB} y GP^{UNB2} usando un conjunto de bases de datos de referencia para la evaluación de técnicas de generación de prototipos.

4.1. Conjunto de bases de datos de referencia utilizado

El conjunto de bases referencia utilizada para evaluar y hacer comparaciones, consiste en 59 problemas de clasificación con múltiples clases, tomados de los repositorios de UCI [Bache and Lichman, 2013] y KEEL [Alcalá-Fdez et al., 2011]. Esta base de datos fue utilizada en los estudio experimentales de Triguero et al. [Triguero et al., 2012], Escalante et al. [Escalante et al., 2013] y Rosales-Pérez et al. [Rosales-Pérez et al., 2014].

De acuerdo a Triguero, el conjunto de bases referencia se divide en pequeñas y grandes bases de datos. Los problemas de las pequeñas bases de datos tienen menos de 2000 instancias y los problemas de las grandes bases de datos tiene más de 2000 instancias. El número de clases está entre 2 y 28, el número de instancias es entre 101 y 19020, y el número de características oscila entre 2 y 90 (atributos numéricos y nominales). 40 son

Tabla 4.1: Descripción del conjunto de las bases de datos utilizadas para el estudio experimental. Para cada base de datos se muestra: el número de instancias, el número de atributos y el número de clases.

Base de datos	**Ins.**	**Atr.**	**Cl.**	**Base de datos**	**Ins.**	**Atr.**	**Cl.**
Abalone	4174	8	28	Marketing	8993	13	9
Appendicitis	106	7	2	Monks	432	6	2
Australian	690	14	2	Movement_libras	360	90	15
Automobile	205	25	6	Newthyroid	215	5	3
Balance	625	4	3	Nursery	12960	8	5
Banana	5300	2	2	Pageblocks	5472	10	5
Bands	539	19	2	Penbased	10992	16	10
Breast-Cancer	286	9	2	Phoneme	5404	5	2
Bupa	345	6	2	Pima	768	8	2
Car	1728	6	4	Ring	7400	20	2
Chess	3196	36	2	Saheart	462	9	2
Cleveland	297	13	5	Satimage	6435	36	7
Coil2000	9822	85	2	Segment	2310	19	7
Contraceptive	1473	9	3	Sonar	208	60	2
Crx	690	15	2	Spambase	4597	57	2
Dermatology	366	33	6	Spectheart	267	44	2
Ecoli	336	7	8	Splice	3190	60	3
Flare-Solar	1066	11	2	Tae	151	5	3
German	1000	20	2	Texture	5500	40	11
Glass	214	9	7	Thyroid	7200	21	3
Haberman	306	3	2	Tic-Tac-Toe	958	9	2
Hayes-Roth	160	4	3	Titanic	2201	3	2
Heart	270	13	2	Twonorm	7400	20	2
Hepatitis	155	19	2	Vehicle	846	18	4
Housevotes	435	16	2	Vowel	990	13	11
Iris	150	4	3	Wine	178	13	3
Led7digit	500	7	10	Wisconsin	699	9	2
Lymphography	148	18	4	Yeast	1484	8	10
Magic	19020	10	2	Zoo	101	16	7
Mammographic	961	5	2				

bases de datos pequeñas y 19 grandes. La Tabla 4.1 muestra una breve descripción de las bases de datos de clasificación utilizadas. Este conjunto de bases nos permite evaluar el desempeño de nuestros métodos y comparar su rendimiento con técnicas del estado del arte en la generación de prototipos.

4.2. Experimentos realizados

Las bases de datos consideradas fueron particionadas por Triguero et al. usando validación cruzada de diez iteraciones ($10 - fcv$, ten fold cross-validation), consiste en dividir los datos de muestra en 10 subconjuntos. Uno de los subconjuntos se utiliza como datos de validación y el resto (9) como datos de entrenamiento. Este proceso se repite por 10 iteraciones, con cada uno de los posibles subconjunto de los datos de prueba. Finalmente la media aritmética de los resultados obtenidos a partir de las 10 particiones es reportada.

Los dos principales objetivos que se quieren medir en los métodos propuestos es la precisión y la reducción del conjunto de entrenamiento, como se muestra en el diagrama de bloques de la Figura 4.1. En este diagrama se muestra el proceso utilizado para obtener la precisión tanto en el conjunto de entrenamiento $\mathcal{T}$ como en el conjunto de validación $\mathcal{V}$, así como la reducción obtenida del conjunto $\mathcal{T}$.

4.3. Visualización de prototipos

Esta sección ilustra el conjunto de prototipos generados por GP^{BAL}, GP^{UNB} y GP^{UNB2}, esto es con el fin de dar una idea sobre el tipo de soluciones encontradas por GP. Para este fin, se utiliza el conjunto de datos Banana, que contiene $5,300$ instancias en el conjunto completo. Se trata de un conjunto de datos artificial de dos características y dos clases. La Figura 4.2 muestra los resultados obtenidos para diferentes valores de τ, la figura ayuda a visualizar y entender la forma en que trabajan los métodos propuestos en este trabajo.

La Figura 4.2.a ilustra como se genera un prototipo por clase usando GP (es decir, $\tau = 1$), es importante resaltar que todos los métodos generan los mismos prototipos con $\tau = 1$. Las Figuras 4.2.b a 4.2.e muestran cómo GP^{BAL} genera prototipos usando dife-

rentes umbrales $\tau = 3$, $\tau = 5$, $\tau = 10$ y $\tau = 50$, respectivamente. Es interesante notar que GP^{UNB2} genera exactamente el mismo conjunto de prototipos que GP^{BAL} para este conjunto de datos, lo que significa que ninguna clase clasifica a todas sus muestras correctamente lo que provoca que ambos métodos generen el mismo conjunto de prototipos. La Figura 4.2.f muestra el conjunto de prototipos obtenidos usando GP^{UNB} con $\tau = 50$. A pesar de usar un umbral grande ($\tau = 50$), los métodos paran cuando ya no encuentran un conjunto de prototipos que mejoren la clasificación, para la base de datos Banana: GP^{BAL} y GP^{UNB2} encontraron 46 prototipos, GP^{UNB} encontró 40 prototipos. Sin embargo, el máximo número de prototipos que se podían generar era de 100 ($\tau \times$ número de clases) prototipos.

Comparando los prototipos generados por GP^{BAL} y GP^{UNB} con $\tau = 50$, se observa que GP^{BAL} genera prototipos que no ayudan a la clasificación, es decir, estos prototipos están lejos de todos los puntos en el conjunto de datos; estos se encuentran en las esquinas inferior izquierda y superior derecha de la Figura 4.2.e. Este comportamiento no está presente en GP^{UNB} (ver Figura 4.2.f), esta es la razón por la cual GP^{UNB} tiene una mejor reducción de conjunto de entrenamiento que GP^{BAL}; sin embargo, las limitaciones impuestas tuvieron un impacto en la precisión como se muestra en las secciones 4.4 y 4.5.

Para el conjunto de datos banana GP^{BAL} y GP^{UNB2} tienen una precisión de $91.25\,\% \pm 0.21$, y GP^{UNB2} tiene una precisión de $91.03\,\% \pm 0.48$ en el conjunto de entrenamiento, mientras que en el conjunto de validación GP^{BAL} y GP^{UNB2} tienen una precisión de $89.70\,\% \pm 1.22$, y GP^{UNB2} tiene una precisión de $89.77\,\% \pm 1.42$.

4.4. Resultados obtenidos en bases de datos pequeñas

Esta sección presenta los resultados de precisión y reducción de $\mathcal{T}$ en las bases de datos pequeñas contenidas en el conjunto de bases de datos utilizado como referencia.

