Universidad de Granada

Soft Computing y Sistemas Inteligentes

Trabajo de Investigación Tutelada

Nuevos modelos de clasificación mediante algoritmos evolutivos

Granada, Septiembre de 2011

Autor:
Alberto Cano Rojas
Director:
Dr. Sebastián Ventura Soto

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

I

INTRODUCCIÓN

1. COMPUTACIÓN EVOLUTIVA

La computación evolutiva engloba un amplio conjunto de técnicas de resolución de problemas basados en la emulación de los procesos naturales de evolución. La principal aportación de la computación evolutiva a la metodología de resolución de problemas consiste en el uso de mecanismos de selección de soluciones potenciales y de construcción de nuevos candidatos por recombinación de características de otras ya presentes, de modo parecido a como ocurre con la evolución de los organismos. No se trata tanto de reproducir ciertos fenómenos que se suceden en la naturaleza sino de aprovechar las ideas genéricas que hay detrás de ellos. En el momento en que se tienen varios candidatos como solución para un problema, surge la necesidad de establecer criterios de calidad y de selección y también la idea de combinar características de las buenas soluciones para obtener otras mejores. Dado que fue en el mundo natural donde primeramente se han planteado problemas de este tipo, no tiene nada de extraño el que al aplicar tales ideas en la resolución de problemas científicos y técnicos se obtengan procedimientos bastante parecidos a los que ya se encuentran por la naturaleza tras un largo periodo de adaptación.

Bajo el paraguas de los algoritmos evolutivos exiten diferentes paradigmas tales como los algoritmos genéticos, los algoritmos meméticos, la evolución diferencial o la programación genética, que se originaron con distintas motivaciones y enfoques. Sin embargo, actualmente los algoritmos tienden a combinar características de otros paradigmas y campos de estudio en la búsqueda de una hibridación que mejore la solución al problema.

Programación Genética

La programación genética (PG) es un paradigma de computación evolutiva, orientada a encontrar programas de ordenador que realicen una tarea definida por el usuario. Se trata de una especialización de los algoritmos genéticos donde cada individuo es un programa de ordenador. Por tanto, puede considerarse una técnica de aprendizaje automático usada para optimizar una población de programas de ordenador según una heurística definida en función de la capacidad del programa para realizar una determinada tarea computacional, definida por el usuario.

Los primeros resultados de la aplicación de la PG fueron reportados por Stephen F. Smith [74] y Nichael L. Cramer [28]. Sin embargo, John R. Koza [56] es considerado el padre de este paradigma, siendo quien lo aplicó a la resolución de varios problemas complejos.

La PG es un paradigma con grandes exigencias desde el punto de vista computacional. Por esta razón, en los años 90 sólo se aplicó a problemas relativamente sencillos. En la actualidad, las mejoras en el hardware y en la propia PG están permitiendo resolver problemas en áreas tales como la computación cuántica, diseño electrónico, programación de juegos, búsqueda y otros. Su flexibilidad en la representación de soluciones le permite abordar múltiples tipos de problemas de optimización y aprendizaje. La Figura 1.1 representa la estructura de un árbol de expresión. La evaluación de un árbol de expresión requiere del uso de una pila donde almacenar los resultados temporales de la evaluación de los nodos y la expresión se puede representar mediante notación postfija o notación polaca inversa (RPN). La notación polaca inversa no necesita usar paréntesis para indicar el orden de las operaciones mientras la aridad del operador sea fija.

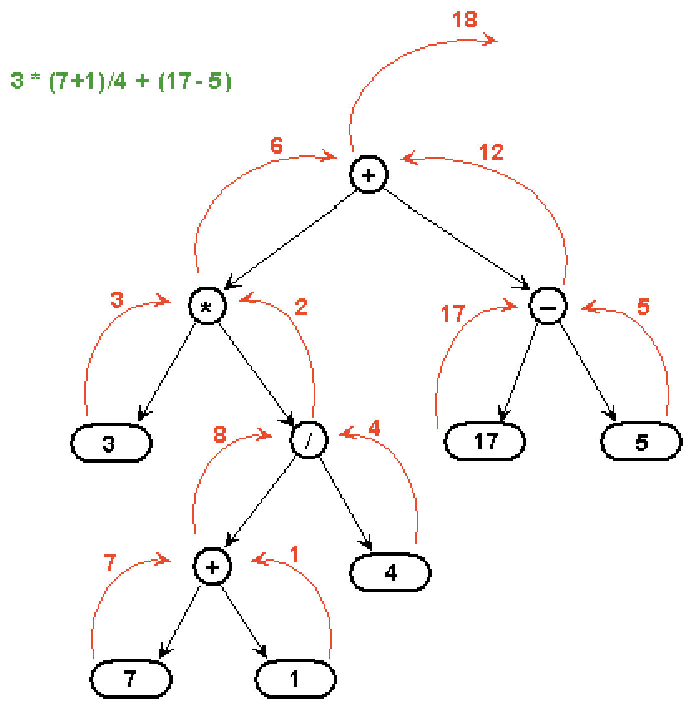


Figura 1.1: Árbol de expresión y su evaluación

2. MINERÍA DE DATOS

La minería de datos [24, 34, 39] se define como la extracción no trivial de información implícita, previamente desconocida y potencialmente útil, a partir de datos. En la actual sociedad de la información, donde día a día se multiplica la cantidad de datos almacenados, la minería de datos es una herramienta fundamental para analizarlos y explotarlos de forma eficaz. Las técnicas de minería de datos permiten obtener conocimiento a partir de las relaciones de los datos y proporcionan a los investigadores y usuarios conocimiento de los datos en forma de modelos descriptivos (clustering y segmentación), modelos predictivos (regresión y clasificación), reglas de asociación, etc.

La minería de datos constituye la fase central del proceso de extracción de conocimiento (Knowledge Discovery), representado en la Figura 2.1. En este sentido la minería de datos es un punto de encuentro de diferentes disciplinas: la estadística, el aprendizaje automático, las técnicas de bases de datos y los sistemas para la toma de decisiones que, conjuntamente, permiten afrontar problemas actuales del tratamiento de la información.

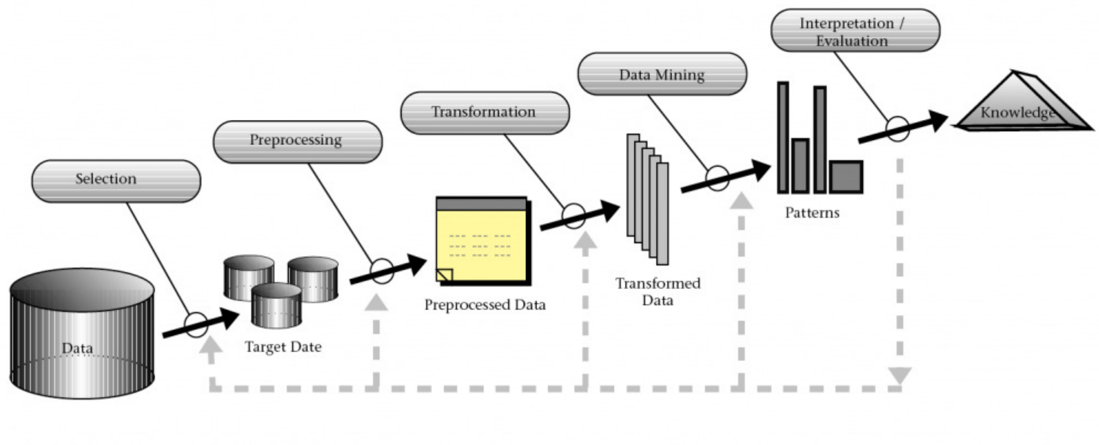


Figura 2.1: Proceso de extracción de conocimiento

Clasificación

La tarea de clasificación es un problema de aprendizaje supervisado que consiste en aprender un clasificador, a partir de ejemplos de entrenamiento etiquetados anteriormente, que establezca una correspondencia entre las entradas y la salida del sistema (etiqueta o clase). Esta etiqueta es categórica en los problemas de clasificación, mientras que es un valor real en los problemas de regresión. La bondad de un clasificador se mide por la tasa de acierto en la clasificación de nuevos ejemplos (generalización). Formalmente, dado un conjunto de ejemplos $X = \{x_1, x_2, ..., x_n\}$, un conjunto de etiquetas $Y = \{y_1, y_2, ..., y_m\}$ y un conjunto de ejemplos de entrenamiento $T = \{(x_i, y_k) \mid y_k \textit{ es la etiqueta de } x_i\}$, un clasificador establece una relación de X a Y de la forma $f(T, x) = y$.

Existe una gran variedad de técnicas de clasificación [55], tales como las redes neuronales [66], las máquinas de vector soporte [11], los clasificadores estadísticos [66] o los métodos de aprendizaje basados en las instancias [2]. Estas técnicas obtienen modelos de clasificación precisos pero se denominan de caja negra, puesto que no permiten conocer la lógica y el razonamiento detrás de la predicción de una clase, de forma que no se puede obtener conocimiento interesante y comprensible del modelo de los datos. Estos clasificadores no son interpretables por un experto y no es posible conocer cuáles son los atributos y condiciones relevantes que condujeron la predicción de la clase para un determinado ejemplo. Esta opacidad les impide ser utilizados en muchas aplicaciones del mundo real que requieren tanto de una alta exactitud como de la comprensibilidad del modelo aprendido, tales como en diagnóstico médico y evaluación de riesgos en créditos y finanzas [60].

Por otro lado, existen otras técnicas de aprendizaje automático, denominadas de caja blanca, que superan estas limitaciones y proporcionan modelos comprensibles con clasificadores interpretables que aportan conocimiento al usuario, tales como los árboles de decisión [86] y los sistemas basados en reglas [71].

Las métricas de calidad de los clasificadores se calculan en base a los valores de la matriz de confusión, que se obtiene a partir de la comparación de las etiquetas verdaderas de los ejemplos y las predichas por el clasificador.

3. MOTIVACIÓN DEL TRABAJO

Este trabajo resume la labor de investigación del autor con la supervisión de sus directores a lo largo del último año, apoyándose en los conocimientos adquiridos en los cursos del máster en Soft Computing y Sistemas Inteligentes, incluyendo algoritmos evolutivos, minería de datos, clasificación, sistemas basados en reglas, validación de resultados, etc.

La primera parte plantea y soluciona el problema de la escalabilidad en algoritmos evolutivos para clasificación, centrándose en sistemas basados en reglas mediante programación genética. Las tarjetas gráficas (GPUs) aportan un modelo de programación masivamente paralelo abierto a acelerar y solucionar múltiples problemas en aprendizaje automático. Por lo tanto, se propone un modelo de evaluación de reglas de clasificación mediante GPU y se realiza un estudio experimental para analizar sus ventajas y medir su rendimiento.

La segunda parte avanza en el problema de la interpretabilidad vs precisión de los sistemas basados en reglas, con el objetivo de encontrar soluciones de compromiso que sean por un lado clasificadores con una alta tasa de acierto, y por otro que aporten reglas sencillas, comprensibles e interpretables por un experto para poder obtener conocimiento de los datos. En este sentido, se proponen dos modelos, uno con el objetivo de maximizar la interpretabilidad y otro para maximizar la exactitud. Ambos modelos son sometidos a estudios experimentales sobre múltiples conjuntos de datos con los que se obtienen buenos resultados en base a las distintas métricas evaluadas.

Finalmente, la tercera parte rompe con los sistemas basados en reglas y se acerca a un modelo basado en las instancias y en la gravitación existente entre ellas. El resultado del trabajo en este campo es un modelo de clasificación gravitatorio que incorpora selección de características y emplea uno de los algoritmos más reconocidos en codificación real como optimizador de la función objetivo.

El conocimiento adquirido a través del estudio de distintos problemas abiertos en clasificación ha permitido al autor formarse en este campo sobre el que ya tenemos pendiente considerable trabajo futuro con el objetivo de realizar la tesis doctoral.

II
ESCALABILIDAD CON GPUS

4. EL PROBLEMA DE LA ESCALABILIDAD

El tiempo requerido por los algoritmos de aprendizaje en clasificación aumenta conforme crece el número de instancias y el número de atributos del conjunto de datos. El grado de complejidad computacional y la dependencia del número de instancias o del número atributos depende de cada algoritmo.

Los algoritmos de aprendizaje supervisado en clasificación evalúan la calidad de los clasificadores aprendidos sometiendo el conjunto de datos de entrenamiento a las reglas del clasificador y midiendo la calidad de su respuesta en la predicción de las etiquetas. Por lo tanto, es lógico que conforme mayor sea el número de ejemplos de entrenamiento, mayor sea el tiempo requerido en evaluar la calidad de los clasificadores. Por otro lado, el número de atributos también puede influir en el coste computacional del algoritmo, y dependerá de la filosofía de construcción del clasificador que tenga el algoritmo. Por ejemplo, los algoritmos de colonias de hormigas son muy sensibles al número de atributos ya que deben guardar un espacio de estados de la exploración de los diferentes atributos.

Centrándonos en algoritmos evolutivos de clasificación basados en reglas, y más concretamente de programación genética, nos encontramos con que cada individuo debe ser evaluado frente a todo el conjunto de datos de entrenamiento. Los individuos del algoritmo de programación genética para clasificación son expresiones compuestas por nodos que conforman reglas para resolver la correcta clasificación de un conjunto de instancias. El uso de programación genética para tareas de clasificación ha demostrado ser una técnica que obtiene buenas soluciones [33]. Un clasificador puede expresarse como un conjunto de reglas de tipo *Si–Entonces*, en las que el antecedente de cada regla está formado por una serie de condiciones que debe cumplir una instancia para que se considere que pertenece a la clase indicada en el consecuente.

La fase de evaluación de los algoritmos evolutivos consiste en tomar cada uno de los individuos de la población y evaluar su aptitud en la resolución del problema. Por lo tanto, el coste computacional debido al número de instancias hay que multiplicarlo por el número de individuos de la población, lo que hace que el aumento de cualquiera de los dos parámetros afecte en gran medida el tiempo de ejecución. Para evaluar la población, se interpreta el genotipo del individuo para obtener el fenotipo (regla o conjunto de reglas), que se evalúan sobre cada instancia del conjunto de datos, obteniendo el número de aciertos y fallos de cada regla sobre dicho conjunto de instancias. Concretamente, se obtienen el número de verdaderos positivos (T_P), falsos positivos (F_P), verdaderos negativos (T_N) y falsos negativos (F_N). Con estos datos se construye la matriz de confusión, que empleando distintas métricas para cada algoritmo, como por ejemplo la exactitud, se obtiene la aptitud (*fitness*) del individuo. Por lo tanto podemos dividir la fase de evaluación en dos pasos: evaluación de las instancias y cálculo del *fitness* usando los datos de la matriz de confusión.

Tradicionalmente el proceso de evaluación ha sido implementado de forma secuencial, tal y como se representa en el Algoritmo 1, por lo que su tiempo de ejecución aumenta conforme el número de instancias o el número de individuos se incrementa.

Algorithm 1 Evaluador de reglas secuencial

```
Require: population_size, number_instances
1: for each individual within the population do
2:     $T_P \leftarrow 0$, $T_N \leftarrow 0$, $F_P \leftarrow 0$, $F_N \leftarrow 0$
3:     for each instance from the dataset do
4:         if individual's rule covers actual instance then
5:             if individual's consequent matches predicted class then
6:                 $T_P$++
7:             else
8:                 $F_P$++
9:             end if
10:        else
11:            if individual's consequent doest not match predicted class then
12:                $T_N$++
13:            else
14:                $F_N$++
15:            end if
16:        end if
17:    end for
18:    individual fitness $\leftarrow$ ConfusionMatrix($T_P, T_N, F_P, F_N$);
19: end for
```

A continuación analizaremos las opciones de paralelización y optimización de la función de evaluación con el objetivo de reducir su coste computacional. El primer paso de la evaluación es por definición paralelo, la evaluación de cada regla sobre cada instancia es un proceso completamente independiente, por lo que éstas se pueden paralelizar en hilos independientes sin ninguna restricción. Su función es interpretar la expresión y comprobar si el resultado coincide con el valor predicho. El resultado de esta comparación se almacena para cada instancia y cada regla en modo de acierto o fallo. Este modelo de evaluación se representa en la Figura 4.1. El número de hilos es igual al número de evaluaciones en un determinado momento, es decir, el producto del número de individuos por el número de instancias, por lo que el número total de hilos puede ascender desde miles hasta millones de ellos.

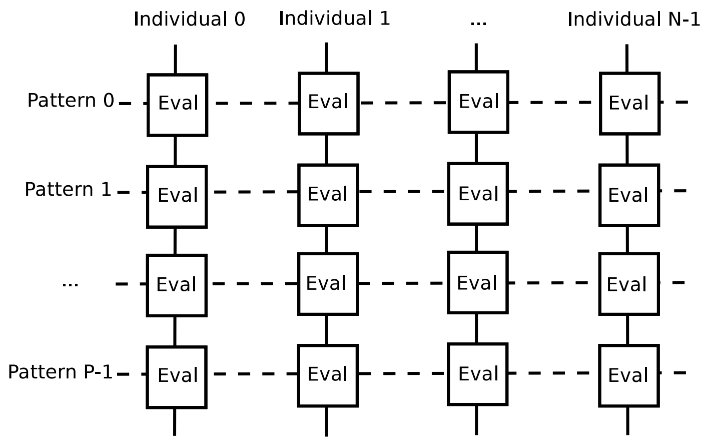


Figura 4.1: Modelo de evaluación paralelo

El segundo paso de la evaluación es una operación de reducción [52] y exige el recuento del número de aciertos y de fallos de cada regla y para todas las reglas. El recuento es un proceso nativamente secuencial, pero puede ser paralelizado mediante la suma de semisumas realizadas en paralelo para cada regla. Además, los diferentes procesos de recuento para cada una de las reglas pueden ser paralelizados sin ninguna interferencia [15, 16, 14, 17, 12, 20, 21].

El primer paso de la evaluación finaliza con el almacenamiento de los aciertos y fallos para cada instancia y regla. El segundo paso debe contarlos de manera eficiente y para ello, cada regla emplea múltiples hilos que realizan el recuento de los resultados, obteniendo semisumas parciales. Posteriormente, se realiza la suma total de dichas semisumas. Un diseño eficiente de este segundo paso requiere tener consideraciones importantes acerca del hardware y de cómo se almacenan los resultados en memoria.

En la próxima sección se presenta el modelo de programación CUDA, mientras que el desarrollo del modelo propuesto de evaluador en GPU se muestra en la sección 6.

5. MODELO DE PROGRAMACIÓN CUDA

CUDA (Compute Unified Device Architecture) es una arquitectura de cálculo paralelo de NVIDIA que aprovecha la gran potencia de la GPU para proporcionar un incremento extraordinario en el rendimiento del sistema.

Arquitectura de la GPU

La GPU es un procesador dedicado que tradicionalmente se ha dedicado exclusivamente al renderizado de gráficos en videojuegos o aplicaciones 3D interactivas.

Hoy en día superan en gran medida el rendimiento de una CPU en operaciones aritméticas y en ancho de banda en transferencias de memoria. La Figura 5.1 representa la evolución del potencial de cálculo en FLOPs (Floating-Point Operations per Second) de las CPUs y las GPUs a lo largo de los últimos años. Su gran potencia se debe a la alta especialización en operaciones de cálculo con valores en punto flotante, predominantes en los gráficos 3D. Conllevan un alto grado de paralelismo inherente, al ser sus unidades fundamentales de cálculo completamente independientes y dedicadas al procesamiento de los vértices y píxeles de las imágenes.

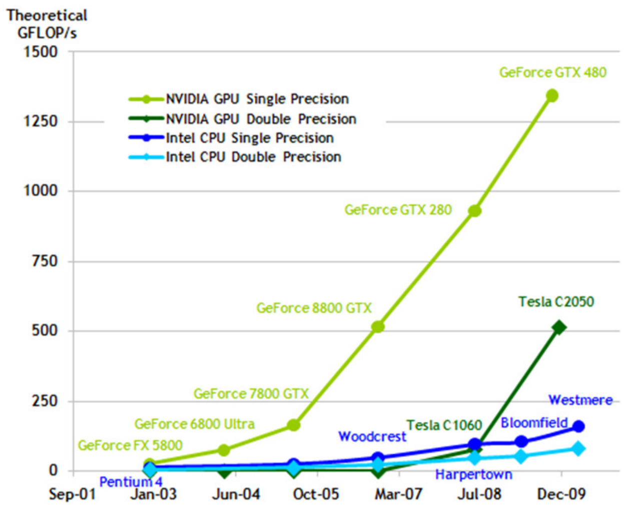


Figura 5.1: Evolución de las FLOPs de las CPUs y las GPUs

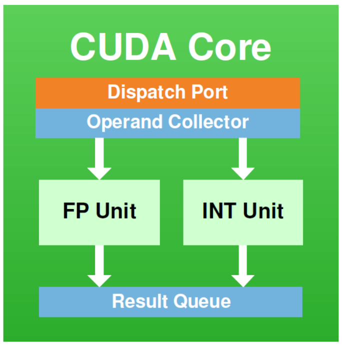


Figura 5.2: Shader o núcleo

Desde 2006, las arquitecturas unifican el procesamiento en unidades versátiles denominadas shaders que poseen una unidad de enteros y otra de punto flotante. Estos shaders o núcleos, representados en la Figura 5.2, se agrupan en una unidad conocida como multiprocesador (Streaming Multiprocessor, SM), ilustrado en la Figura 5.3, que además contiene algo de memoria compartida entre los núcleos y que gestiona la planificación y ejecución de los hilos en los núcleos de su multiprocesador. Una GPU se compone de varios grupos de multiprocesadores interconectados a una memoria global GDDR (Graphics Double Data Rate). El número de núcleos por multiprocesador depende de la generación de la gráfica, y el número de multiprocesadores en la GPU determina su potencia máxima y su modelo dentro de la generación.

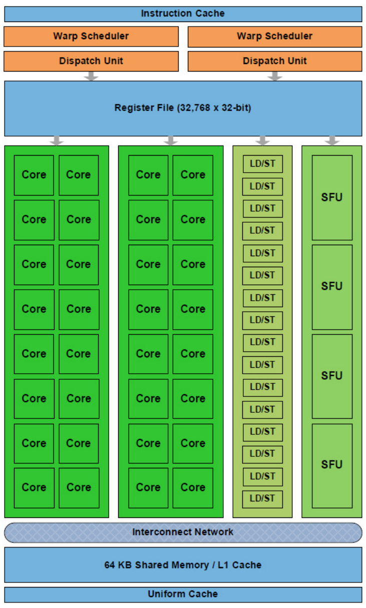


Figura 5.3: Streaming Multiprocessor

Kernels

CUDA es un entorno de desarrollo de algoritmos en GPU que extiende el lenguaje de programación C. Las funciones a ejecutar se denominan kernels. Cuando se realizan llamadas a funciones kernel, éstas se ejecutan N veces en paralelo en N hilos CUDA diferentes, a diferencia de las funciones tradicionales del lenguaje C.

Un kernel se define usando el prefijo *__global__* en su declaración y el número de hilos que ejecuta se determina en cada llamada al kernel. Cada hilo del kernel se identifica mediante un threadID que es accesible dentro del kernel mediante las variables $threadIdx$.

Como ejemplo inicial, el siguiente código realiza la suma de dos vectores A y B de tamaño N y almacena el resultado en el vector C.

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float * C) {
  int i = threadIdx.x;
  C[i] = A[i] + B[i];
}

int main() {
  // Kernel invocation with N threads
  VecAdd<<<1, N>>>(A, B, C);
}
```

Jerarquía de Hilos

Los hilos del kernel se pueden identificar haciendo uso de la variable $threadIdx$, un vector de 3 componentes que permite establecer configuraciones unidimensionales, bidimensionales o tridimensionales de hilos. Estos hilos se agrupan en un bloque por lo que se proporciona una computación versátil para dominios de vectores, matrices o volúmenes.

El índice de un hilo y su threadID están relacionados de la siguiente manera:

- Bloque unidimensional: el threadID se corresponde con el índice x para un bloque de dimensión Dx.

- Bloque bidimensional: para un bloque de dimensiones (Dx, Dy) el threadID de un hilo con índice (x, y) es $(Dx * y + x)$.
- Bloque tridimensional: para un bloque de dimensiones (Dx, Dy, Dz) el threadID de un hilo con índice (x, y, z) es $(Dx * Dy * z + Dx * y + x)$.

Como ejemplo, el siguiente código realiza la suma de dos matrices A y B de tamaño NxN y almacena el resultado en la matriz C.

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N], float C[N][N])
{
  int i = threadIdx.x;
  int j = threadIdx.y;
  C[i][j] = A[i][j] + B[i][j];
}

int main()
{
  ...
  // Kernel invocation with one block of N * N * 1 threads
  int numBlocks = 1;
  dim3 threadsPerBlock(N, N);
  MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
}
```

Existe un límite del número de hilos por bloque, puesto que todos los hilos de un bloque se ejecutan en un multiprocesador y deben compartir los recursos de memoria de dicho multiprocesador. En las GPUs actuales un bloque de puede contener hasta 1024 hilos, es decir $Dx * Dy * Dz \leq 1024$.

Sin embargo, un kernel puede ejecutar concurrentemente múltiples bloques de hilos, por lo que el número total de hilos es igual al producto del número de bloques por el número de hilos por bloque. Los bloques se organizan a su vez en configuraciones unidimensionales o bidimensionales como ilustra la Figura 5.4.

El número de hilos por bloque y el número de bloques en el grid (matriz de bloques de hilos) se especifican como parámetros en la llamada a ejecución del kernel.

Cada bloque del grid se identifica mediante una variable $blockIdx$ accesible dentro del kernel. La dimensión del bloque también se puede obtener mediante la variable $blockDim$.

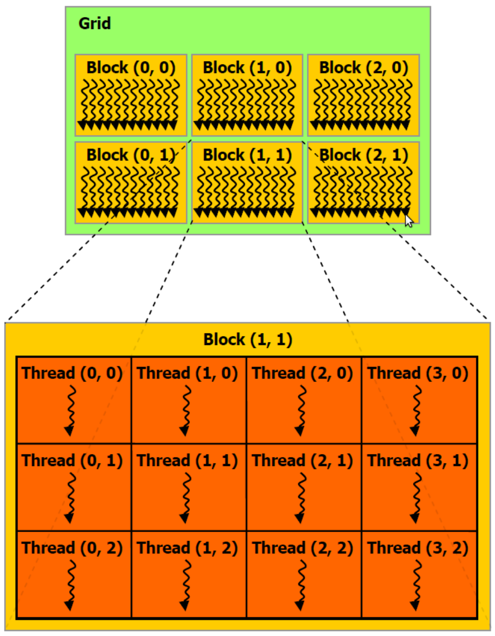


Figura 5.4: Grid de bloques de hilos

Extendiendo el código del ejemplo anterior de $MatAdd()$ para manejar múltiples bloques y por lo tanto, matrices de mayores dimensiones, quedaría:

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N], float C[N][N]) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  int j = blockIdx.y * blockDim.y + threadIdx.y;