La Tabla 4.2 muestra el promedio y la desviación estándar obtenida en término

de precisión en $\mathcal{V}$ y reducción de $\mathcal{T}$ en las 40 bases de datos de tamaño pequeño utilizadas como referencia; para valores de $\tau = \{1, 10, 20, 30, 40, 50\}$. Los resultados en negrita son en los cuales se obtienen mejor precisión y reducción. Cómo se puede observar, la mejor precisión obtenida fue para GP^{BAL}, GP^{UNB} y GP^{UNB2}, respectivamente. Sin embargo, en términos de reducción GP^{UNB} fue el mejor.

Tabla 4.2: Rendimiento promedio y la desviación estándar, así como el porcentaje de reducción promedio obtenida por los métodos propuestos para pequeñas bases de datos utilizando diferentes umbrales τ.

	Precisión del conjunto de validación $\mathcal{V}$			**Reducción del conjunto de entrenamiento $\mathcal{T}$**		
τ	GP^{BAL}	GP^{UNB}	GP^{UNB2}	GP^{BAL}	GP^{UNB}	GP^{UNB2}
$\tau = 50$	**74.38 % ± 13.86**	73.89 % ± 14.17	74.33 % ± 13.70	88.61 %	**92.59** %	90.12 %
$\tau = 40$	74.38 % ± 13.86	73.89 % ± 14.17	74.33 % ± 13.70	88.61 %	92.59 %	90.12 %
$\tau = 30$	74.37 % ± 13.86	73.89 % ± 14.16	74.30 % ± 13.73	88.66 %	92.60 %	90.20 %
$\tau = 20$	74.30 % ± 13.90	73.87 % ± 14.17	74.24 % ± 13.76	89.00 %	92.71 %	90.60 %
$\tau = 10$	74.05 % ± 13.98	73.75 % ± 14.27	74.07 % ± 13.85	90.92 %	93.34 %	91.80 %
$\tau = 1$	70.35 % ± 16.25	70.35 % ± 16.25	70.35 % ± 16.25	98.58 %	98.58 %	98.58 %

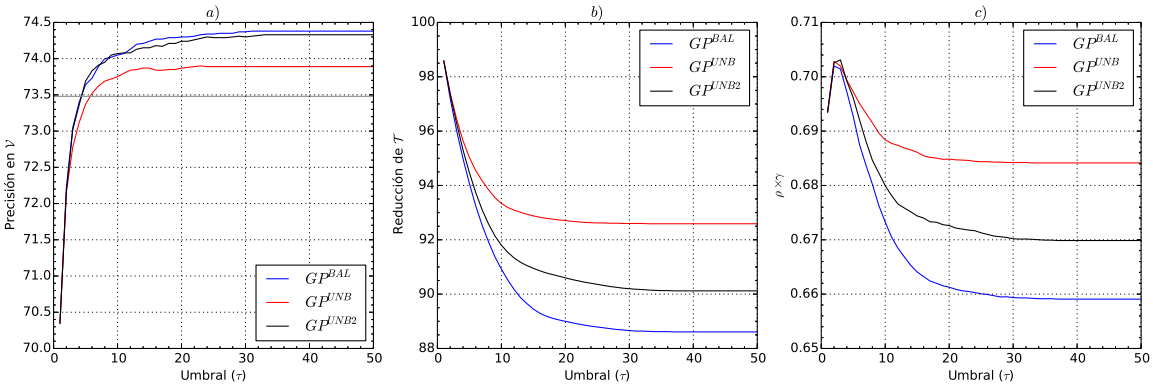

Figura 4.3: a) Precisión en $\mathcal{V}$, b) reducción de $\mathcal{T}$ y c) precisión en $\mathcal{V}$ (ρ) por el porcentaje de reduccción (γ); usando los métodos propuestos con diferentes umbrales τ en bases de datos pequeñas. La línea gris en 73.48 en a) indica la precisión obtenida por el clasificador NN.

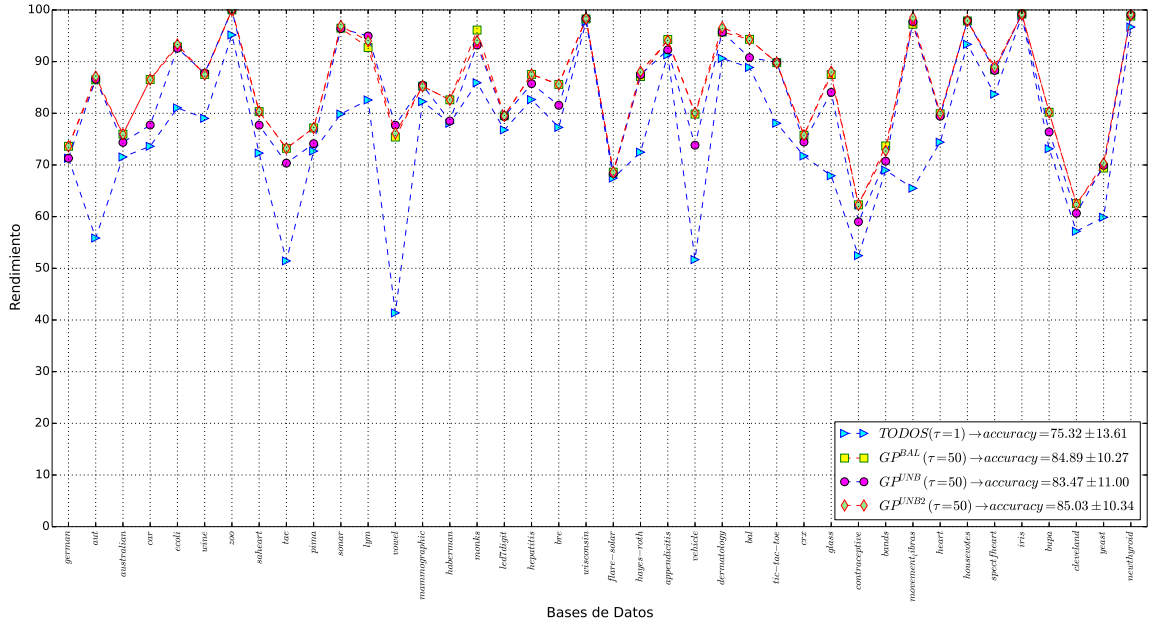

Figura 4.4: Rendimiento obtenido en términos de precisión por GP^{BAL}, GP^{UNB} y GP^{UNB2} en el conjunto de entrenamiento en bases de datos pequeñas

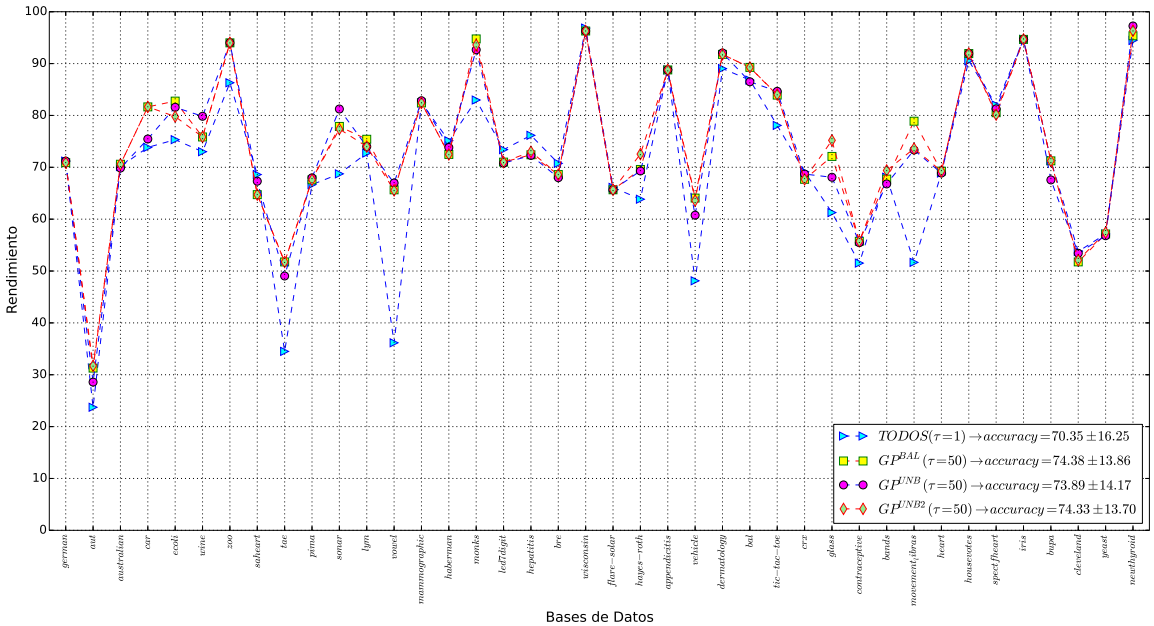

Figura 4.5: Rendimiento obtenido en términos de precisión por obtenido por GP^{BAL}, GP^{UNB} y GP^{UNB2} en el conjunto de validación en bases de datos pequeñas

La información presentada en Tabla 4.2 es complementada con la Figura 4.3. Figura 4.3.a muestra la precisión en $\mathcal{V}$ (ρ); 4.3.b muestra la reducción de $\mathcal{T}$ (γ); y 4.3.c muestra la precisión por la reducción ($\rho \times \gamma$)[1]. Estas figuras se obtienen al variar τ de 1 a 50. Cómo se puede ver para pequeños valores de τ, la precisión mejora considerablemente y la compensación entre precisión y reducción es muy buena. Para valores superiores de $\tau \geq 35$, los métodos propuestos no pueden encontrar más conjuntos de prototipos que ayuden a la clasificación. La línea gris en 4.3.a representa el rendimiento obtenido por un clasificador 1NN, cabe destacar que los métodos propuestos superan significativamente el rendimiento de 1NN en estas bases de datos y se logra una reducción considerable.

En la Figura 4.4, se puede ver el rendimiento obtenido en el conjunto de entrenamiento de los problemas de conjuntos pequeños, mientras que en la Figura 4.5 se muestra el rendimiento obtenido en el conjunto de validación usando los métodos propuestos. Se puede observar que la precisión en el conjunto de validación mejora significativamente en casi todos los problemas al aumentar el valor de τ, sin embargo otros problemas sufren sobre-entrenamiento debido a las pocas muestras que se tienen.

4.5. Resultados obtenidos en bases de datos grandes

Esta sección presenta los resultados de precisión y reducción de $\mathcal{T}$ en bases de datos grandes contenidas en el conjunto de bases de datos utilizado como referencia.

La Tabla 4.3 presenta los resultados obtenidos por los métodos propuestos en las 19 bases de datos grandes para $\tau = \{1, 10, 20, 30, 40, 50\}$. Se presenta la precisión y desviación estándar en validación y reducción del conjunto de entrenamiento, donde se puede observar que al incrementar el valor de τ se obtienen mejor rendimiento. Por otra parte, la reducción en el conjunto de entrenamiento $\mathcal{T}$ disminuye, este proceso se puede ver gráficamente en la Figura 4.6. Los resultados en negrita son en los cuales se obtuvieron mejor

[1]La reducción por la precisión se utiliza como una medida que indica la compensación que hay entre ambas [Triguero et al., 2012]. Es decir, que tanto un algoritmo PG reduce $\mathcal{T}$ sin disminuir su precisión.

precisión y reducción. Como se puede ver: GP^{UNB2} obtiene la mejor precisión, GP^{UNB} la peor precisión pero la mejor reducción.

Tabla 4.3: Rendimiento promedio y desviación estándar, así como el porcentaje de reducción promedio obtenida por los métodos propuestos para bases de datos grandes utilizando diferentes umbrales τ

	Precisión del conjunto de validación $\mathcal{V}$			**Reducción del conjunto de entrenamiento $\mathcal{T}$**		
τ	GP^{BAL}	GP^{UNB}	GP^{UNB2}	GP^{BAL}	GP^{UNB}	GP^{UNB2}
$\tau = 50$	82.64 % ± 19.90	81.67 % ± 19.80	**82.67 % ± 20.04**	96.39 %	**97.87** %	96.33 %
$\tau = 40$	82.61 % ± 19.90	81.63 % ± 19.79	82.66 % ± 19.96	96.68 %	97.94 %	96.69 %
$\tau = 30$	82.47 % ± 19.97	81.57 % ± 19.78	82.53 % ± 20.00	97.18 %	98.03 %	97.21 %
$\tau = 20$	82.27 % ± 19.92	81.45 % ± 19.81	82.37 % ± 19.90	97.94 %	98.20 %	97.93 %
$\tau = 10$	81.67 % ± 19.80	81.07 % ± 19.78	81.82 % ± 19.76	98.86 %	98.96 %	98.86 %
$\tau = 1$	75.09 % ± 19.02	75.09 % ± 19.02	75.09 % ± 19.02	99.88 %	99.88 %	99.88 %

La Figura 4.6 muestra la precisión, reducción, y precisión por reducción ($\rho \times \gamma$) complementaria a la Tabla 4.3. Para obtener estos gráficos se varia τ de 1 a 50. La línea gris en la Figura 4.6.a representa la precisión obtenida por un clasificador 1NN; los métodos propuestos superan el rendimiento de 1NN en bases de datos grandes con valores de $\tau \geq 5$. El comportamiento es similar al obtenido en bases de datos pequeñas, la diferencia es que aún con valores grandes de τ el rendimiento continúa mejorando. Estos resultados eran de esperarse, ya que al tener muchas muestras es más complicada la tarea de clasificación lo cual es reflejado en más prototipos por clase, sin embargo la compensación entre precisión y reducción es mucho mejor comparado con los resultados obtenidos en bases de datos pequeñas.

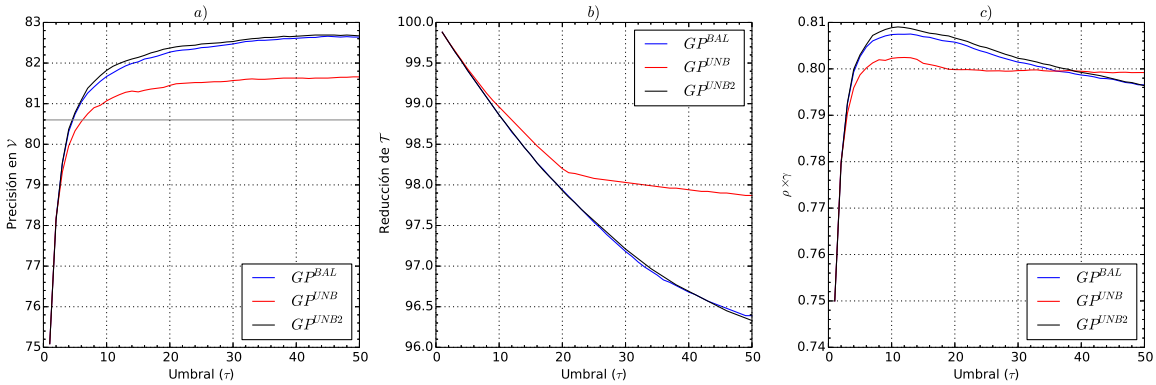

Figura 4.6: a) Precisión en $\mathcal{V}$, b) reducción de $\mathcal{T}$ y c) precisión en $\mathcal{V}$ (ρ) por el porcentaje de reduccción (γ); usando los métodos propuestos con diferentes umbrales τ en bases de datos grandes. La línea gris en 80.60 en a) indica la precisión obtenida por el clasificador NN.