  if (i < N && j < N)
    C[i][j] = A[i][j] + B[i][j];
}

int main() {
  ...
  // Kernel invocation
  dim3 threadsPerBlock(16, 16);
  dim3 numBlocks(N / threadsPerBlock.x, N / threadsPerBlock.y);
  MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
}
```

El bloque de hilos tiene un tamaño de 16x16 (256 hilos). El grid se compone de la cantidad suficiente de bloques como para computar la suma sobre todos los elementos de las matrices.

El modelo de programación CUDA mediante la jerarquía de hilos y bloques permite por lo tanto la identificación unívoca de cualquier hilo en un espacio de 5 dimensiones representado en la Figura 5.5 .

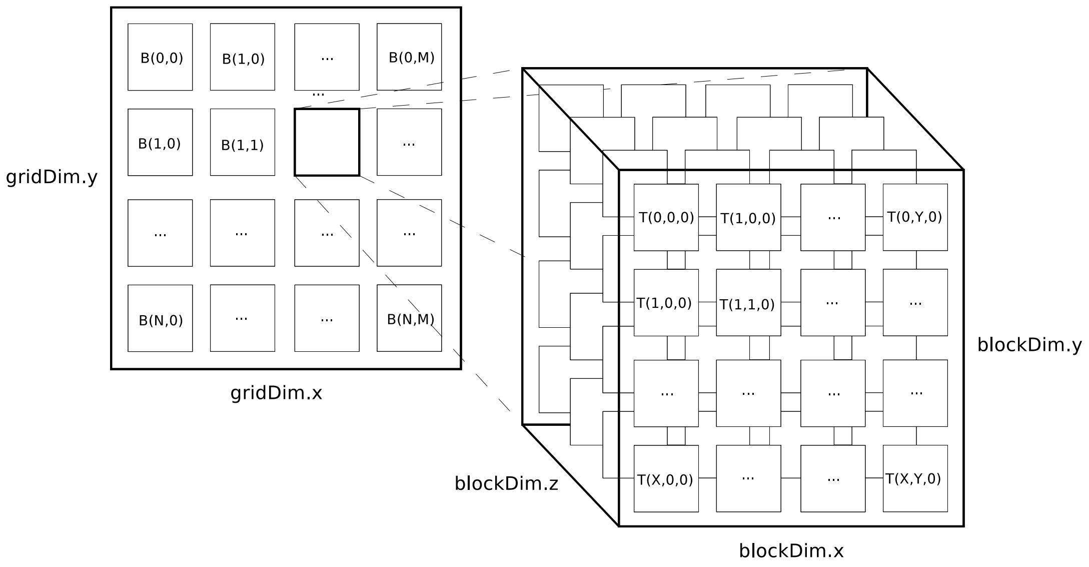


Figura 5.5: Jerarquía de hilos y bloques

Los bloques se ejecutan independientemente en cada multiprocesador. Debe ser posible ejecutarlos en cualquier orden, en paralelo o en serie. Esta independencia permite que los bloques sean planificados en cualquier orden y en cualquier multiprocesador, facilitando a los desarrolladores la programación de código que escale con el número de procesadores.

Los hilos dentro de un bloque pueden cooperar compartiendo datos mediante la memoria compartida y sincronizando su ejecución para coordinar los accesos a memoria. La sincronización de hilos dentro de un bloque se lleva a cabo mediante la llamada a *__syncthreads*() que actúa como barrera en la que los hilos se bloquean hasta que todos los hilos del bloque alcancen la barrera.

Jerarquía de Memoria

La memoria de la GPU necesita ser rápida y suficientemente amplia para procesar millones de polígonos y texturas. En la Figura 5.6 se representa la evolución del ancho de banda de la memoria principal en las CPUs y GPUs a lo largo de los últimos años.

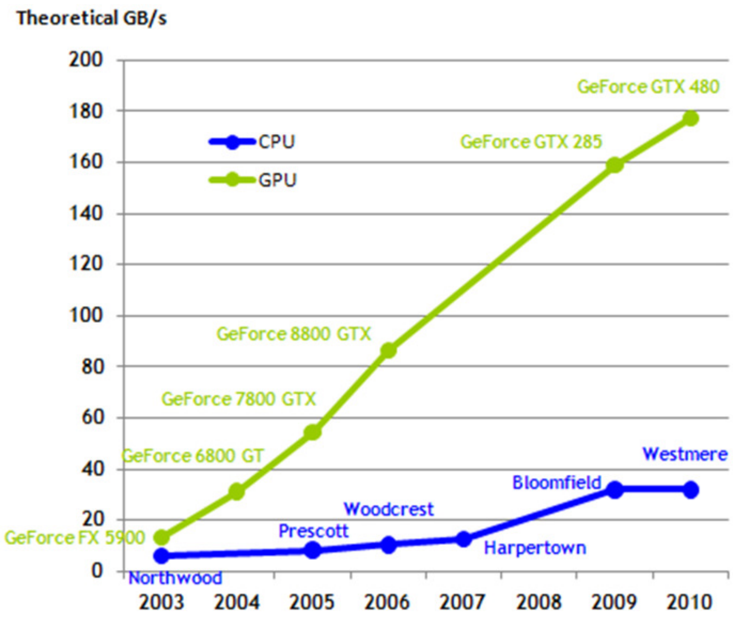


Figura 5.6: Evolución del ancho de banda de la memoria de las CPUs y las GPUs

Existen cuatro grandes espacios de memoria diferenciados: local, global, de constantes y compartida. Cada uno de dichos espacios de memoria están especializados para unas determinadas funciones. Se diferencian en su tiempo y modo de acceso y en el tiempo de vida de los datos que contienen.

Cada hilo tiene un espacio de memoria local en forma de registros del multiprocesador para almacenar variables locales al hilo. El número de registros por multiprocesador es de 16384 o 32768, dependiendo de la generación de la GPU, este factor limitará el número de bloques que se pueden ejecutar concurrentemente en un multiprocesador, es decir, el grado de ocupación del multiprocesador. Por ejemplo, un kernel que consuma 24 registros, ejecutado con 256 hilos por bloque, nos dará que un bloque necesita para su ejecución 6144 registros. Con tal demanda de registros, podremos servir concurrentemente un máximo de 2 o 4 bloques por multiprocesador. Es interesante por lo tanto, minimizar el consumo de registros para maximizar la ocupación del multiprocesador.

La memoria global en forma de chips GDDR proporciona un espacio de memoria muy amplio de hasta varios GB que comparten todos los hilos de todos los bloques. Sin embargo, al encontrarse fuera del chip de la GPU, sufre de una alta latencia en su acceso de alrededor de 400-800 ciclos. Todos los hilos pueden leer y escribir en la memoria global, donde deberán almacenar el resultado de la ejecución del kernel.

La memoria de constantes es una zona especializada de la memoria global en la que muchos hilos pueden leer el mismo dato simultáneamente. Esta memoria se encuentra limitada a 64 KB por multiprocesador. Un valor que se lea de la memoria de constantes se sirve a todos los hilos del warp (grupo 32 hilos que entran en ejecución), resultando en el servicio de 32 lecturas de memoria en un único acceso. Esto proporciona una caché muy rápida que sirva a múltiples accesos simultáneos a memoria.

La memoria compartida es una zona de memoria construida en el multiprocesador que se comparte entre todos los hilos de un bloque. Su tamaño es muy reducido, apenas de 16 KB. Sin embargo, al encontrarse dentro del multiprocesador proporciona un acceso con mínima latencia y su contenido sólo se mantiene durante la ejecución de un bloque, por lo que cuando éste termina su ejecución el contenido de la memoria compartida se desecha. Los kernels que leen o escriben un rango conocido de memoria con localidad espacial o temporal pueden emplear la memoria compartida como una caché administrada vía software donde cooperar. La memoria compartida proporciona un método natural de cooperación entre los hilos con mínima latencia, reduciendo los accesos a memoria global y mejorando la velocidad de ejecución. De nuevo, el uso que se haga de memoria compartida determinará el máximo número de bloques que se podrán ejecutar concurrentemente en el multiprocesador.

Para evitar desperdiciar cientos de ciclos esperando que sean servidos los accesos de lectura o escritura en memoria global, estos se suelen agrupar en accesos coalescientes aprovechando la planificación del warp para solapar las latencias de acceso.

Se dice que los accesos son coalescientes si los hilos en el warp acceden a cualquier palabra en cualquier orden y se emite una única instrucción de acceso a memoria para el acceso al segmento direccionado. Una buena forma de lograrlo es hacer que el hilo i-ésimo acceda a la posición iésima de memoria, así cada hilo accederá a su dirección efectiva pero el conjunto del warp se servirá con un único acceso a memoria global. Garantizar la coalescencia en los accesos a memoria es una de los criterios de mayor prioridad a la hora de optimizar la ejecución de los kernels en la GPU.

6. EVALUACIÓN DE REGLAS EN GPU

En la sección 4 se detalló la función de evaluación y se observó que el proceso de evaluación puede dividirse en dos fases completamente paralelizables. El primer paso de la evaluación finaliza con el almacenamiento de los aciertos y fallos para cada instancia y regla. El segundo paso debe contarlos de manera eficiente y para ello cada regla emplea un número determinado de hilos en donde cada uno de los hilos realiza el conteo de una parte de los resultados. Posteriormente, se realiza la suma total de estas semisumas parciales.

Las decisiones de diseño del evaluador en GPU han seguido las recomendaciones de la guía de programación de NVIDIA CUDA y del manual de buenas prácticas [90]. La guía de programación CUDA recomienda un número de hilos que sea múltiplo del tamaño del warp (conjunto de hilos que entran en ejecución por el despachador) que en las arquitecturas actuales es de 32 hilos. Una buena aproximación sería bloques de hilos de 32, 64, 96 o 128 hilos. Empleando los resultados de nuestra experimentación, hemos concluido que el número óptimo de hilos para nuestro problema es de 128, por lo que procederemos a realizar la explicación con este número, aunque podría variar en futuras arquitecturas.

La paralelización de la primera fase de la evaluación en la que se determina si una regla cubre o no a una instancia es inmediata. Cada hilo representa la evaluación de un individuo sobre una instancia. Los hilos se agrupan en conjuntos de 128 para formar un bloque de hilos, requiriendo por lo tanto un número de bloques de (N^o instancias / 128) para la evaluación de un individuo. A su vez, esto debe extenderse para todos los individuos configurando así una matriz bidimensional de bloques de hilos de tamaño (N^o instancias / 128) x N^o individuos, representada en la Figura 6.1. Es interesante destacar que en conjuntos de datos grandes con una población numerosa, el número total de hilos en la GPU alcanzará una cifra de millones de hilos de ejecución.

Una primera aproximación al recuento paralelo de los resultados de todas las evaluaciones sería que el primer hilo sumase los (N^o instancias / 128) primeros valores, el segundo los (N^o instancias / 128) siguientes, etc. Sin embargo, esta forma es

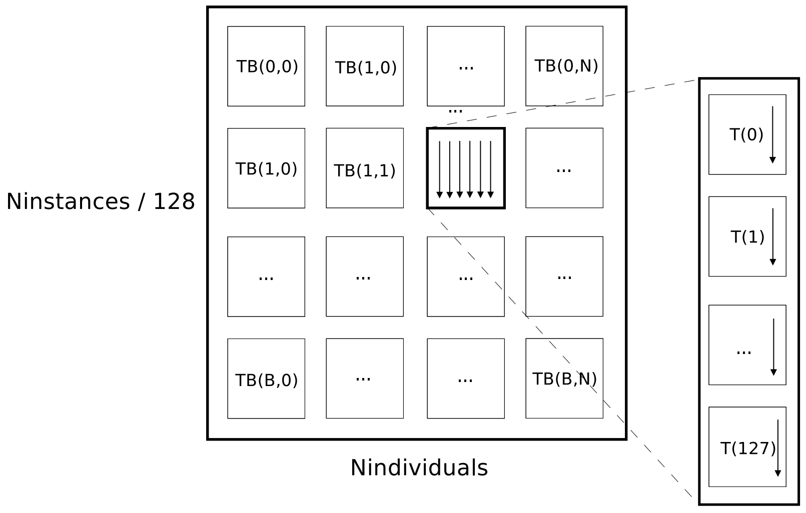


Figura 6.1: Matriz de bloques de hilos de ejecución

ineficiente en el sentido de que no favorece la coalescencia en los accesos a memoria. Las posiciones de memoria a las que acceden los hilos del warp en ejecución se encuentran distanciadas (Nº instancias / 128) bytes, por lo que para servir dichos accesos el controlador de memoria las serializará resultando en múltiples accesos a memoria, uno para cada valor.

Una segunda aproximación sería que el primer hilo sumase los resultados de los valores en las posiciones de memoria 0, 128, 256... el segundo hilo las posiciones 1, 129, 257... hasta llegar al último hilo que sumaría las posiciones 127, 255, 383... De esta forma tenemos igualmente 128 hilos realizando semisumas en paralelo, pero además los hilos del warp acceden a posiciones de memorias consecutivas de memoria por lo que empleando una única transferencia de memoria de 32 bytes es suficiente para traernos todos los datos de memoria necesarios. Se dice que este tipo de acceso es coalesciente y es más eficiente al poder disponer de muchos más datos en un único acceso a memoria, evitando latencias innecesarias. La paralelización para cada individuo se realiza mediante la asignación a cada individuo de un bloque de hilos independiente. Este esquema con la segunda aproximación se refleja en la Figura 6.2.

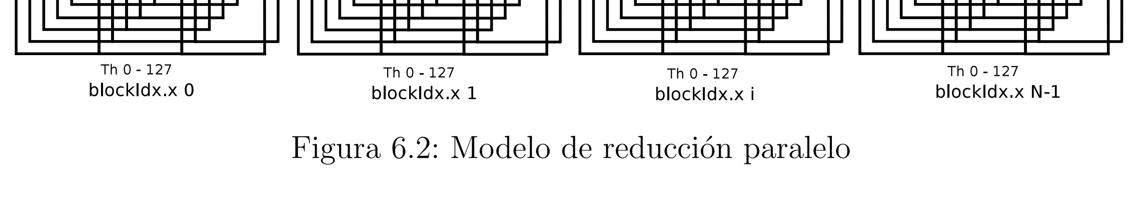


Figura 6.2: Modelo de reducción paralelo

Estudio experimental del modelo GPU

En esta sección se detalla la configuración de los experimentos, los conjuntos de datos, los algoritmos paralelizados y los parámetros de ejecución.

Algoritmos paralelizados

Para mostrar la flexibilidad y la aplicabilidad de nuestro modelo, tres algoritmos de clasificación de programación genética propuestos en la literatura se paralelizarán con nuestra propuesta, de la misma forma que se podría aplicar a otros algoritmos o paradigmas. Estos algoritmos están implementados en Java en el software de computación evolutiva JCLEC [80, 13]. A continuación se detallan las especificaciones principales de cada una de las propuestas que han sido consideradas en el estudio.

Algoritmo de Falco

Falco, Cioppa y Tarantino [29] proponen un método para obtener el *fitness* del clasificador mediante la evaluación de los antecedentes sobre todas las instancias en el conjunto de datos. Para ello, emplean los operadores lógicos AND, OR y NOT, los operadores relacionales $=, <, >, \leq, \geq$ y dos operadores de intervalos (IN, OUT). El proceso de evolución se repite tantas veces como clases tiene el conjunto de datos. En cada iteración, el algoritmo se centra en una clase y mantiene la mejor regla obtenida para construir el clasificador.

La función de aptitud calcula la diferencia entre el número de casos en los que la regla predice correctamente la pertenencia o no de la clase y el número de ejemplos en los que ocurre lo contrario y la predicción es incorrecta. Finalmente, la función de aptitud se define como:

$$fitness = nI - ((T_P + T_N) - (F_P + F_N)) + \alpha * N$$

donde nI es el número de instancias, α es un valor entre 0 y 1, y N es el número de nodos en la regla. Cuanto más cercano esté α a 1, más importancia se le otorga a la simplicidad de la regla.

Algoritmo de Tan

Tan, Tay, Lee y Heng [76] proponen una versión modificada del algoritmo de estado estacionario que utiliza una población externa y el elitismo para asegurarse de que los mejores individuos de la generación actual sobrevivan en la próxima generación.

La función de aptitud combina dos indicadores que son comunes en el dominio, la sensibilidad (Se) y la especificidad (Sp), que se define de la siguiente manera:

$$Se = \frac{T_P}{T_P + w1 * F_N} \qquad Sp = \frac{T_N}{T_N + w2 * F_P}$$

Los parámetros $w1$ y $w2$ se utilizan para ponderar la influencia de los falsos negativos y falsos positivos en el cálculo de la aptitud. Esto es muy importante porque estos valores son fundamentales en cierto tipo de problemas como el diagnóstico médico. La disminución de $w1$ o el aumento de $w2$ generalmente mejora los resultados, pero también aumenta el número de reglas. El rango de [0,2-1] por $w1$ y [1-20] por $w2$ suele ser razonable para la mayoría de los casos. Finalmente, la función de aptitud se define por el producto de estos dos parámetros.

$$fitness = Se * Sp$$

La propuesta de Tan et al. es similar a la de Falco et al. excepto el operador OR, pues la combinación de AND y NOT puede generar todas las reglas necesarias. Por lo tanto, la simplicidad de las reglas se ve afectada. Tan et al. introduce también la técnica de *token competition* propuesta por Wond y Leung [84] y se emplea como un enfoque para promover la diversidad. La mayoría de las veces, sólo unas cuantas reglas son útiles y cubren la mayoría de los casos, mientras que la mayoría son redundantes. Esta técnica es una manera eficaz de eliminar las reglas redundantes.