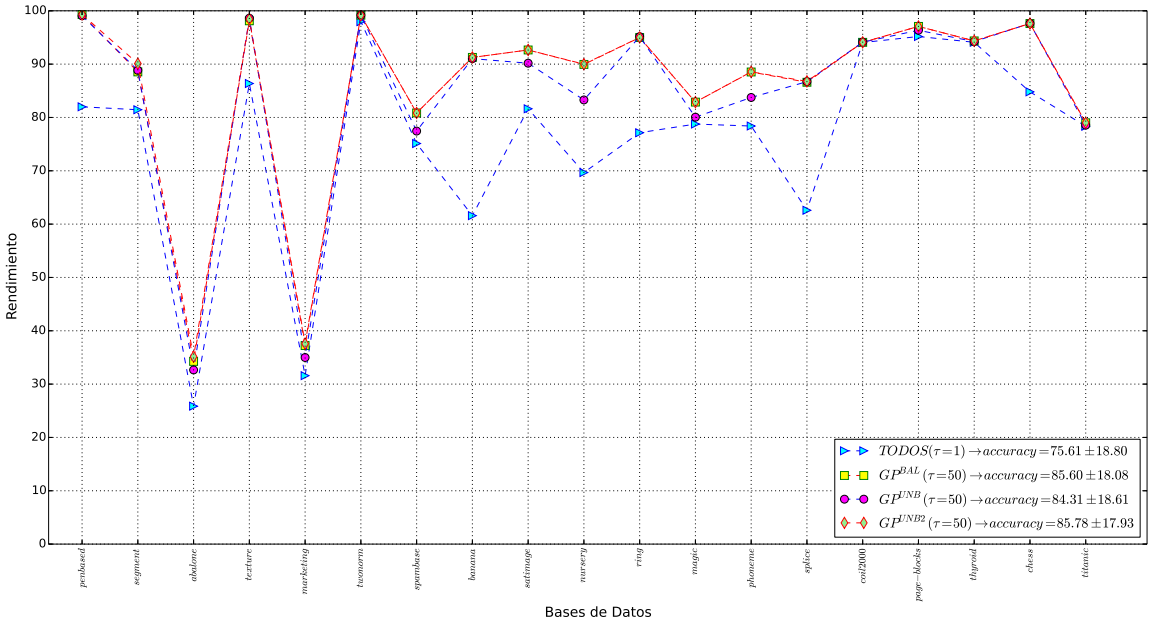

Figura 4.7: Rendimiento obtenido en términos de precisión por GP^{BAL}, GP^{UNB} y GP^{UNB2} en el conjunto de entrenamiento en bases de datos grandes

La Figura 4.7 muestra el rendimiento obtenido en el conjunto de entrenamiento, y la Figura 4.8 el rendimiento obtenido en el conjunto de validación de las bases de datos grandes usando los métodos propuestos. Se logra mejorar significativamente el rendimiento en validación al incrementar el número de prototipos. Notar que tanto en validación como

entrenamiento se obtiene un rendimiento similar, lo que significa que los métodos propuestos no sufren de sobre-entrenamiento en bases de datos grandes.

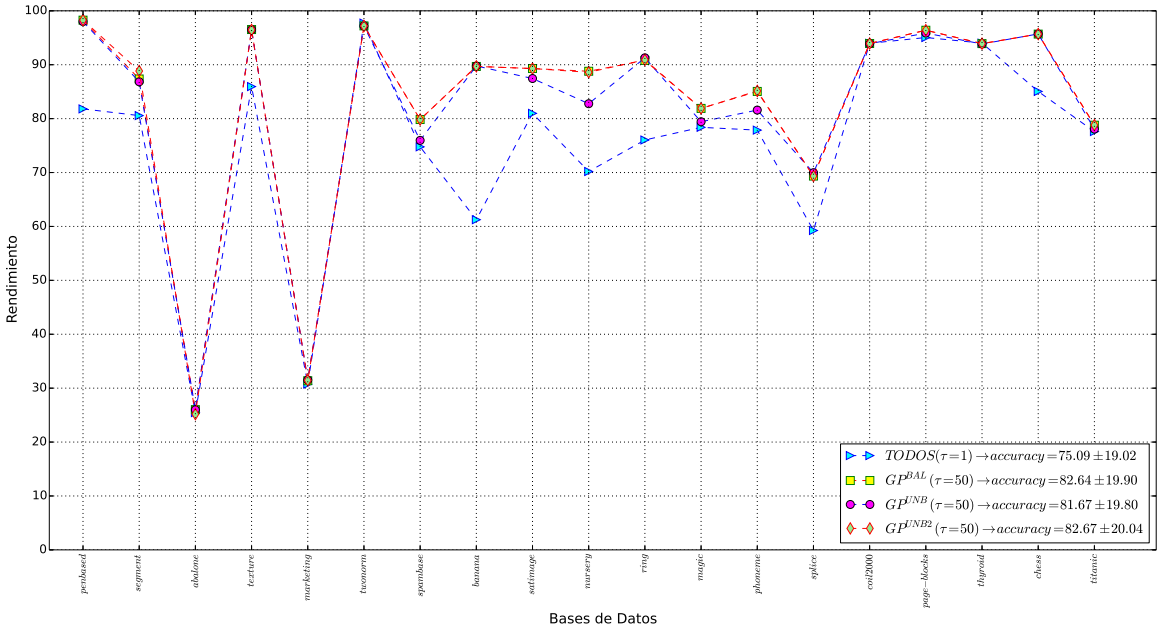

Figura 4.8: Rendimiento obtenido en términos de precisión por GP^{BAL}, GP^{UNB} y GP^{UNB2} en el conjunto de validación en bases de datos grandes

La Tabla 4.4 muestra los resultados obtenidos en el conjunto de bases de datos utilizadas como referencia para valores de $\tau = 1, 10, 20, 30, 40, 50$. Como se puede observar GP^{BAL} y GP^{UNB2} estadísticamente tienen la misma precisión con una diferencia de 0.01 %, pero GP^{UNB2} tiene una reducción mucho mejor que GP^{BAL}. En consecuencia, GP^{UNB2} tiene la mejor compensación entre precisión y reducción del conjunto $\mathcal{T}$. La Figura 4.9.c corrobora esta compensación representada por $\rho \times \gamma$, donde se puede ver que GP^{UNB2} tiene el pico más grande, es decir, tiene el mejor balance entre precisión y reducción.