Algoritmo de Bojarczuk

Bojarczuk, Lopes y Freitas [10] presentan un método en el que se evalúa cada regla para todas las clases de forma simultánea para una determinada instancia. El clasificador se construye tomando la mejor regla para cada clase generada durante el proceso evolutivo. Este algoritmo no tiene operador de mutación.

Esta propuesta utiliza los operadores lógicos AND, OR y NOT, de esta manera el tamaño de las reglas generadas se reduce. Los modelos de PG no suelen producir soluciones simples y es necesario evaluar la comprensibilidad de la solución. La comprensibilidad de una regla es inversamente proporcional a su tamaño. Por lo tanto, definen la simplicidad (Sy) de una regla como:

$$Sy = \frac{maxnodes - 0{,}5 * numnodes - 0{,}5}{maxnodes - 1}$$

donde *maxnodes* es la profundidad máxima del árbol sintáctico, *numnodes* es el número de nodos de la regla, y *Se* y *Sp* son la sensibilidad y la especificidad descritos por Tan et al. empleando los pesos $w1$ y $w2$ igual a 1. Finalmente, el *fitness* es el producto de estos tres parámetros.

$$fitness = Se * Sp * Sy$$

Comparación con BioHEL

Uno de los trabajos más recientes y similar a nuestra propuesta es la de M. Franco et al [37]. En este trabajo se acelera la evaluación del sistema de BioHEL con GPGPUs. BioHEL es un sistema de aprendizaje evolutivo diseñado para hacer frente a grandes conjuntos de datos. Los autores proporcionan los resultados, la información de perfiles CUDA y el software en el sitio web `http://www.cs.nott.ac.uk/~mxf/biohel`. Los experimentos llevados a cabo comparan nuestro modelo y su aceleración con la aceleración de la versión CUDA de BioHEL sobre varios conjuntos de datos con el fin de demostrar las mejoras aportadas por nuestro modelo de paralelización. Las opciones de configuración del sistema BioHEL fueron los proporcionados por los autores en los archivos de configuración.

Conjuntos de datos

Para evaluar el desempeño del modelo de evaluación propuesto de PG, hemos seleccionado varios conjuntos de datos del repositorio de UCI [64] y del sitio web de la herramienta de software KEEL [4] y de su repositorio de conjuntos de datos [3]. Estos conjuntos de datos son muy variados teniendo en cuenta diferentes grados de complejidad. El número de ejemplos varía desde el más simple que contiene 150 instancias, hasta los más complejos que contienen hasta un millón de instancias. Además, el número de atributos y clases son diferentes en diferentes bases de datos. Esta información se resume en la Tabla 6.1. La gran variedad de conjuntos de datos considerados nos permite evaluar los resultados del modelo bajo diferentes complejidades del problema. Es interesante mencionar que algunos de estos conjuntos de datos tales como KDDcup o Poker no han sido comúnmente tratados hasta la fecha, porque no son habitualmente tratables por los modelos tradicionales.

Tabla 6.1: Complejidad de los conjuntos de datos

Dataset	#Instancias	#Atributos	#Clases
Iris	150	4	3
New-thyroid	215	5	3
Ecoli	336	7	8
Contraceptive	1473	9	3
Thyroid	7200	21	3
Penbased	10992	16	10
Shuttle	58000	9	7
Connect-4	67557	42	3
KDDcup	494020	41	23
Poker	1025010	10	10

Configuración de la experimentación

El código de evaluación de la GPU se compila en una biblioteca compartida y se cargan en el software JCLEC [80] utilizando JNI. Utilizando dicha biblioteca, nuestro modelo puede ser fácilmente exportado y empleado en cualquier otro sistema de aprendizaje evolutivo.

Los experimentos se llevaron a cabo en dos PCs equipados con un procesador Intel Core i7 de cuatro núcleos a 2.66GHz y 12 GB de memoria principal DDR3. Un PC cuenta con dos NVIDIA GeForce GTX 285 con 2 GB de GDDR3 RAM de vídeo y el otro cuenta con dos NVIDIA GeForce GTX 480 con 1,5 GB de RAM de vídeo GDDR5. El sistema operativo fue en ambos Ubuntu 10.04 de 64 bits.

Resultados del modelo GPU

En esta sección se analizan los resultados de los experimentos, cuyo propósito es analizar el efecto de la complejidad del conjunto de datos sobre el desempeño del modelo de evaluación GPU y la escalabilidad de la propuesta. Cada algoritmo se ejecuta en todos los conjuntos de datos mediante un enfoque secuencial, un enfoque paralelo en CPU, y un enfoque GPU masivamente paralelo. La primera sección compara el desempeño de nuestra propuesta sobre los diferentes algoritmos. La segunda sección presenta los resultados del sistema BioHEL y los compara con los obtenidos por nuestra propuesta.

Resultados de nuestro modelo GPU

En esta sección se analiza el rendimiento alcanzado por nuestra propuesta con tres algoritmos de PG. El tiempo de ejecución y el incremento de velocidad de los algoritmos de clasificación sobre los diferentes conjuntos de datos considerados se muestran en las Tablas 6.2, 6.3, y 6.4 donde cada columna está etiquetada con la configuración de la ejecución, indicado de izquierda a derecha de la siguiente manera: el conjunto de datos, el tiempo de ejecución de la versión secuencial en Java expresado en segundos, la aceleración del modelo propuesto utilizando JNI y un hilo de la CPU, dos hilos de CPU, cuatro hilos de CPU, una GPU GTX 285, dos GPUs GTX 285, y con una y dos GPUs GTX 480. Los resultados corresponden a la ejecución de los algoritmos con una población de 200 individuos y 100 generaciones.

Tabla 6.2: Resultados algoritmo de Falco et al.

Tiempo de ejecución (s)		Aceleración						
Dataset	Java	1 CPU	2 CPU	4 CPU	1 285	2 285	1 480	2 480
Iris	2.0	0.48	0.94	1.49	2.96	4.68	2.91	8.02
New-thyroid	4.0	0.54	1.03	1.99	4.61	9.46	5.18	16.06
Ecoli	13.7	0.49	0.94	1.38	6.36	10.92	9.05	17.56
Contraceptive	26.6	1.29	2.52	3.47	31.43	55.29	50.18	93.64
Thyroid	103.0	0.60	1.15	2.31	37.88	69.66	75.70	155.86
Penbased	1434.1	1.15	2.26	4.37	111.85	207.99	191.67	391.61
Shuttle	1889.5	1.03	2.02	3.87	86.01	162.62	182.19	356.17
Connect-4	1778.5	1.09	2.14	3.87	116.46	223.82	201.57	392.86
KDDcup	154183.0	0.91	1.77	3.30	136.82	251.71	335.78	653.60
Poker	108831.6	1.25	2.46	4.69	209.61	401.77	416.30	820.18

Tabla 6.3: Resultados algoritmo de Bojarczuk et al.

Tiempo de ejecución (s)		Aceleración						
Dataset	Java	1 CPU	2 CPU	4 CPU	1 285	2 285	1 480	2 480
Iris	0.5	0.50	0.96	1.75	2.03	4.21	2.39	5.84
New-thyroid	0.8	0.51	1.02	1.43	2.99	5.85	3.19	9.45
Ecoli	1.3	0.52	1.02	1.15	3.80	8.05	5.59	11.39
Contraceptive	5.4	1.20	2.40	2.47	14.58	31.81	26.41	53.86
Thyroid	27.0	0.56	1.11	2.19	25.93	49.53	56.23	120.50
Penbased	42.6	0.96	1.92	3.81	18.55	36.51	68.01	147.55
Shuttle	222.5	1.18	2.35	4.66	34.08	67.84	117.85	253.13
Connect-4	298.9	0.69	1.35	2.65	42.60	84.92	106.14	214.73
KDDcup	3325.8	0.79	1.55	2.98	30.89	61.80	135.28	306.22
Poker	6527.2	1.18	2.32	4.45	39.02	77.87	185.81	399.85

Tabla 6.4: Resultados algoritmo de Tan et al.

Tiempo de ejecución (s)		Aceleración						
Dataset	Java	1 CPU	2 CPU	4 CPU	1 285	2 285	1 480	2 480
Iris	2.6	0.44	0.80	1.01	2.94	5.44	4.90	9.73
New-thyroid	6.0	0.77	1.43	1.78	7.13	12.03	9.15	21.74
Ecoli	22.5	0.60	1.16	2.09	9.33	16.26	14.18	25.92
Contraceptive	39.9	1.28	2.44	3.89	40.00	64.52	60.60	126.99
Thyroid	208.5	1.10	2.11	2.66	64.06	103.77	147.74	279.44
Penbased	917.1	1.15	2.23	4.25	86.58	148.24	177.78	343.24
Shuttle	3558.0	1.09	2.09	3.92	95.18	161.84	222.96	431.80
Connect-4	1920.6	1.35	2.62	4.91	123.56	213.59	249.81	478.83
KDDcup	185826.6	0.87	1.69	3.20	83.82	138.53	253.83	493.14
Poker	119070.4	1.27	2.46	4.66	158.76	268.69	374.66	701.41

Los resultados en las tablas proporcionan información útil. En algunos casos, la evaluación externa de la CPU es ineficiente para determinados conjuntos de datos como Iris, New-thyroid o Ecoli. Esto es porque el tiempo necesario para transferir los datos de la memoria de la máquina virtual Java a la memoria nativa es mayor que limitarse a hacer la evaluación en la máquina virtual de Java. Sin embargo, en todos los casos, independientemente del tamaño del conjunto de datos, la evaluación en GPU es siempre mucho más rápida. Si nos fijamos en los resultados de los conjuntos de datos más pequeños como Iris, New-thyroid y Ecoli, se puede ver que su aceleración es aceptable y, específicamente, Ecoli es hasta 25 veces más rápido. La aceleración de estos pequeños conjuntos de datos no sería muy útil, debido al poco tiempo necesario para su ejecución, pero vale la pena para conjuntos de datos mayores. Por otro lado, si nos centramos en conjuntos de datos complejos, la aceleración es mayor debido a que el modelo puede sacar el máximo provecho a los multiprocesadores de la GPU. Hay que tener en cuenta que KDDcup y Poker son conjuntos de datos acelerados hasta 653X y 820X más rápido, respectivamente. También podemos apreciar que la escalabilidad de la propuesta es casi perfecta, ya que duplicando el número de hilos o el número de tarjetas gráficas, el tiempo de ejecución se reduce casi a la mitad. La Figura 6.3. Resume el aumento de velocidad media, dependiendo del número de instancias.

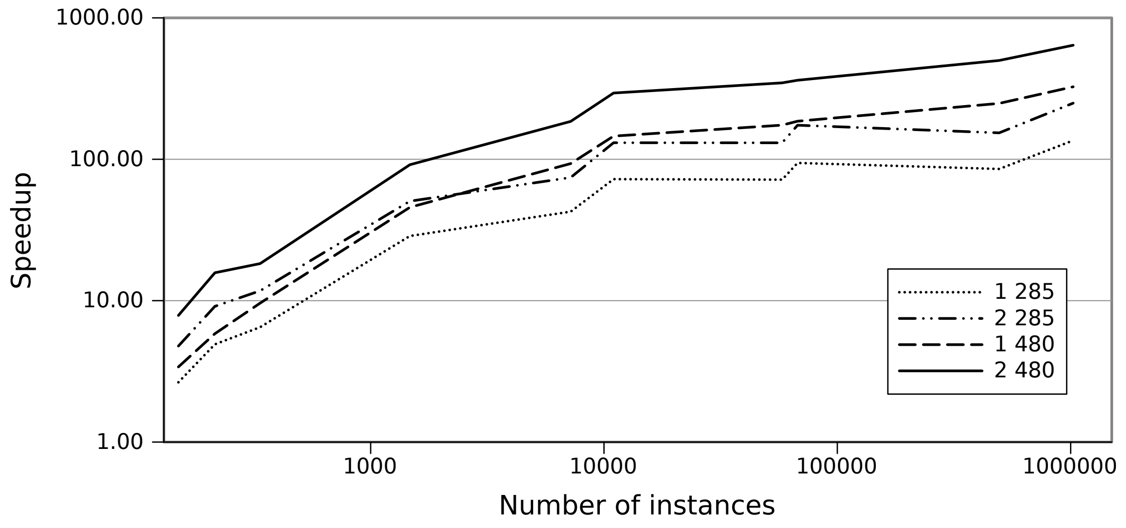


Figura 6.3: Aceleraciones medias

El hecho de obtener aceleraciones muy significativas en todos los dominios de problemas (conjuntos de datos pequeños y complejos) es porque nuestra propuesta es un modelo híbrido que aprovecha tanto la paralelización de los individuos como de las instancias. Un gran aumento de velocidad no sólo se logra mediante la clasificación de un gran número de casos, sino por una población suficientemente grande. La clasificación de los conjuntos de datos pequeños no requiere de muchos individuos, pero problemas de alta dimensionalidad por lo general requieren una gran población para proporcionar diversidad genética en la población. Por lo tanto, una aceleración grande se logra mediante la maximización de ambos parámetros. Estos resultados nos permiten determinar que el modelo propuesto alcanza una excelente aceleración en los algoritmos empleados. En concreto, la mejor aceleración es 820X cuando se utiliza el algoritmo de Falco et al. y el conjunto de datos de poker, en este caso el tiempo de ejecución es reducido de 30 horas a sólo dos minutos.

Resultados de BioHEL

Esta sección analiza los resultados obtenidos por BioHEL, que se muestran en la Tabla 6.5 donde la primera columna indica el tiempo de ejecución de la versión de serie expresado en segundos, la segunda columna muestra la aceleración de la versión de CUDA con una NVIDIA GTX 285 GPU y la tercera con una NVIDIA GTX 480 GPU. Estos resultados muestran que para un conjunto de datos con un bajo número de instancias, la versión de CUDA BioHEL es más lenta que la versión serie. Sin embargo, el aumento de velocidad obtenida es mayor cuando aumenta el número de instancias, logrando una aceleración de hasta 34.

Tabla 6.5: Resultados de BioHEL

Tiempo de ejecución (s)		Aceleración	
Dataset	Serial	1 285	1 480
Iris	0.5	0.64	0.66
New-thyroid	0.9	0.93	1.32
Ecoli	3.7	1.14	6.82
Contraceptive	3.3	3.48	3.94
Thyroid	26.4	2.76	8.70
Penbased	147.9	5.22	20.26
Shuttle	418.4	11.54	27.84
Connect-4	340.4	10.18	12.24
KDDcup	503.4	14.95	28.97
Poker	3290.9	11.93	34.02

Los resultados de la aceleración para el sistema BioHEL mostrados en la Tabla 6.5, en comparación con los resultados obtenidos por nuestra propuesta muestros en las Tablas 6.2, 6.3, y 6.4 demuestran la mayor eficiencia y aceleración de nuestro modelo. Una de las mejores ventajas de nuestra propuesta es que escala a múltiples GPUs mientras que BioHEL no lo permite. Ambos modelos utilizan un esquema de dos kernels. Sin embargo, el nuestro no realiza una reducción de primer nivel en el primer kernel, con el objetivo de evitar las esperas por sincronización de los hilos y retrasos innecesarios aunque esto signifique emplear mayor memoria para almacenar todos los resultados. Por lo tanto, los requisitos de memoria son mayores, pero la operación de reducción se realiza más rápido puesto que el acceso a memoria alineada es mucho más eficiente. Por otra parte, nuestra propuesta mejora la salida de instrucciones hasta 1,45, es decir, el número de instrucciones que pueden ejecutarse en una unidad de tiempo. Por lo tanto, nuestra propuesta alcanza un rendimiento de 1 Teraflops con dos GPUs NVIDIA GTX 480 con 480 núcleos a 700 MHz.

III

INTERPRETABILIDAD Y PRECISIÓN

7. INTERPRETABILIDAD VS PRECISIÓN

La tarea de clasificación se ha resuelto con éxito utilizando múltiples técnicas [36]. Por un lado, existen métodos como las redes neuronales artificiales (ANN) [66], clasificadores bayesianos [66], máquinas de vectores soporte (SVM) [11], y métodos de aprendizaje basados en las instancias [2]. Estas técnicas logran obtener muy buenos modelos de clasificación, pero son opacos. Los modelos predictivos opacos evitan que el usuario pueda seguir la lógica detrás de una predicción y la obtención de conocimiento interesante del modelo. Estos clasificadores no son directamente interpretables por un experto y no es posible descubrir cuáles son los atributos relevantes para predecir la clase de un ejemplo. Esta opacidad impide que se pueda utilizar en muchas aplicaciones reales donde se requiere tanto la exactitud y facilidad de comprensión, tales como el diagnóstico médico y la evaluación del riesgo de crédito [60].

Por otro lado, existen técnicas de aprendizaje automático que superan esta limitación y ofrecen clasificadores transparentes y comprensibles: como los árboles de decisión [86] y los sistemas basados en reglas [71]. Los Algoritmos Evolutivos [89], y concretamente los de Programación Evolutiva (PE) [88] y Programación Genética (PG) [33], se han aplicado con éxito para construir árboles de decisión y sistemas basados en reglas fácilmente. Estos clasificadores son muy interesantes porque son fáciles de interpretar y visualizables. Los sistemas basdos en reglas son especialmente fáciles de usar y comprender, y muchos trabajos se han centrado en la extracción de reglas a partir de modelos de ANN y SVM [7, 59, 60].

La clasificación con sistemas basados en reglas trata con dos objetivos contradictorios en el modelo obtenido: la interpretabilidad, la capacidad del comportamiento del sistema real de un modo comprensible, así como la exactitud, la capacidad para representar fielmente el sistema real. La obtención de un alto grado de interpretabilidad y exactitud es un objetivo contradictorio y, en la práctica, una de las dos propiedades prevalece sobre la otra. El encontrar el mejor equilibrio entre ellos es un problema de optimización que es muy difícil de resolver de manera eficiente. La interpretación de los modelos se está convirtiendo en un indicador cada vez más relevante, tan importante como la exactitud. Prueba de esta tendencia se encuentra en estudios recientes que abordan este problema [22, 41, 49, 50].

Por el contrario, en la búsqueda de interpretabilidad, los sistemas difusos son los habitualmente considerados [57]. Estos sistemas cuentan con reglas muy interpretables, ya que emplean variables lingüísticas para hacer frente a la vaguedad del lenguaje humano. Uno de estos sistemas es el algoritmo de González y Pérez, SLAVE (Structural Learning Algorithm on Vague Environment) [45], que es un algoritmo genético que utiliza un método iterativo para aprender reglas difusas. Sin embargo, esta definición de la interpretabilidad no está relacionada con la búsqueda de los clasificadores simples en términos de un menor número de reglas y condiciones, que es en el que estamos realmente interesados.

No existe una definición bien establecida de la interpretabilidad de un sistema basado en reglas. La interpretabilidad de un conjunto de reglas es muy importante, debido a que conjuntos de muchas reglas o reglas muy complejas carecen de interés. Sin embargo, hay algunos indicadores que nos permiten estimar la interpretabilidad y comprensibilidad de un clasificador basado en reglas, que son descritos por S. García et al. [41]: estos son el número de reglas y el número de condiciones. Una métrica para estimar la complejidad de un clasificador basado en reglas fue proporcionada por D. Nauck [63], quien propone una medida de interpretabilidad en relación con el número de condiciones del clasificador y el número de clases del conjunto de datos:

$$complexity = \frac{m}{\sum_{i=1}^{r} n_i}$$

donde m es el número de clases, r es el número de reglas, y n_i es el número de condiciones de la regla i-ésima Esta medida es 1 si el clasificador sólo contiene una regla por la clase con una condición cada una y se acerca a 0 cuanto más reglas y condiciones se utilizan. Sin embargo, esta medida puede ser de hasta 2 dado un conjunto de datos con dos clases donde el algoritmo aprende un clasificador que contiene una regla con una condición y otra regla por defecto y sin condiciones para la predicción de la clase por defecto.

8. ICRM. UN MODELO DE REGLAS INTERPRETABLES

En este capítulo se presenta un algoritmo de Programación Evolutiva para resolver problemas de clasificación de forma comprensible. Este algoritmo, llamado ICRM (Interpretable Classification Rule Mining) [18], está diseñado para obtener una base de reglas con el número mínimo de reglas y condiciones, con el fin de maximizar su interpretabilidad, mientras que se mantenga la obtención de resultados competitivos en la exactitud. El algoritmo cumple con algunas características deseables de un sistema basado en reglas interpretable. En primer lugar, garantiza la obtención del número mínimo de reglas. Esto es posible gracias a que cada regla predice una única clase y una única clase es predicha por una sola regla, excepto una última clase, considerada la clase predeterminada, que se asigna cuando ninguna de las reglas disponibles se disparan. En segundo lugar, no hay reglas totalmente contradictorias, es decir, no hay ningún par de reglas con los mismos antecedentes y consecuentes diferentes. Por otra parte, no hay redundancia, es decir, no existe una regla cuyo antecedente es también un subconjunto de los antecedentes de cualquier otra regla con el mismo consecuente. En tercer lugar, el orden de las reglas en el clasificador está implícito en el proceso de aprendizaje del algoritmo, de modo que el proceso evolutivo es directamente responsable de esta tarea. Por último, no sólo garantiza el número mínimo de reglas, sino que también busca el número mínimo de condiciones en los antecedentes de estas reglas, que se logra mediante la selección de las características más relevantes que discriminan la separación entre las clases y cada atributo se emplea una única vez en una regla.

Los experimentos llevados a cabo en 23 conjuntos de datos y 8 algoritmos muestran el desempeño competitivo de nuestra propuesta en términos de exactitud de la predicción y el tiempo de ejecución, mientras que logra obtener resultados significativamente mejores que los otros algoritmos en términos de todas las medidas de interpretabilidad consideradas: el número mínimo de reglas, el número mínimo de condiciones por regla, y el número mínimo de condiciones del clasificador.

Trabajos relacionados

Al considerar un sistema de aprendizaje basado en reglas, los diferentes métodos genéticos de aprendizaje siguen dos enfoques con el fin de codificar las reglas dentro de una población de individuos. El primero de ellos representa a un individuo como una sola regla, y el clasificador, conjunto de reglas, es proporcionado por la combinación de varios individuos de la población (cooperación) o por medio de diferentes procesos evolutivos (competición). El segundo representa a un individuo como un conjunto completo de reglas. La principal ventaja de este enfoque en comparación con el primero es que permite hacer frente al problema de la cooperación - competición, que implica la interacción entre las reglas en el proceso evolutivo [5, 65]. Un método basado en este enfoque es MPLCS (Memetic Pittsburgh Learning Classifier System) [6]. Sin embargo, su problema principal es controlar el número de reglas de los individuos, ya que el número total de reglas en la población puede crecer bastante, lo que aumenta su complejidad, el coste computacional y puede convertirse en un problema inmanejable.

Por lo tanto, en cualquier sistema basado en reglas que utiliza una lista de decisión se presenta el problema de decidir cúantas reglas son necesarias para resolver el problema y la forma en que deben ser priorizadas. Las respuestas a estas preguntas dependen de la codificación del sistema. Algunas propuestas, como la ILGA [46], fijan el número de reglas a priori, es decir, la complejidad en relación con el número de reglas es decidida por el usuario, pero la mayoría de las propuestas dejan el problema del número y el orden de las reglas a un algoritmo de aprendizaje automático. El orden de las reglas de una lista de decisión no es trivial y es muy importante, ya que se disparan y están priorizadas en un determinado orden.

Dos propuestas de Bojarczuk et al. [10] y Falco et al. [29] ordenan las reglas conforme al *fitness* obtenido en el proceso evolutivo. Además, se introduce un factor de simplicidad que penaliza el valor del *fitness* de una regla. Este factor depende de la simplicidad en el número actual de nodos (terminales y funciones) de la regla y del tamaño máximo permitido. El valor de *fitness* se calcula por medio de la exactitud y simplicidad. Sin embargo, esta función de *fitness* permite que un individuo con una menor exactitud se haga mejor por el simple hecho de ser más simple.

Más recientemente, algunos autores han utilizado los algoritmos evolutivos para ordenar las reglas a posteriori, como CO-Evolutionary Rule Extractor (CORE) algorithm [77] de Tan et al. en el que cada gen representa el índice de la regla y el cromosoma representa el orden de las reglas, con el objetivo de lograr mejores resultados. Por lo tanto, sería interesante no separar el proceso de obtención de buenas reglas de clasificación de su posición inherente en la lista de decisión.

Descripción del modelo

En esta sección se describen las características más relevantes y el modelo de ejecución del algoritmo propuesto, que consta de tres fases. En la primera fase, el algoritmo crea un conjunto de reglas que exploran los dominios de los atributos. En la segunda fase, el algoritmo itera para encontrar reglas de clasificación y construye el clasificador. Por último, la tercera fase optimiza la exactitud del clasificador. La Figura 8.1. Muestra el flujo de trabajo del algoritmo. Las tres secciones siguientes describen las fases del algoritmo.

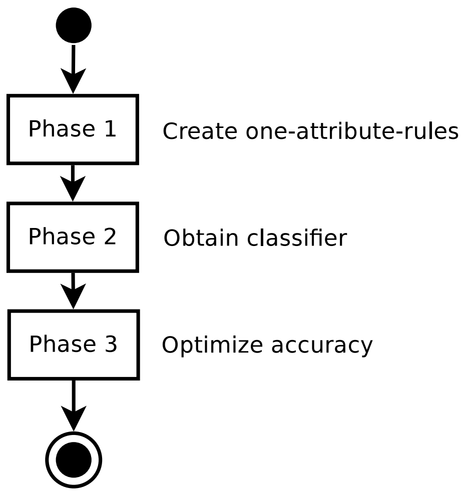


Figura 8.1: Flujo del Algoritmo ICRM

Fase 1: exploración de los dominios de los atributos

Esta fase explora los dominios de atributos, cuya salida es un conjunto de reglas compuesta por una única comparación atributo–valor. La idea es encontrar el mejor atributo–valor de las posibles comparaciones para clasificar cada una de las clases. La Figura 8.2 muestra la gramática usada para representar la comparación atributo–valor.

$\langle S \rangle \rightarrow \langle cmp \rangle$
$\langle cmp \rangle \rightarrow \langle op\_num \rangle\ \langle variable \rangle\ \langle valor \rangle$
$\langle cmp \rangle \rightarrow \langle op\_cat \rangle\ \langle variable \rangle\ \langle valor \rangle$
$\langle op\_num \rangle \rightarrow\ \geq\ |\ \leq$
$\langle op\_cat \rangle \rightarrow\ =\ |\ \neq$
$\langle variable \rangle \rightarrow$ *Cualquier atributo del conjunto de datos*
$\langle valor \rangle \rightarrow$ *Cualquier valor del dominio del atributo*

Figura 8.2: Gramática empleada para crear las comparaciones atributo–valor

El objetivo es garantizar la exploración de los dominios de todos los atributos. Los dominios pueden ser categóricos o numéricos. Por un lado, los dominios categóricos están limitados a un número cerrado de etiquetas o valores nominales. Por lo tanto, la búsqueda de una etiqueta correspondiente que mejor clasifica a una clase es simple y se trata de un problema de combinatoria con un número relativamente bajo de las soluciones. El algoritmo crea una correspondecia atributo–etiqueta para cada etiqueta y operador de relación nominal.

Por otro lado, los dominios numéricos cuentan con infinitas soluciones. Por lo tanto, un método iterativo para explorar los dominios numéricos se describirá a continuación. La idea es explorar inicialmente el rango del dominio numérico en varios puntos distribuidos uniformemente (cuyos valores se optimizan en la Fase 2). El algoritmo genera una comparación atributo–valor para cada uno de estos puntos y cada uno de los operadores relacionales numéricos. Todas estas reglas se almacenan y se van a utilizar en la Fase 2. El pseudo-código del procedimiento de creación de estas comparaciones atributo-valoradas se muestra en el Algoritmo 2.

Fase 2: obtención del clasificador

El objetivo de esta fase es la obtención de reglas de clasificación interpretables y precisas. La entrada de esta fase es el conjunto comparaciones atributo–valor creadas en la primera fase y el resultado es un conjunto de reglas de clasificación en forma de clasificador. La Figura 8.3 muestra el flujo de trabajo de esta fase. Las siguientes secciones detallan el procedimiento para obtener las reglas, la codificación de los individuos, el operador genético empleado, y la función de *fitness* que evaluará la aptitud de los individuos.

Algorithm 2 Procedimiento de creación de las comparaciones atributo–valor