Tabla 4.4: Rendimiento promedio y desviación estándar, así como el porcentaje de reducción promedio obtenida por los métodos propuestos para todas las bases de datos de referencia usando diferentes umbrales τ

	Precisión del conjunto de validación $\mathcal{V}$			**Reducción del conjunto de entrenamiento $\mathcal{T}$**		
τ	GP^{BAL}	GP^{UNB}	GP^{UNB2}	GP^{BAL}	GP^{UNB}	GP^{UNB2}
$\tau = 50$	**78.51 % ± 16.88**	77.78 % ± 16.98	78.50 % ± 16.87	92.50 %	**95.23 %**	93.22 %
$\tau = 40$	78.49 % ± 16.88	77.76 % ± 16.98	78.50 % ± 16.83	92.65 %	95.26 %	93.40 %
$\tau = 30$	78.42 % ± 16.91	77.73 % ± 16.97	78.42 % ± 16.86	92.92 %	95.31 %	93.71 %
$\tau = 20$	78.28 % ± 16.91	77.66 % ± 16.99	78.31 % ± 16.83	93.47 %	95.45 %	94.27 %
$\tau = 10$	77.86 % ± 16.89	77.41 % ± 17.02	77.94 % ± 16.81	94.89 %	96.15 %	95.33 %
$\tau = 1$	72.72 % ± 17.63	72.72 % ± 17.63	72.72 % ± 17.63	99.23 %	99.23 %	99.23 %

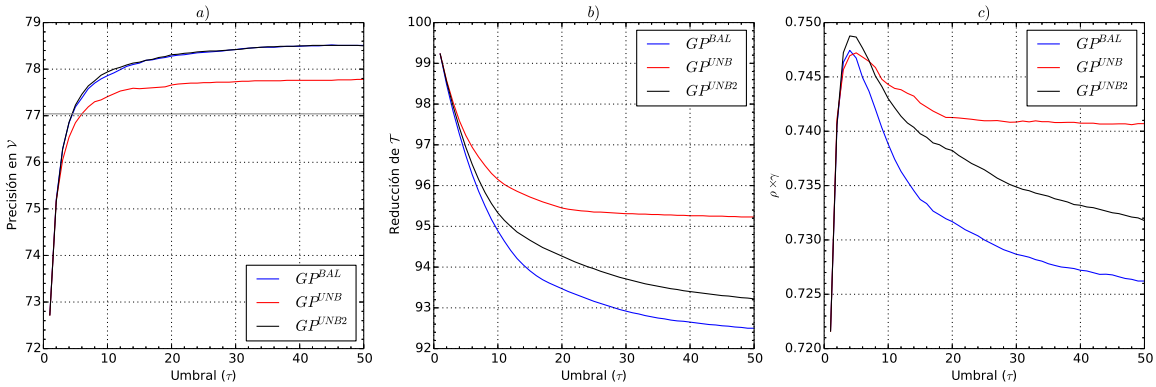

Figura 4.9: a) Precisión en $\mathcal{V}$, b) reducción de $\mathcal{T}$ y c) precisión en $\mathcal{V}$ (ρ) por el porcentaje de reduccción (γ); usando los métodos propuestos con diferentes umbrales τ en todas las bases de datos utilizadas. La línea gris en 77.04 en a) indica la precisión obtenida por el clasificador NN.

De las Figuras 4.5 y 4.8, se puede observar que 30 de los 59 problemas no hay una diferencia significativa en precisión de los diferentes sistemas GP propuestos. Por otra parte, 22 de los 59 problemas no se supero significativamente con $\tau = 50$ la precisión obtenida con $\tau = 1$, es decir, el 37 % de los problemas se podría probar con $\tau = 1$ y se obtendría la mejor precisión.

El último análisis realizado a los sistemas propuestos tiene como objetivo conocer si el uso de solo atributos numéricos, atributos nominales o combinación de estos dos; tiene un impacto en la precisión de los métodos. La Tabla 4.5 muestra la precisión media

de la agrupación de los conjuntos de datos acorde al tipo de atributos utilizados en cada problema. Los problemas se agrupan de acuerdo al tipo de características: ya sea valores numéricos, nominales o mixtos. De las 59 bases de datos consideradas, 41 tienen solo atributos numéricos, 10 tienen solo atributos nominales (breast, car, chess, flare-solar, housevotes, mammographic, nursery, splice, tic-tac-toe, zoo) y 8 tienen atributos numéricos-nominales (abalone, australian, automobile, crx, dermatology, german, lymphography, saheart). El mejor rendimiento de los métodos propuestos se obtuvo en bases da datos con solo un tipo de atributo ya sea numérico o nominal. Esto se puede deber a que se usa distancia euclidiana como medida de similitud, porque es sensible a las unidades de medida de las variables y no se recomienda usarlo cuando las variables utilizan diferentes unidades de medida, como es el caso de conjuntos de datos con atributos mixtos.

Tabla 4.5: Precisión media y desviación estándar en los conjuntos de datos con similar tipo de características.

Método	**Valores númericos**	**Valores Nominales**	**Valores mixtos**
GP^{BAL}	$78.66\,\% \pm 15.13$	$82.20\,\% \pm 10.42$	$62.30\,\% \pm 20.91$
GP^{UNB}	$78.01\,\% \pm 15.27$	$81.10\,\% \pm 10.40$	$62.21\,\% \pm 21.45$
GP^{UNB2}	$78.67\,\% \pm 15.05$	$82.19\,\% \pm 10.41$	$62.07\,\% \pm 20.90$

4.6. Comparación de los resultados obtenidos con técnicas del estado del arte

Esta sección compara los resultados de los métodos propuestos con técnicas del estado del arte como EMOPG [Rosales-Pérez et al., 2014], GPPG [Escalante et al., 2013] y otros [Triguero et al., 2012]. La Tabla 4.6 muestra la media y la desviación estándar de los resultados obtenidos por GP^{UNB2} y por los estudios de referencia, se utiliza GP^{UNB2} porque tiene mejor compensación entre precisión y reducción. Esta tabla muestra los resultados obtenidos en términos de precisión de clasificación en $\mathcal{V}$ y reducción del conjunto $\mathcal{T}$, al considerar: conjuntos de datos pequeños, conjuntos de datos grandes y todos los conjuntos de datos (media de los resultados de los conjuntos de datos grandes y pequeños).

Tabla 4.6: Media y desviación estandar del rendimiento y porcentaje de reducción obtenida por GP^{UNB2} para bases de datos pequeñas, grandes, y todas. Para la comparación se muestra el rendimiento obtenido por EMOPG, GPPG, GENN, PSCSA y 1NN. Los mejores resultados se muestran en negrita.

	Precisión en $\mathcal{V}$			**Reducción de $\mathcal{T}$**		
Método	**Todas**	**Pequeñas**	**Grandes**	**Todas**	**Pequeñas**	**Grandes**
GP^{UNB2}	**78.50 % ± 16.87**	74.33 % ± 13.70	**82.67 % ± 20.04**	93.22 %	90.12 %	96.33 %
EMOPG	76.53 % ± 17.68	71.73 % ± 15.39	81.34 % ± 19.97	98.82 %	98.11 %	99.53 %
GPPG	76.68 % ± 4.01	72.13 % ± 6.73	81.23 % ± 1.41	86.93 %	76.69 %	97.17 %
GENN	78.48 % ± 18.57	**75.64 % ± 15.45**	81.33 % ± 21.70	17.19 %	18.62 %	15.76 %
PSCSA	66.94 % ± 20.39	66.82 % ± 18.74	67.07 % ± 22.05	**99.23 %**	**98.58 %**	**99.88 %**
1NN	77.04 % ± 19.44	73.48 % ± 16.64	80.60 % ± 22.24	0 %	0 %	0 %

De la Tabla 4.6, se puede observar que solo GP^{UNB2} y GENN superan al clasificador 1NN en bases de datos pequeñas y en todas las consideradas. En bases de datos grandes y en todas GP^{UNB2} tiene la mejor precisión. Por otra parte, en términos de porcentaje de reducción obtenidos por cada método, se puede observar que PSCSA obtiene la mejor tasa de reducción de conjunto $\mathcal{T}$ y EMOPG es el segundo mejor en todos los casos. GP^{UNB2} es el tercer mejor método en términos de reducción tanto en pequeños conjuntos de datos como en todos, y el cuarto en conjuntos de datos grandes. Sin embargo, GP^{UNB2} tiene tasas de reducción mayores al 90 % en todos los casos. GENN es el peor método en términos de reducción, 1NN no se considera porque no es un método PG.