```
1:  for $i = 1$ to $numberAttributes$ do
2:      if attribute(i) is numerical then
3:          $step \leftarrow$ domain(i) / numberPoints
4:          $left \leftarrow$ minValueDomain(i)
5:          for $j = 1$ to numberPoints-1 do
6:              $value \leftarrow$ left + j*step
7:              createRule($\geq$ attribute(i) value)
8:              createRule($\leq$ attribute(i) value)
9:          end for
10:     else if attribute(i) is categorical then
11:         for all $label$ in $domain(i)$ do
12:             createRule($=$ attribute(i) label)
13:             createRule($\neq$ attribute(i) label)
14:         end for
15:     end if
16: end for
```

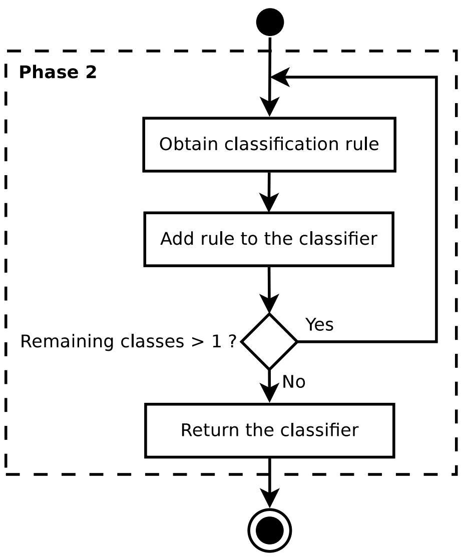


Figura 8.3: Flujo de la Fase 2

1) Obtener reglas de clasificación

En esta sección se describe el proceso de obtención de las reglas de clasificación. El objetivo de la Fase 2 es obtener una base completa de reglas con tantas reglas como clases en donde cada regla predice una única clase y cada clase es predicha por una sola regla. En cada iteración de este procedimiento se obtiene una regla de clasificación, que predecirá a una clase aún no cubierta y que es la clase más fácil de clasificar entre las restantes. Para ello, se desarrolla en paralelo tantos procesos evolutivos como clases por cubrir. La salida de cada proceso evolutivo es una regla de clasificación que predice a una clase única. La mejor regla de entre todas las salidas de los procesos evolutivos se selecciona y se añade al clasificador (base de reglas). La clase predicha por el consecuente de esta regla se marca como cubierta y el algoritmo elimina las instancias de dicha clase del conjunto de entrenamiento. Este proceso se repite tantas veces como número de clases menos una. La última clase se configura como la clase predeterminada del clasificador, clase por defecto. La Figura 8.4 muestra el flujo de trabajo para obtener las reglas de clasificación, donde K es el número de clases aún no cubiertas.

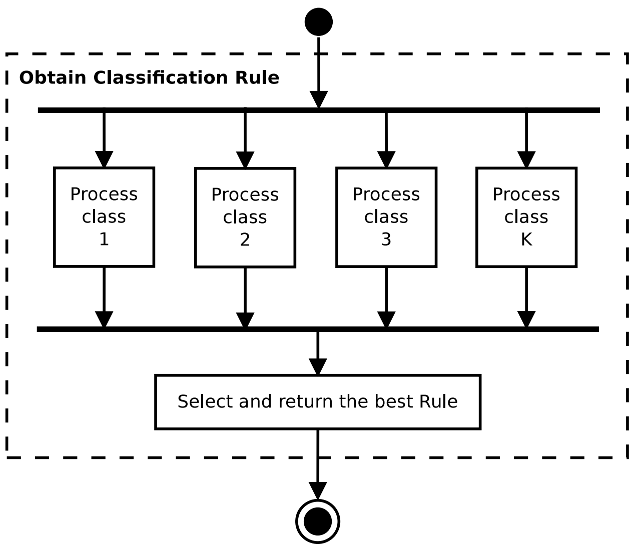


Figura 8.4: Obtención de las reglas de clasificación

Este procedimiento iterativo implica evolucionar tantos algoritmos evolutivos como número de clases por cubrir, es decir, que para un conjunto de datos con N clases, la primera iteración evoluciona N algoritmos evolutivos, la segunda, $N - 1$, la tercera, $N - 2$, y así sucesivamente, hasta que sólo hay una clase por cubrir. Puede parecer que esto requiere un alto costo computacional. Sin embargo, en cada iteración el número de clases para discernir y el número de instancias es menor que en la iteración anterior, reduciendo la dimensionalidad del problema (número de instancias y clases por cubrir). En consecuencia, cada iteración se ejecuta más rápido que la anterior. Los resultados experimentales y los tiempos de ejecución mostrados en la sección 8.4.6 demuestran la gran eficiencia del algoritmo.

Las siguientes secciones describen la representación del individuo, la descripción del algoritmo, el operador genético y la función de aptitud del algoritmo evolutivo empleado para obtener las reglas de clasificación.

2) Representación del individuo

La Fase 2 utiliza una representación individuo = regla para aprender reglas interpretables SI-ENTONCES que se insertan en una lista de decisión para construir un clasificador. Esta representación proporciona una mayor eficiencia, y aborda el problema de cooperación–competición al tratar con el orden de las reglas en el proceso evolutivo. El antecedente de las reglas representa un conjunto de comparaciones atributo–valor y el consecuente contiene la clase predicha para una instancia que cumpla el antecedente de la regla. La Figura 8.5 muestra la gramática utilizada para representar las reglas. La derivación $\langle S \rangle$ de la gramática es el símbolo inicial de la gramática de la Figura 8.2.

$$\langle S' \rangle \rightarrow \langle S \rangle \mid \langle S' \rangle \; AND \; \langle S \rangle$$

Figura 8.5: Gramática para la representación de los individuos

3) Generación evolutiva de reglas

El proceso de aprendizaje del algoritmo funciona como un algoritmo evolutivo generacional y elitista. El proceso evolutivo encontrará la conjunción de comparaciones atributo–valor sobre los atributos relevantes para distinguir una clase de las otras, por medio del operador genético. Para cada regla, el operador genético emplea los atributos aún no cubiertos por la regla para encontrar la mejor condición sobre cualquier otro atributo que, añadido a la regla, mejore su exactitud. Por lo tanto, el número máximo de generaciones se establece como el número de atributos. La supervivencia de las mejores reglas está garantizada por la copia a la siguiente generación. La Figura 8.6 muestra el flujo de trabajo del algoritmo evolutivo.

El operador genético añade nuevas cláusulas (condiciones) a la regla. Cuando una nueva condición se agrega, el área de búsqueda y el número de instancias cubiertas por la regla se reducen. Así, en lugar de buscar nuevas condiciones en el conjunto de entrenamiento completo, el operador genético se centra en la búsqueda de los atributos que ahora sean verdaderamente relevantes en el subconjunto de entrenamiento de las instancias cubiertos por la regla. El operador genético devolverá una condición para que se adjunte a la regla, creando un nuevo individuo. Si el operador genético no encuentra una condición sobre un atributo relevante que mejore la exactitud de la regla, el individuo se marca como inmejorable. Además, cuando una regla ya ha sido valorada sobre todos los atributos también se marca como inmejorable.

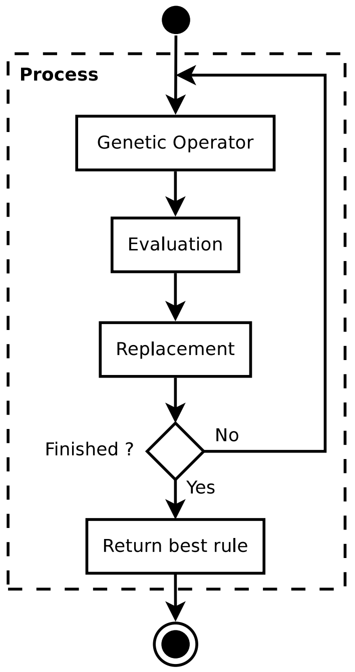


Figura 8.6: Flujo de la generación de reglas

El método de evaluación calcula el valor de *fitness* para cada individuo creado por el operador genético. Cuanto mayor sea su *fitness*, mejor clasifica la regla.

Finalmente, el método de reemplazo selecciona a los mejores individuos de la generación anterior y los hijos como la población para la próxima generación, manteniendo el tamaño de la población. El algoritmo termina cuando el número máximo de generaciones se alcanza o todos los individuos de la población están marcados como inmejorables. Por último, cuando el algoritmo termina, se devuelve el mejor individuo que maximice la función de aptitud y cuya regla esté constituida por comparaciones sobre atributos verdaderamente relevantes.

4) Operador genético

Esta sección describe el operador genético diseñado para encontrar las condiciones sobre los atributos relevantes que, añadidas a una regla ya existente, mejoren su clasificación. Los parámetros de este operador incluyen el conjunto de comparaciones atributo–valor creado en la inicialización, la clase a predecir, y el subconjunto de instancias cubiertas por la regla actual. El operador ejecuta un algoritmo de escalada microgenético (hill-climbing microgenetic algorithm, MGA) [51] para optimizar los valores numéricos de las comparaciones atributo–valor sobre el subconjunto de instancias. El operador devuelve una nueva comparación atributo–valor que mejore la predicción de la clase del consecuente. Esta nueva comparación se añade al antecedente de la regla actual, creando una nueva regla.

La optimización de los valores numéricos de las comparaciones atributo–valor se trata como un problema de optimización de una función continua real. El MGA utiliza un operador de mutación para aproximar los valores numéricos de las reglas al máximo de la función. Cada regla transforma el valor de la comparación dentro de un rango cercano [-*step*, +*step*] usando el paso *step* descrito en el Algoritmo 2. El procedimiento de selección es responsable de asegurar la supervivencia de los mejores individuos y después de un pequeño número de generaciones, los individuos convergen en el valor que mejor clasifica la clase dada.

5) Función de ajuste

Los resultados de la comparación de la clase predicha por las reglas y la clase verdadera de las instancias se emplean para construir la matriz de confusión, que es una tabla con dos filas y dos columnas, que indica el número de verdaderos positivos (T_P), verdaderos negativos (T_N), los falsos positivos (F_P) y los falsos negativos (F_N). Empleando estos valores, se construye la matriz de confusión y se obtienen dos indicadores, sensibilidad (Se) y especificidad (Sp). La sensibilidad, también llamada recall en algunos campos, mide la proporción de verdaderos positivos que son correctamente identificados como tales, mientras que la especificidad mide la proporción de los negativos que se han identificado correctamente.

$$Se = \frac{T_P}{T_P + F_N} \qquad Sp = \frac{T_N}{T_N + F_P}$$

El valor de *fitness* se calcula por medio de la sensibilidad y especificidad con el fin de maximizar ambas métricas. Además, para asegurar la interpretabilidad de las reglas representadas por los individuos, para reglas igualmente precisas, la más sencilla con el número más bajo de condiciones, es la que prevalece.

$$Fitness = Se * Sp$$

Fase 3: optimización de la exactitud

En esta sección se describe el proceso evolutivo para la optimización del clasificador obtenido en la Fase 2. Las reglas del clasificador se han seleccionado y optimizado de forma individual para maximizar el producto de la sensibilidad y especificidad. Sin embargo, los valores de las comparaciones numéricas de las reglas pueden ser un poco ajustados para mejorar la exactitud global del clasificador. Por lo tanto, la Fase 3 ejecuta un proceso evolutivo que mejora la exactitud del clasificador.

La Fase 3 utiliza una representación individuo = clasificador. La población se inicializa utilizando el clasificador proporcionado por la Fase 2 como semilla. El algoritmo itera la mutación de las reglas de los clasificadores y selecciona los mejores clasificadores generados hasta que un cierto número de generaciones se han realizado. El tamaño de la población y el número máximo de generaciones son parámetros especificados en la sección 8.3.3.

El operador de mutación se aplica a todos los individuos de la población. Dado un individuo padre, se crean tantos nuevos clasificadores como reglas contiene, ya que para cada regla se va a generar un nuevo clasificador hijo con dicha regla mutada, con el fin de ajustar las reglas para mejorar el rendimiento global del clasificador. La mutación de una regla funciona como una búsqueda local y ajusta los valores de las comparaciones atributo–valor dentro de un rango cercano [-*step*, *+step*] usando el paso *step* descrito en el Algoritmo 2, similar al MGA que se describe en el operador genético de la Fase 2.

La función de aptitud calcula la calidad de los individuos de la población. En la fase 3, la aptitud de un individuo es la exactitud del clasificador, ya que todos los individuos tienen la misma complejidad. La exactitud es la relación entre el número de instancias clasificadas correctamente frente al número total de instancias.

Estudio experimental del modelo ICRM

Esta sección describe los detalles de los experimentos realizados en diferentes conjuntos de datos para evaluar las capacidades de la propuesta y compararla con otros métodos de clasificación.

Los experimentos llevados a cabo comparan los resultados de 8 algoritmos de clasificación con más de 23 conjuntos de datos. Estos conjuntos de datos han sido seleccionados de la página web del repositorio de KEEL [3] y los algoritmos están disponibles en la herramienta de software KEEL [4]. Los conjuntos de datos junto con sus particiones están disponibles para facilitar las comparaciones futuras con nuevas técnicas.

En primer lugar, se presentan los conjuntos de datos y los algoritmos utilizados para la comparación. En segundo lugar, se lleva a cabo un análisis de los parámetros del algoritmo ICRM con el fin de encontrar una combinación adecuada de los parámetros. En tercer lugar, se proporciona una breve descripción de la metodología experimental y métricas calculadas. Finalmente, se presentan las pruebas estadísticas utilizadas para la validación de los resultados.

Tabla 8.1: Complejidad de los conjuntos de datos

Data set	#Instances	#Attributes	#Classes
Appendicitis	106	7	2
Australian	690	14	2
Bupa	345	6	2
Contraceptive	1473	9	3
Dermatology	366	34	6
Ecoli	336	7	8
Flare	1066	11	6
Glass	214	9	7
Haberman	306	3	2
Heart	270	13	2
Iris	150	4	3
Lymphography	148	18	4
Monk-2	432	6	2
New-thyroid	215	5	3
Page-blocks	5472	10	5
Pima	768	8	2
Saheart	462	9	2
Sonar	208	60	2
Tae	151	5	3
Thyroid	7200	21	3
Vehicle	846	18	4
Wine	178	13	3
Zoo	101	16	7

Conjuntos de datos

Los conjuntos de datos utilizados en los experimentos han sido seleccionados de la página web de repositorio de KEEL [3] y son muy variados en su grado de complejidad, el número de clases, el número de atributos, y el número de instancias. El número de clases es de hasta 8, el número de atributos entre 3 y 60, y el número de instancias varía de 101 a 7.200. La Tabla 8.1 resume la información acerca de estos conjuntos de datos. Los conjuntos de datos se dividen en 10 particiones siguiendo el método de validación cruzada 10-fold cross validation [53, 82] y están disponibles para facilitar las comparaciones futuras con otros métodos.

Algoritmos de clasificación

En esta sección se describen los algoritmos utilizados en nuestros experimentos, que se obtienen a partir de la herramienta de software KEEL [4]. Algunos de los métodos más relevantes presentados hasta la fecha se comparan para determinar si nuestro algoritmo evolutivo es competitivo en los distintos conjuntos de datos. Cuatro de los siete métodos fueron presentados en la sección de trabajos relacionados (MPLCS, ILGA, CORE, SLAVE), pero también consideramos interesante comparar con otros tres métodos bien conocidos, incluyendo un modelo no evolutivo (RIPPER), el clásico árbol de decisión (C4.5) del que extraer reglas (C45R), y un método de colonias de hormigas (AntMiner+).

- MPLCS [6]: Memetic Pittsburgh Learning Classifier System que combina operadores de búsqueda local.
- ILGA [46]: Incremental Learning approach to Genetic Algorithms con diferentes esquemas de inicialización basados en diferentes inicializaciones de la población del algoritmo genético.
- CORE [77]: COevolutionary Rule Extractor que coevoluciona reglas y bases de reglas concurrentemente en dos poblaciones que cooperan para producir buenos clasificadores.
- Ant Miner+ [67]: algoritmo de minería de reglas inspirado en el comportamiento de las colonias de hormigas.
- SLAVE [45]: Structural Learning Algorithm on Vague Environment es un algoritmo genético que usa una aproximación iterativa para aprender reglas difusas.
- RIPPER [25]: mejora del eficiente algoritmo IREP (incremental reduced error pruning) algorithm.
- C45R [70]: C45Rules genera un conjunto de reglas a partir del árbol de decisión producido por C4.5.

Los autores de los modelos establecieron en sus respectivos trabajos diferentes configuraciones de los parámetros de sus métodos. Los parámetros que nosotros hemos empleado en el estudio comparativo son los parámetros óptimos encontrados por los autores en sus respectivos estudios experimentales.

Parámetros del modelo ICRM

En esta sección se analiza la configuración de los parámetros ICRM para determinar la influencia de cada parámetro y sus combinaciones en términos de exactitud y coste computacional. ICRM se ha implementado en el software JCLEC [80] y sus principales parámetros se muestran en la Tabla 8.2. Las tres fases diferentes del algoritmo tiene algunos parámetros cuyos valores óptimos se muestran en la Tabla 8.2. La Fase 1 requiere del número de puntos en los que los dominios son inicialmente explorados. La Fase 2 requiere un valor de presión selectiva evaluado en (0,1], el cual determina el número de atributos relevantes para explorar en relación con el número de atributos. El operador genético de la fase 2 requiere dos parámetros: el tamaño de la población y el número de generaciones del MGA que optimiza las reglas. La Fase 3 también requiere dos parámetros: el tamaño de la población y el número de generaciones del algoritmo que optimiza el clasificador final. El efecto de estos parámetros no tiene ninguna relevancia en los resultados de la exactitud, pero es importante con respecto al tiempo de cómputo, que aumenta considerablemente conforme el tamaño de la población o el número de generaciones aumenta.

Tabla 8.2: Parámetros del algoritmo ICRM

	Parameter	Value
Fase 1	Número de puntos	10
	Presión selectiva	0.5
Fase 2	Población MGA	10
	Generaciones MGA	10
Fase 3	Población GA	5
	Generaciones GA	100

Configuración de la experimentación

Los experimentos consideran las siguientes métricas de evaluación de los clasificadores: la exactitud de predicción, el número de reglas del clasificador, el número de condiciones por regla, el número de condiciones total del clasificador, la medida de la complejidad descrita en la sección 7, y el tiempo de ejecución. Todos los experimentos se han repetido con cinco semillas diferentes para los métodos estocásticos, y los resultados medios son los que se muestran en las tablas. Todos los algoritmos se han ejecutado sobre los conjuntos de datos empleando validación cruzada 10-fold cross validation. Los experimentos se han ejecudado en un PC con un procesador Intel core i7 a 2.66 GHz, 12 GB DDR3 de memoria y dos gráficas NVIDIA GTX 480. El sistema operativo era GNU/Linux Ubuntu 11.04 64 bit.

Análisis estadístico

Para poder analizar los resultados de los experimentos es necesario realizar algunos tests estadísticos no paramétricos con el objetivo de validar los resultados y las conclusiones [42, 43]. Para evaluar si existen diferencias significativas en los resultados de los distintos algoritmos, primero se realizar el test de Iman y Davenport. Este útil test no paramétrico fue recomendado por Demsar [30], y se aplica para obtener el ranking de los K algoritmos sobre los N conjuntos de datos (en nuestro caso hay 8 algoritmos y 23 conjuntos de datos) conforme a una F-distribución. Cuando el test de Iman y Davenport indica que existen diferencias significativas, es necesario realiar un test post hoc, en nuestro caso, el test de Bonferroni–Dunn [31] se emplea para encontrar las diferencias significativas entre los algoritmos de la compartiva múltiple. El test asume que los resultados de dos clasificadores es significativamente distinto si difieren en su ranking al menos un cierto valor que se denomina distancia crítica. Finalmente, el test de pares de Wilcoxon [83] se emplea para evaluar múltiples comparaciones para cada par de algoritmos.

Resultados del modelo ICRM

En esta sección se analizan los resultados de los experimentos y se compara nuestro modelo frente a los otros algoritmos. Con el fin de demostrar la eficacia y la eficiencia de nuestro modelo, la exactitud, el tiempo de ejecución, y la interpretabilidad del clasificador serán evaluados. Aunque los problemas acerca de la interpretabilidad de los clasificadores han estado recibiendo mucha atención en los últimos años, no existe una definición bien establecida para evaluar la interpretabilidad de una base de reglas. En este trabajo, se analizan diferentes definiciones para medir la interpretabilidad y se comparan con el resto de las propuestas.

Exactitud

La exactitud determina el ratio del número de instancias correctamente clasificadas frente al número total de instancias. Obviamente, la exactitud de la clasificación es uno de los factores más importantes a la hora de diseñar modelos de clasificación, puesto que se desea obtener clasificadores tan precisos como sea posible a la hora de realizar las predicciones.

Tabla 8.3: Exactitud

Exactitud	ICRM	C45R	RIPPER	MPLCS	AntMiner+	CORE	ILGA	SLAVE
Appendicitis	84.00 %	83.18 %	78.70 %	**86.36 %**	85.70 %	84.76 %	85.76 %	83.88 %
Australian	85.51 %	85.17 %	83.00 %	**86.23 %**	85.65 %	84.15 %	84.98 %	70.63 %
Bupa	59.61 %	**65.05 %**	62.12 %	62.46 %	41.54 %	59.12 %	55.40 %	58.77 %
Contraceptive	51.19 %	51.14 %	51.69 %	**54.84 %**	46.26 %	46.28 %	44.17 %	43.68 %
Dermatology	**96.35 %**	94.36 %	93.00 %	95.61 %	87.10 %	35.94 %	58.77 %	91.65 %
Ecoli	77.74 %	77.67 %	74.82 %	80.89 %	74.09 %	65.49 %	63.52 %	**84.44 %**
Flare	68.57 %	68.39 %	67.20 %	**72.96 %**	70.49 %	65.92 %	62.73 %	0.00 %
Glass	**68.55 %**	68.05 %	63.97 %	65.41 %	49.36 %	50.26 %	50.13 %	59.61 %
Haberman	**74.59 %**	71.19 %	47.82 %	73.49 %	70.49 %	73.94 %	72.00 %	71.78 %
Heart	74.81 %	79.01 %	75.68 %	**81.36 %**	80.37 %	71.98 %	65.43 %	79.75 %
Iris	**97.07 %**	96.67 %	93.78 %	95.78 %	93.56 %	94.67 %	92.89 %	95.33 %
Lymphography	80.31 %	73.60 %	74.75 %	**80.74 %**	74.20 %	63.32 %	77.69 %	68.75 %
Monk-2	97.27 %	**100.00 %**	**100.00 %**	99.55 %	97.27 %	92.84 %	54.37 %	97.27 %
New-thyroid	92.75 %	**93.53 %**	93.20 %	92.42 %	89.16 %	91.36 %	91.66 %	90.13 %
Page-blocks	95.02 %	95.51 %	**96.46 %**	94.34 %	93.76 %	90.38 %	92.78 %	93.55 %
Pima	74.11 %	71.83 %	69.23 %	**74.83 %**	72.10 %	72.97 %	73.36 %	73.37 %
Saheart	**69.19 %**	68.04 %	60.02 %	68.70 %	67.95 %	68.83 %	67.30 %	64.06 %
Sonar	69.39 %	70.55 %	75.28 %	**77.33 %**	74.63 %	53.38 %	70.80 %	73.63 %
Tae	49.18 %	45.49 %	49.29 %	**57.06 %**	33.12 %	45.00 %	42.51 %	47.47 %
Thyroid	98.79 %	**99.60 %**	99.47 %	94.57 %	98.26 %	75.89 %	94.05 %	93.04 %
Vehicle	63.12 %	66.66 %	69.15 %	**70.53 %**	63.32 %	38.53 %	58.53 %	63.69 %
Wine	93.56 %	**94.90 %**	93.77 %	92.27 %	86.90 %	94.39 %	88.93 %	92.11 %
Zoo	95.83 %	92.81 %	92.69 %	**96.56 %**	49.90 %	93.25 %	85.87 %	0.00 %
Promedio	78.98 %	78.80 %	76.74 %	**80.62 %**	73.27 %	70.12 %	71.03 %	69.42 %
Ranking	3.13	3.59	4.24	**2.17**	5.48	5.74	6.13	5.52

La Tabla 8.3 muestra la exactitud media de los resultados para las particiones de test y el ranking de los algoritmos. Los mejores valores de exactitud, con un valor de ranking de 2.17, se han obtenido con el algoritmo MPLCS. Sin embargo, los resultados obtenidos por nuestra propuesta sólo difieren un 1.6 % menos de media con un ranking de 3.13, que es el segundo mejor valor. El estadístico de Iman y Davenport para la exactitud distribuído conforme a la distribución F con $K - 1 = 7$ y $(K - 1)(N - 1) = 154$ grados de libertad es 11.457. El test establece un valor de distribución de 2.757 para un nivel de confianza con alpha igual a 0.01. Este valor es menor que el valor del estadístico obtenido 11.457. Por lo tanto, el test rechaza la hipótesis nula y por lo tanto se puede decir que existen diferencias significativas entre los resultados de la exactitud de los algoritmos comparados.

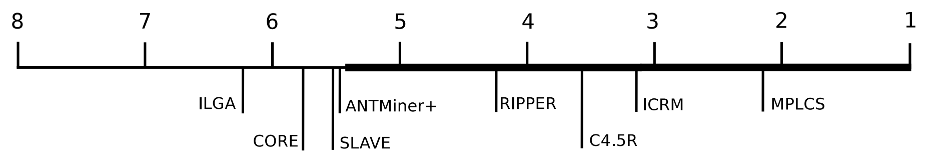


Figura 8.7: Test de Bonferroni–Dunn para la exactitud

La Figura 8.7 muestra la aplicación del test de Bonferroni–Dunn para la exactitud con un alpha igual a 0.01, cuya diferencia crítica es 2.303. Este gráfico representa los algoritmos y sus rankings a lo largo de la recta. El valor de diferencia crítica se representa mediante una línea horizontal gruesa y aquellos valores fuera del rango crítico son algoritmos que obtienen resultados estadísticamente significativamente diferentes que el algoritmo de control, en este caso, nuestra propuesta ICRM. Por lo tanto, los algoritmos a la izquierda de la diferencia crítica son algoritmos significativamente peores que ICRM, y aquellos a la derecha de la diferencia crítica obtienen resultados significativamente mejores, dado un grado de confianza. Observando esta figura, no existe ningún algoritmo de la comparativa significativamente mejor que ICRM con respecto a la exactitud, mientras que AntMiner+, CORE, ILGA, y SLAVE obtienen clasificadores con una exactitud significativamente peor.

La Tabla 8.4 muestra los resultados del test de Wilcoxon para la exactitud, realizando múltiples comparaciones por pares entre nuestra propuesta y los otros métodos. Los p-values devueltos por el test indican que no existen diferencias significativas cuando se compara ICRM contra C45R y MPLCS. Sin embargo, estas diferencias sí son significativas cuando se compara con AntMiner+, CORE, ILGA y SLAVE, alcanzando p-values menores de 0.01, es decir, con una confianza superior al 99 %.

Tabla 8.4: Test de Wilcoxon para la exactitud

ICRM vs	R^+	R^-	p-value
C45R	150.0	126.0	≥ 0.2
RIPPER	185.5	90.5	0.155590
MPLCS	79.0	197.0	≥ 0.2
AntMiner+	208.0	45.0	0.006650
CORE	267.0	9.0	7.868E-6
ILGA	267.0	9.0	7.868E-6
SLAVE	211.0	42.0	0.004684

Número de reglas

El número de reglas determina la complejidad del modelo. Cuanto mayor sea el número de reglas, mayor probabilidad de conflictos entre ellas (contradicciones y solapamiento) y mayor es la dificultad para entender las condiciones necesarias para predecir la clase de una instancia determinada.

Tabla 8.5: Número de reglas

# Reglas	ICRM	C45R	RIPPER	MPLCS	AntMiner+	CORE	ILGA	SLAVE
Appendicitis	**2.00**	3.20	7.30	4.13	2.10	3.63	30.00	4.83
Australian	**2.00**	11.07	23.53	4.83	5.50	4.03	30.00	6.97
Bupa	**2.00**	8.57	23.80	6.60	**2.00**	3.43	30.00	6.30
Contraceptive	**3.00**	31.00	64.60	5.67	3.13	8.93	30.00	44.90
Dermatology	**6.00**	9.20	14.27	7.77	6.70	6.47	30.00	10.57
Ecoli	8.00	11.40	35.50	8.90	7.63	**6.33**	30.00	12.77
Flare	6.00	27.90	101.60	16.67	20.30	**4.80**	30.00	0.00
Glass	7.00	10.07	22.20	8.27	**4.57**	7.03	30.00	16.27
Haberman	**2.00**	4.30	17.67	4.37	**2.00**	4.13	30.00	5.63
Heart	**2.00**	10.83	13.97	6.90	4.87	5.47	30.00	7.47
Iris	3.00	5.00	6.30	4.07	**2.97**	3.63	30.00	3.00
Lymphography	4.00	11.60	13.10	7.57	**3.70**	6.43	30.00	4.37
Monk-2	**2.00**	6.00	4.00	4.00	3.00	3.57	30.00	3.00
New-thyroid	**3.00**	6.80	6.93	5.43	3.57	3.30	30.00	5.50
Page-blocks	**5.00**	22.50	51.23	7.10	14.70	5.43	30.00	9.90
Pima	**2.00**	8.40	25.10	5.43	12.30	3.13	30.00	8.57
Saheart	**2.00**	6.70	24.57	6.23	3.70	6.37	30.00	11.30
Sonar	**2.00**	8.70	8.13	6.17	4.83	1.00	30.00	8.37
Tae	3.00	7.57	28.43	5.57	**2.00**	6.47	30.00	11.03
Thyroid	**3.00**	10.33	11.43	4.03	24.17	3.97	30.00	6.93
Vehicle	**4.00**	19.80	46.70	15.17	25.97	6.77	30.00	34.77
Wine	**3.00**	5.00	5.47	4.47	3.77	3.07	30.00	4.03
Zoo	7.00	8.70	9.10	7.00	**4.20**	7.03	30.00	0.00
Promedio	**3.61**	11.07	24.56	6.80	7.29	4.98	30.00	9.85
Ranking	**1.48**	5.35	6.93	3.87	2.76	2.74	7.57	5.30

La Tabla 8.5 muestra el número medio de reglas de los clasificadores obtenidos y el ranking de los algoritmos. Los mejores valores con un menor número de reglas son para nuestra propuesta, con un valor de ranking de 1.48, seguido por CORE y AntMiner+. El estadístico de Iman y Davenport para el número de reglas distribuído conforme a la distribución F con $K - 1 = 7$ y $(K - 1)(N - 1) = 154$ grados de libertad es 67.253. El test establece un valor de distribución de 2.757 para un nivel de confianza con alpha igual a 0.01. Este valor es menor que el valor del estadístico obtenido 67.253. Por lo tanto, el test rechaza la hipótesis nula y por lo tanto se puede decir que existen diferencias significativas entre el número de reglas de los clasificadores obtenidos por los algoritmos comparados.

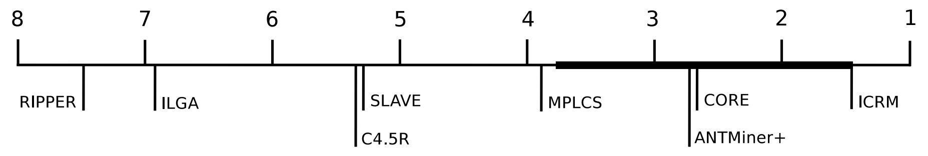


Figura 8.8: Test de Bonferroni–Dunn para el número de reglas