La Figura 4.10 muestra el comportamiento de 25 métodos evolutivos y no evolutivos para PG (que se consideran en el estudio comparativo realizado en [Triguero et al., 2012]), EMOPG introducido en [Rosales-Pérez et al., 2014] y GPPG [Escalante et al., 2013] con respecto a la reducción de $\mathcal{T}$ y precisión en $\mathcal{V}$ sobre bases de datos pequeñas. Se puede observar que GP^{UNB2} sólo fue superado por tres métodos en términos de exactitud siendo GENN el mejor. Los tres métodos propuestos exceden la precisión obtenida por un clasificador 1NN y su reducción es superior al 88 %, los cuales son los objetivos principales en métodos de PG[2].

La Figura 4.11 muestra el comportamiento de 20 métodos evolutivos y no evoluti-

[2]Los métodos PG tienen como objetivo obtener instancias representativas del conjunto de entrenamiento que son más pequeñas que la original, tal que la precisión no se ve afectada [Wilson and Martinez, 2000]

vos para PG (que se consideran en el estudio comparativo realizado en [Triguero et al., 2012]), EMOPG introducido en [Rosales-Pérez et al., 2014] y GPPG [Escalante et al., 2013] con respecto a la reducción de $\mathcal{T}$ y precisión en $\mathcal{V}$ sobre bases de datos grandes. Se puede observar que los métodos propuestos son los mejores en términos de precisión (siendo GP^{UNB2} el mejor de los tres).

4.7. Conclusiones

Un experimental extenso estudio se llevó a cabo con el fin de evaluar el rendimiento de los enfoques propuestos. De los resultados obtenidos se concluye lo siguiente: GP^{UNB2} tiene muy buena precisión similar a GENN (el mejor método en términos de precisión reportado por [Triguero et al., 2012]), pero GP^{UNB2} tiene un rendimiento en reducción considerablemente mejor que GENN. PSCSA tiene la mejor reducción, pero los peores resultados en la precisión entre los métodos considerados. Vale la pena señalar que los métodos propuestos con $\tau = 1$ tienen una reducción del conjunto $\mathcal{T}$ equivalente a PSCSA; sin embargo, estos obtienen una mayor precisión en las tres particiones de los conjuntos de datos. EMOPG tiene buena reducción y resultados competitivos en precisión, pero no supera a GP^{UNB2} y GENN en precisión. Por otra parte, GPPG tiene una reducción similar a GP^{UNB2} pero su precisión es menor.

En el siguiente capítulo se exponen las conclusiones del trabajo y se mencionan algunas mejoras que se pueden realizar al enfoque propuesto, así como trabajos futuros de investigación.

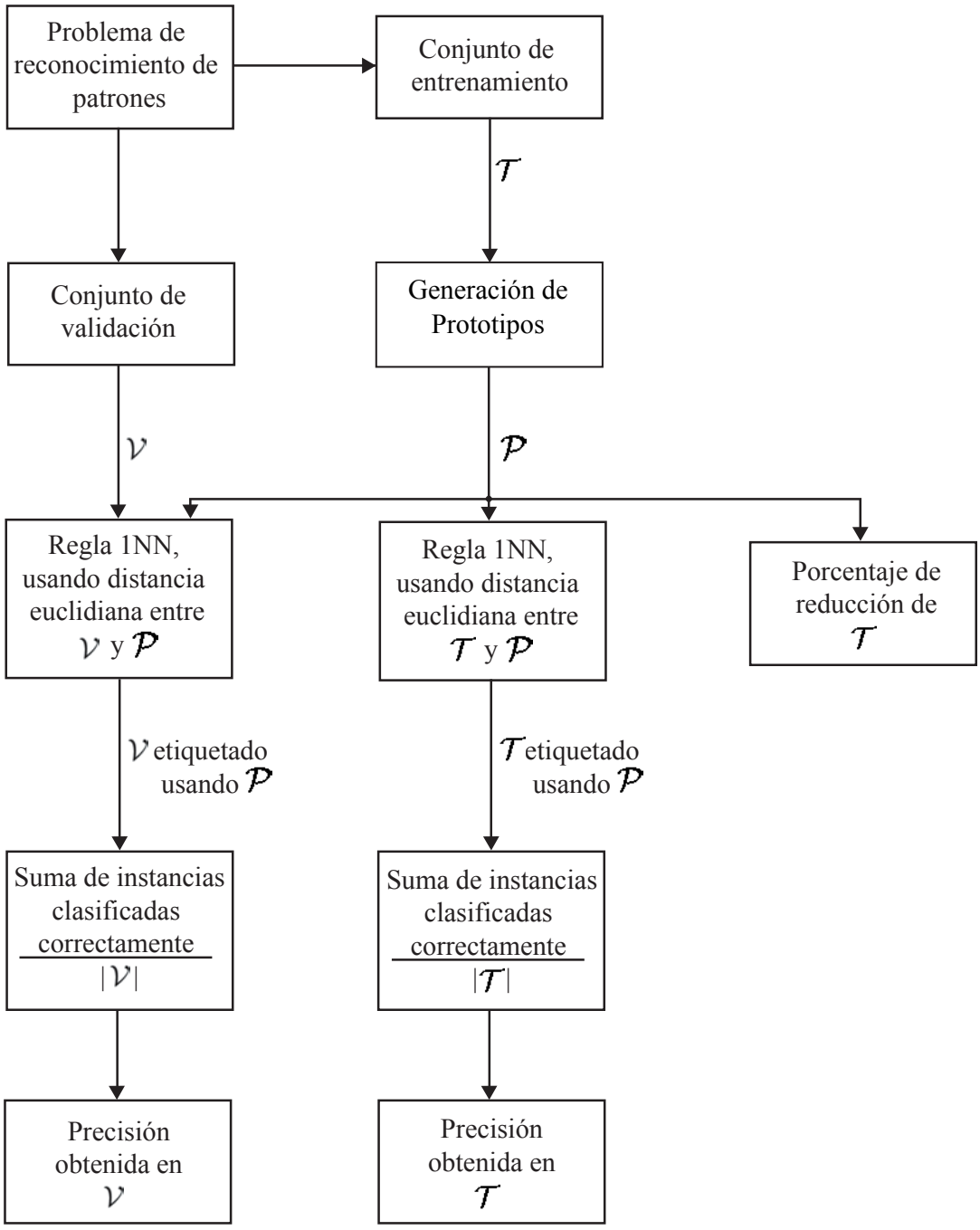

Figura 4.1: Diagrama de bloques del procedimiento utilizado para evaluar la precisión y la reducción de los PG.

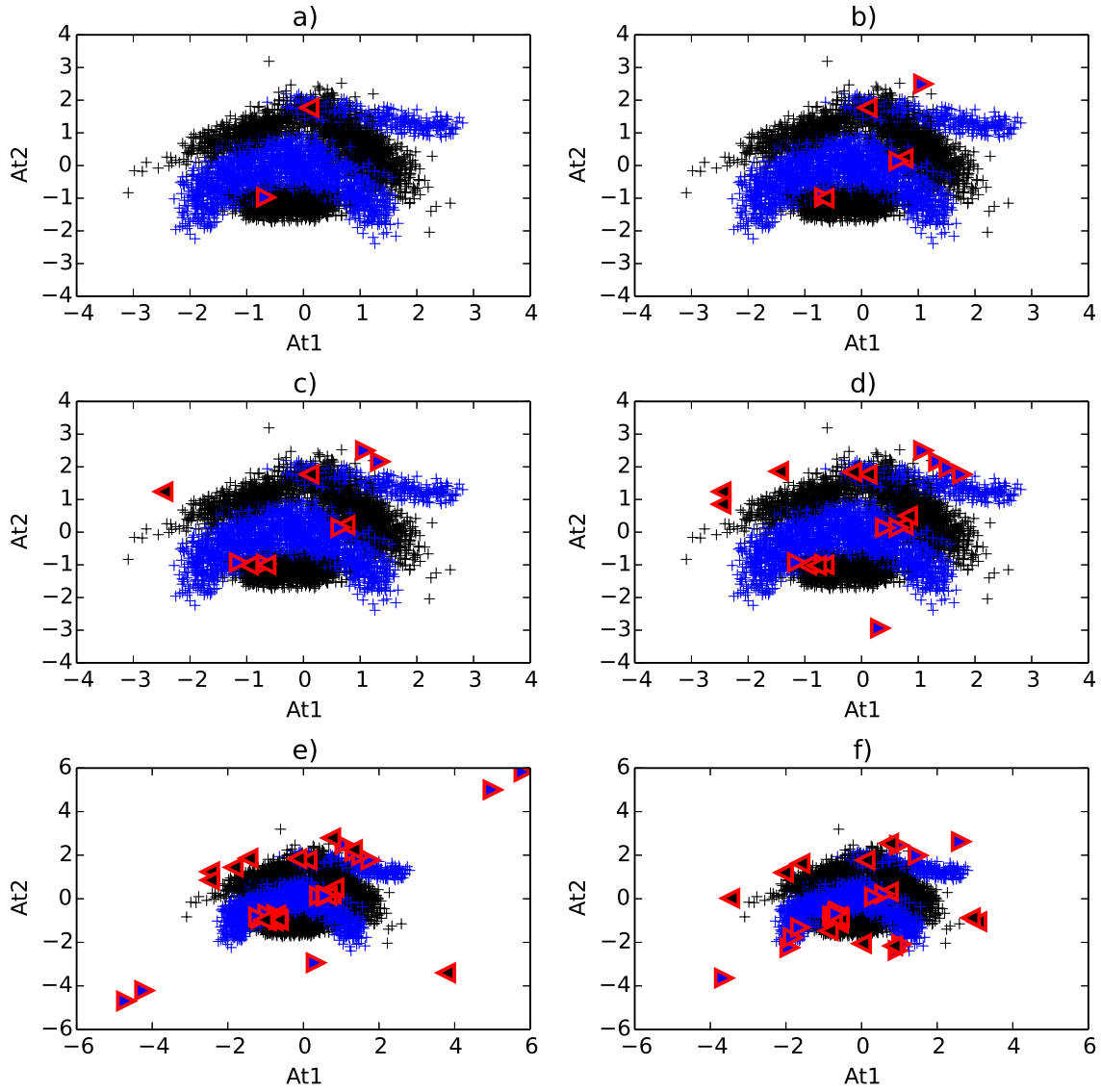

Figura 4.2: Prototipos generados para el conjunto de datos Banana. a) Prototipos generados con $\tau = 1$, notar que los tres métodos propuestos generan los mismos prototipos en este caso. b) Prototipos generados usando GP^{BAL} con $\tau = 3$. c) Prototipos generados usando GP^{BAL} con $\tau = 5$. d) Prototipos generados usando GP^{BAL} con $\tau = 10$. e) Prototipos generados usando GP^{BAL} con $\tau = 50$. f) Prototipos generados usando GP^{UNB} con $\tau = 50$.

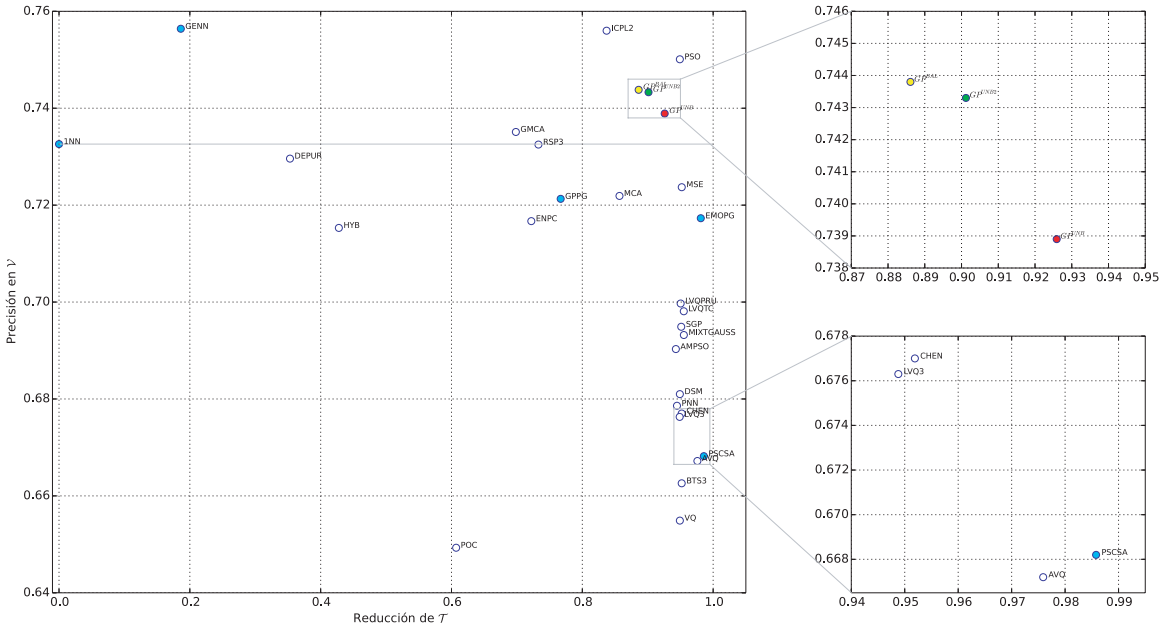

Figura 4.10: Precisión en $\mathcal{V}$ contra Reducción de $\mathcal{T}$. Se consideran 25 métodos PG reportados en Triguero et al., GPPG y EMOPG en bases de datos pequeñas.

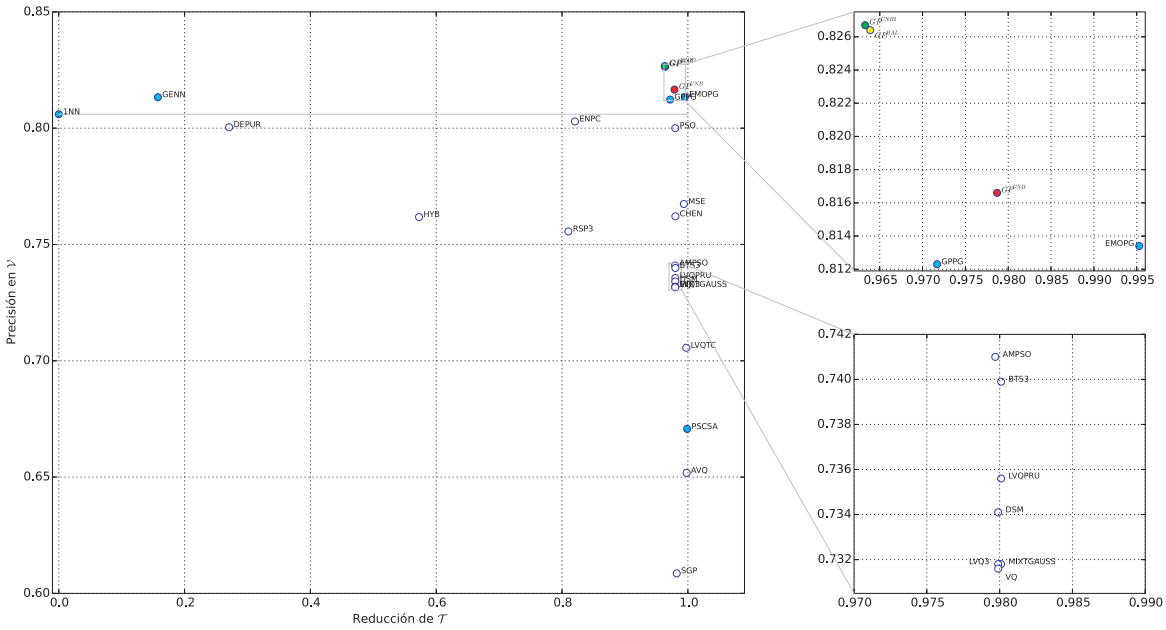

Figura 4.11: Precisión en $\mathcal{V}$ contra Reducción de $\mathcal{T}$. Se consideran 20 métodos PG reportados en Triguero et al., GPPG y EMOPG en bases de datos grandes.

Capítulo 5

Conclusiones

En esta contribución se han propuesto tres variantes de métodos para generar prototipos basados en GP para la tarea de clasificación de patrones. Una característica distintiva de nuestra formulación es que se basa en un procedimiento iterativo incremental. Los métodos propuestos se basan en el mismo enfoque para generar prototipos, la principal diferencia está en cómo se seleccionan las clases para las cuales se generarán prototipos dentro del algoritmo evolutivo: GP^{BAL} genera el mismos número por clase; GP^{UNB} genera prototipos dependiendo de la precisión de clasificación por clase; GP^{UNB2} genera prototipos solo para clases que no clasifican todas sus muestras correctamente.

Un experimental extenso estudio se llevó a cabo con el fin de evaluar el rendimiento de los enfoques propuestos. Se obtuvieron mejores resultados en conjuntos de datos grandes y conjuntos de datos con un solo tipo de atributo (ya sea numérico o nominal). Sin embargo, se obtuvieron resultados muy competitivos en conjuntos de datos pequeños y conjuntos de datos con atributos mixtos (numéricos y nominales). Los resultados experimentales muestran que los métodos propuestos son competitivos con técnicas del estado del arte, de hecho, los resultados obtenidos por los métodos propuestos en conjuntos de datos grandes son los mejores reportados hasta el momento para el conjunto de bases utilizadas como referencia. Además de su desempeño competitivo, nuestros métodos propuestos son simples de implementar.

5.1. Trabajos Futuros

Como parte de trabajos futuros, se encuentran los siguientes:

1. Diseñar una estrategia para eliminar los prototipos que no asisten a la clasificación para que se puedan conseguir mejores tasas de reducción de conjunto y hacer más eficiente el proceso.

2. Probar con diferentes medidas de distancias, con el fin de que el rendimiento de problemas heterogéneos (es decir, problemas con atributos tanto numéricos como nominales) no se vea afectado. Una posible función podría ser el Valor de Diferencia de Métricas Heterogéneas (HVDM, por sus siglas en inglés) [Wilson and Martinez, 1997].

3. Aplicar RPROP para optimizar las constantes en GP, porque se observo que GP utiliza una gran cantidad de constantes para la generación de prototipos, en realidad las instancias también se podrían considerar como constantes.

Referencias

[Alcalá-Fdez et al., 2011] Alcalá-Fdez, J., Fernandez, A., Luengo, J., Derrac, J., García, S., Sánchez, L., and Herrera, F. (2011). KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework. *Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287.

[Bache and Lichman, 2013] Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences.

[Bishop, 1995] Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA.

[Cordelia et al., 2005] Cordelia, L., De Stefano, C., Fontanella, F., and Marcelli, A. (2005). Genetic programming for generating prototypes in classification problems. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1149–1155.

[Cordella et al., 2006] Cordella, L., De Stefano, C., Fontanella, F., and Marcelli, A. (2006). Looking for prototypes by genetic programming. In Zheng, N., Jiang, X., and Lan, X., editors, *Advances in Machine Vision, Image Processing, and Pattern Analysis*, volume 4153 of *Lecture Notes in Computer Science*, pages 152–159. Springer Berlin Heidelberg.

[de Castro and Timmis, 2002] de Castro, L. N. and Timmis, J. (2002). *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer.

[Duda et al., 2000] Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification (2nd Edition)*. Wiley-Interscience.

[Escalante et al., 2013] Escalante, H., Mendoza, K., Graff, M., and Morales-Reyes, A. (2013). Genetic programming of prototypes for pattern classification. In Sanches, J., Micó, L., and Cardoso, J., editors, *Pattern Recognition and Image Analysis*, volume 7887 of *Lecture Notes in Computer Science*, pages 100–107. Springer Berlin Heidelberg.

[Espejo et al., 2010] Espejo, P., Ventura, S., and Herrera, F. (2010). A survey on the application of genetic programming to classification. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 40(2):121–144.

[Fayed et al., 2007] Fayed, H. A., Hashem, S. R., and Atiya, A. F. (2007). Self-generating prototypes for pattern classification. *Pattern Recognition*, 40(5):1498–1509.

[Fernández and Isasi, 2004] Fernández, F. and Isasi, P. (2004). Evolutionary design of nearest prototype classifiers. *Journal of Heuristics*, 10(4):431–454.

[Garain, 2008] Garain, U. (2008). Prototype reduction using an artificial immune model. *Pattern Anal. Appl.*, 11(3-4):353–363.

[Garcia et al., 2012] Garcia, S., Derrac, J., Cano, J., and Herrera, F. (2012). Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(3):417–435.

[Han et al., 2005] Han, J., Kamber, M., and Pei, J. (2005). *Data Mining: Concepts and Techniques (2nd Edition)*. ELSEVIER, Morgan Kaufmann.

[Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, 2 edition.

[Koplowitz and Brown, 1981] Koplowitz, J. and Brown, T. A. (1981). On the relation of performance to editing in nearest neighbor rules. *Pattern Recognition*, 13(3):251 – 255.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

[Poli et al., 2008] Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`. (With contributions by J. R. Koza).

[Rosales-Pérez et al., 2014] Rosales-Pérez, A., Escalante, H., Coello Coello, C., Gonzalez, J., and Reyes-Garcia, C. (2014). An evolutionary multi-objective approach for prototype generation. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 1100–1107.

[Triguero et al., 2012] Triguero, I., Derrac, J., Garcia, S., and Herrera, F. (2012). A taxonomy and experimental study on prototype generation for nearest neighbor classification. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(1):86–100.

[Wilson and Martinez, 1997] Wilson, D. and Martinez, T. (1997). Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research (JAIR)*, 6(1):1–34.

[Wilson and Martinez, 2000] Wilson, D. and Martinez, T. (2000). Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286.

Glosario

ANN *Artificial Neural Network* - Red Neuronal Artificial

EMOPG *Evolutionary Multi-Objective approach for Prototype Generation* - Enfoque Evolutivo Multi-Objetivo para la Generación de Prototipos

ENPC *Evolutionary Nearest Prototype Classifiers* - Clasificadores Evolutivos basados en Prototipos más Cercanos

GENN *Generalized Editing using Nearest Neighbor* - Edición Generalizada utilizando Vecinos más Cercanos

GP *Genetic Programming* - Programación Genética

GPPG *Genetic Programming for Prototype Generation* - Programación Genética para la Generación de Prototipos

HVDM *Heterogeneous Value Difference Metric* - Valor de Diferencia de Métricas Heterogéneas

kNN *k Nearest Neighbor* - k Vecinos más Cercanos

NN *Nearest Neighbor* - Vecino más Cercano

PAES *Pareto Archived Evolution Strategy*

PG *Prototype Generation* - Generación de Prototipos

PS *Prototype Selection* - Selección de Prototipos

PSCSA *Prototype Selection Clonal Selection Algorithm* - Algoritmo de Selección Clonal para la Selección de Prototipos