La Figura 8.8 muestra la aplicación del test de Bonferroni–Dunn para el número de reglas con un alpha igual a 0.01, cuya diferencia crítica es 2.303. Los algoritmos a la izquierda de la diferencia crítica son algoritmos significativamente peores que ICRM con una confianza mayor del 99 %. Observando esta figura, todos los algoritmos excepto ANTMiner+ y CORE obtienen resultados significativamente peores que ICRM con respecto al número de reglas.

La Tabla 8.6 muestra los resultados del test de Wilcoxon para el número de reglas, realizando múltiples comparaciones por pares entre nuestra propuesta y los otros métodos. Los p-values devueltos por el test indican que existen diferencias significativas cuando se compara ICRM contra todos los demás algoritmos excepto para AntMiner+, cuyo p-value es de 0.0319.

Tabla 8.6: Test de Wilcoxon para el número de reglas

ICRM vs	R^+	R^-	p-value
C45R	276.0	0.0	2.384E-7
RIPPER	276.0	0.0	2.384E-7
MPLCS	253.0	0.0	4.768E-7
AntMiner+	222.0	54.0	0.031920
CORE	238.0	38.0	0.001458
ILGA	276.0	0.0	2.384E-7
SLAVE	253.0	0.0	4.768E-7

Condiciones por regla

El número de condiciones por regla determina la longitud y complejidad de una regla. Cuanto mayor sea el número de condiciones, menor es su comprensibilidad.

La Tabla 8.7 muestra el número medio de condiciones por regla de los clasificadores obtenidos y el ranking de los algoritmos. Los mejores valores con un menor número de condiciones por regla son para nuestra propuesta, con un valor de ranking de 1.20, seguido de lejos por C45R. El estadístico de Iman y Davenport para el número medio de condiciones por regla distribuído conforme a la distribución F con $K-1=7$ y $(K-1)(N-1)=154$ grados de libertad es 18.501. El test establece un valor de distribución de 2.757 para un nivel de confianza con alpha igual a 0.01. Este valor es menor que el valor del estadístico obtenido 18.501. Por lo tanto, el test rechaza la hipótesis nula y por lo tanto se puede decir que existen diferencias significativas entre el número de condiciones por regla de los clasificadores obtenidos por los algoritmos comparados.

Tabla 8.7: Condiciones por regla

# Cond / Regla	ICRM	C45R	RIPPER	MPLCS	AntMiner+	CORE	ILGA	SLAVE
Appendicitis	0.99	**0.96**	2.04	1.88	1.87	1.55	3.22	2.51
Australian	**0.50**	3.15	3.22	3.49	4.00	2.21	8.36	3.52
Bupa	**0.90**	2.48	3.28	3.25	1.50	1.83	2.81	3.40
Contraceptive	**1.00**	5.04	5.93	2.92	3.14	1.76	4.09	4.19
Dermatology	**1.43**	2.56	2.44	3.38	8.30	9.15	14.12	1.98
Ecoli	**1.89**	2.98	2.43	2.47	2.82	2.69	3.18	3.30
Flare	**1.23**	3.02	5.20	6.20	4.71	3.42	9.14	0.00
Glass	**2.06**	3.17	2.46	2.67	2.65	3.21	4.16	3.46
Haberman	**0.50**	1.31	2.63	1.54	0.65	0.94	1.47	2.19
Heart	**1.00**	2.68	2.52	3.97	4.07	2.03	5.48	3.59
Iris	1.00	1.14	1.53	**0.87**	1.29	1.28	1.87	1.36
Lymphography	**1.25**	2.06	2.14	4.61	3.61	3.96	13.13	1.97
Monk-2	1.00	1.67	1.25	1.31	2.31	0.98	2.87	**0.67**
New-thyroid	**1.29**	1.87	1.55	1.69	1.51	2.38	2.34	1.67
Page-blocks	**1.43**	3.46	3.31	2.56	3.36	2.72	4.40	3.61
Pima	**0.50**	2.34	4.06	3.35	4.72	1.50	3.60	3.42
Saheart	**0.50**	1.97	3.48	3.38	3.94	1.70	4.40	3.83
Sonar	**0.89**	2.66	1.93	6.43	14.82	15.00	25.09	6.07
Tae	**1.17**	2.17	2.91	2.50	**1.17**	1.37	2.27	2.94
Thyroid	**1.38**	2.70	3.97	1.94	10.34	2.98	9.43	3.12
Vehicle	**1.25**	3.60	3.06	3.72	6.60	3.46	8.03	5.96
Wine	**1.47**	1.62	1.72	2.18	2.26	4.34	5.71	3.14
Zoo	**1.14**	2.11	1.94	1.39	3.26	4.85	13.53	0.00
Promedio	**1.12**	2.47	2.83	2.94	4.04	3.27	6.64	2.87
Ranking	**1.20**	3.91	4.57	4.39	5.02	4.00	7.17	5.74

La Figura 8.9 muestra la aplicación del test de Bonferroni–Dunn para el número de condiciones por regla con un alpha igual a 0.01, cuya diferencia crítica es 2.303. Los algoritmos a la izquierda de la diferencia crítica son algoritmos significativamente peores que ICRM. Todos los algoritmos muestran diferencias significativas comparados con nuestra propuesta, que es la que obtiene mejores resultados. Además, nuestra propuesta emplea los atributos verdaderamente relevantes. Observando los resultados de la Tabla 8.7, existen algunas diferencias significativas, por ejemplo, entre nuestra propuesta e ILGA sobre el conjunto de datos Sonar (2 clases y 60 atributos). Mientras que nuestra propuesta emplea un número medio de 0.89 condiciones por regla, ILGA requiere 25.09. Por lo tanto, ICRM apenas considera uno de los 60 atributos como relevantes, mientras que ILGA 25. En este caso, la diferencia de la exactitud para este conjunto de datos y métodos es de apenas un 1.4 %, mientras que el número de reglas es significativamente mejor.

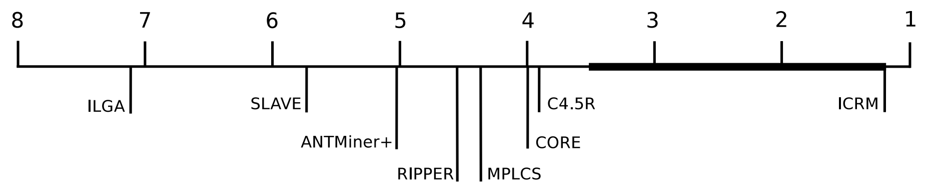


Figura 8.9: Test de Bonferroni–Dunn para el número de condiciones por regla

La Tabla 8.8 muestra los resultados del test de Wilcoxon para el número de condiciones por regla, realizando múltiples comparaciones por pares entre nuestra propuesta y los otros métodos. Los p-values devueltos por el test indican que existen diferencias significativas cuando se compara ICRM con todos los demás métodos con una confianza superior al 99 %.

Tabla 8.8: Test de Wilcoxon para el número de condiciones por regla

ICRM vs	R^+	R^-	p-value
C45R	275.0	1.0	4.768E-7
RIPPER	276.0	0.0	2.384E-7
MPLCS	273.0	3.0	1.192E-6
AntMiner+	253.0	0.0	4.768E-7
CORE	275.0	1.0	4.768E-7
ILGA	276.0	0.0	2.384E-7
SLAVE	266.0	10.0	1.025E-5

Condiciones por clasificador

El número total de condiciones del clasificador determina su complejidad total, independientemente del número de reglas, puesto que mide el número total de condiciones que se deben evaluar para clasificar las instancias. Cuanto mayor sea el número de condiciones, menor será su comprensibilidad y mayor será el tiempo requerido para evaluar las condiciones cuando se sometan a la predicción de la clase de las instancias.

Tabla 8.9: Condiciones del clasificador

# Cond / Clrf	ICRM	C45R	RIPPER	MPLCS	AntMiner+	CORE	ILGA	SLAVE
Appendicitis	**1.98**	3.10	14.80	7.77	3.90	6.13	96.47	12.27
Australian	**1.00**	35.17	75.93	17.20	23.20	10.50	250.83	24.70
Bupa	**1.80**	21.37	78.47	21.60	3.00	6.03	84.17	21.60
Contraceptive	**3.00**	156.43	382.50	17.27	11.73	16.77	122.70	188.00
Dermatology	**8.60**	23.60	34.90	26.23	55.17	62.27	423.73	21.20
Ecoli	**15.14**	34.10	86.40	21.80	21.60	17.07	95.43	42.03
Flare	**7.40**	84.57	528.40	103.80	95.80	16.23	274.10	0.00
Glass	**14.42**	31.87	54.67	22.07	12.23	22.70	124.67	56.47
Haberman	**1.00**	5.80	46.60	6.70	1.30	4.03	44.10	12.30
Heart	**2.00**	29.40	35.30	27.47	19.93	12.97	164.50	27.07
Iris	**3.00**	5.70	9.70	3.57	3.83	4.70	55.97	4.07
Lymphography	**5.00**	23.97	28.10	34.97	13.47	26.80	393.87	8.40
Monk-2	**2.00**	10.00	5.00	5.23	6.93	4.20	86.07	**2.00**
New-thyroid	**3.86**	12.70	10.73	9.27	5.33	7.80	70.17	9.10
Page-blocks	**7.17**	78.20	169.47	18.50	49.40	14.67	131.97	35.77
Pima	**1.00**	20.10	101.90	18.30	58.03	5.00	108.13	29.83
Saheart	**1.00**	13.37	85.53	21.10	14.90	12.03	132.10	43.30
Sonar	**1.78**	23.10	15.80	39.73	73.73	15.00	752.67	51.13
Tae	3.50	16.53	83.13	13.83	**2.33**	9.20	68.23	32.40
Thyroid	**4.14**	27.80	45.60	7.77	250.17	14.70	282.83	23.07
Vehicle	**5.00**	71.33	143.50	56.70	171.40	28.10	240.97	208.17
Wine	**4.42**	8.10	9.37	9.73	8.50	13.27	171.23	12.60
Zoo	**8.00**	18.40	17.73	9.73	13.93	34.17	405.77	0.00
Promedio	**4.62**	32.81	89.72	22.62	39.99	15.84	199.16	37.63
Ranking	**1.11**	4.61	6.26	4.11	3.74	3.35	7.61	5.22

La Tabla 8.9 muestra el número medio de condiciones de los clasificadores obtenidos y el ranking de los algoritmos. Los mejores valores con un menor número de condiciones son para nuestra propuesta, con un valor de ranking de 1.11, seguido de lejos por CORE. El estadístico de Iman y Davenport para el número medio de condiciones distribuído conforme a la distribución F con $K-1 = 7$ y $(K-1)(N-1) = 154$ grados de libertad es 36.767. El test establece un valor de distribución de 2.757 para un nivel de confianza con alpha igual a 0.01. Este valor es menor que el valor del estadístico obtenido 36.767. Por lo tanto, el test rechaza la hipótesis nula y por lo tanto se puede decir que existen diferencias significativas entre el número de condiciones de los clasificadores obtenidos por los algoritmos comparados.

La Figura 8.10 muestra la aplicación del test de Bonferroni–Dunn para el número de condiciones con un alpha igual a 0.01, cuya diferencia crítica es 2.303. Los algoritmos a la izquierda de la diferencia crítica son algoritmos significativamente peores que ICRM. Todos los algoritmos muestran diferencias significativas comparados con nuestra propuesta, que es la que obtiene mejores resultados. Observando los resultados de la Talba 8.9, existen diferencias significativas con todos los algoritmos excepto CORE para el número de condiciones de los clasificadores generados.

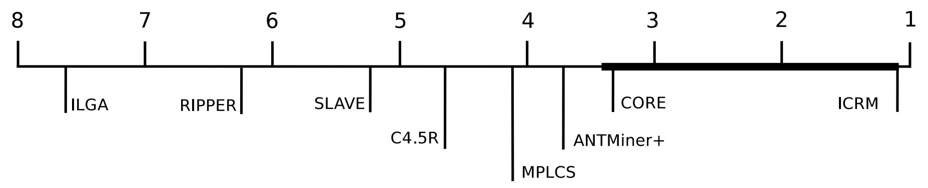


Figura 8.10: Test de Bonferroni–Dunn para el número de condiciones del clasificador

La Tabla 8.10 muestra los resultados del test de Wilcoxon para el número de condiciones del clasificador, realizando múltiples comparaciones por pares entre nuestra propuesta y los otros métodos. Los p-values devueltos por el test indican que existen diferencias significativas cuando se compara ICRM con todos los demás métodos con una confianza superior al 99 %.

Tabla 8.10: Test de Wilcoxon para el número de condiciones del clasificador

ICRM vs	R^+	R^-	p-value
C45R	276.0	0.0	2.384E-7
RIPPER	276.0	0.0	2.384E-7
MPLCS	276.0	0.0	2.384E-7
AntMiner+	264.0	12.0	1.669E-5
CORE	276.0	0.0	2.384E-7
ILGA	276.0	0.0	2.384E-7
SLAVE	253.0	0.0	4.768E-7

Complejidad

El índice de complejidad proporcionado por D. Nauck [63] y descrito en la sección 7, permite evaluar la interpretabilidad de una base de reglas y combina varios de los factores previos.

Tabla 8.11: Complejidad

Complejidad	ICRM	C45R	RIPPER	MPLCS	AntMiner+	CORE	ILGA	SLAVE
Appendicitis	**1.01**	0.65	0.14	0.26	0.51	0.33	0.02	0.16
Australian	**2.00**	0.06	0.03	0.12	0.09	0.19	0.01	0.08
Bupa	**1.11**	0.09	0.03	0.09	0.67	0.33	0.02	0.09
Contraceptive	**1.00**	0.02	0.01	0.17	0.26	0.18	0.02	0.02
Dermatology	**0.70**	0.25	0.17	0.23	0.11	0.10	0.01	0.28
Ecoli	**0.53**	0.23	0.09	0.37	0.37	0.47	0.08	0.19
Flare	**0.81**	0.07	0.01	0.06	0.06	0.37	0.02	0.00
Glass	**0.49**	0.22	0.13	0.32	0.57	0.31	0.06	0.12
Haberman	**2.00**	0.34	0.04	0.30	1.54	0.50	0.05	0.16
Heart	**1.00**	0.07	0.06	0.07	0.10	0.15	0.01	0.07
Iris	**1.00**	0.53	0.31	0.84	0.78	0.64	0.05	0.74
Lymphography	**0.80**	0.17	0.14	0.11	0.30	0.15	0.01	0.48
Monk-2	**1.00**	0.20	0.40	0.38	0.29	0.48	0.02	**1.00**
New-thyroid	**0.78**	0.24	0.28	0.32	0.56	0.38	0.04	0.33
Page-blocks	**0.70**	0.06	0.03	0.27	0.10	0.34	0.04	0.14
Pima	**2.00**	0.10	0.02	0.11	0.03	0.40	0.02	0.07
Saheart	**2.00**	0.15	0.02	0.09	0.13	0.17	0.02	0.05
Sonar	**1.12**	0.09	0.13	0.05	0.03	0.13	0.00	0.04
Tae	0.86	0.18	0.04	0.22	**1.29**	0.33	0.04	0.09
Thyroid	**0.72**	0.11	0.07	0.39	0.01	0.20	0.01	0.13
Vehicle	**0.80**	0.06	0.03	0.07	0.02	0.14	0.02	0.02
Wine	**0.68**	0.37	0.32	0.31	0.35	0.23	0.02	0.24
Zoo	**0.88**	0.38	0.39	0.72	0.50	0.20	0.02	0.00
Promedio	**1.04**	0.20	0.12	0.26	0.38	0.29	0.03	0.20
Ranking	**1.11**	4.61	6.26	4.04	3.85	3.37	7.57	5.20

La Tabla 8.11 muestra el índice de complejidad medio de los clasificadores obtenidos y el ranking de los algoritmos. Los mejores valores de complejidad son para nuestra propuesta, con un valor de ranking de 1.11. El estadístico de Iman y Davenport para el número medio de condiciones distribuído conforme a la distribución F con $K-1 = 7$ y $(K-1)(N-1) = 154$ grados de libertad es 37.248. El test establece un valor de distribución de 2.757 para un nivel de confianza con alpha igual a 0.01. Este valor es menor que el valor del estadístico obtenido 37.248. Por lo tanto, el test rechaza la hipótesis nula y por lo tanto se puede decir que existen diferencias significativas entre los valores de complejidad de los clasificadores obtenidos por los algoritmos comparados.

La Figura 8.11 muestra la aplicación del test de Bonferroni–Dunn para el grado de complejidad con un alpha igual a 0.01, cuya diferencia crítica es 2.303. Los algoritmos a la izquierda de la diferencia crítica son algoritmos significativamente peores que ICRM. Todos los algoritmos, excepto CORE, muestran diferencias significativas comparados con nuestra propuesta, que es la que obtiene mejores resultados con respecto a la medida de complejidad de los clasificadores. Es interesante destacar que los resultados de CORE, siendo el segundo algoritmo más interpretable por esta medida, es de los que peor exactitud tiene, mientras que el nuestro se muestra el segundo mejor. Además, la exactitud de MPLCS apenas supera un 1.6 % de media a ICRM, mientras que el valor del índice de complejidad para MPLCS es de 0.26, significativamente peor que para ICRM con 1.04.

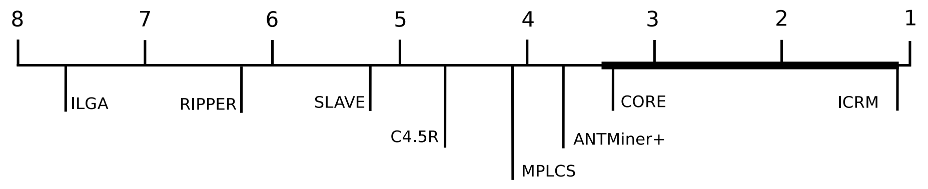


Figura 8.11: Test de Bonferroni–Dunn para la complejidad

La Tabla 8.12 muestra los resultados del test de Wilcoxon para la complejidad del clasificador, realizando múltiples comparaciones por pares entre nuestra propuesta y los otros métodos. Los p-values devueltos por el test indican que existen diferencias significativas cuando se compara ICRM con todos los demás métodos con una confianza superior al 99 %. Estos son los mejores resultados alcanzados por nuestra propuesta de entre todas las métricas consideradas.

Tabla 8.12: Test de Wilcoxon para la complejidad

ICRM vs	R^+	R^-	p-value
C45R	276.0	0.0	2.384E-7
RIPPER	276.0	0.0	2.384E-7
MPLCS	276.0	0.0	2.384E-7
AntMiner+	268.0	8.0	5.960E-6
CORE	276.0	0.0	2.384E-7
ILGA	276.0	0.0	2.384E-7
SLAVE	253.0	0.0	4.768E-7

Tiempo de ejecución

La Tabla 8.13 muestra el tiempo de ejecución de los algoritmos, que incluye el tiempo (en segundos) para construir el clasificador a partir de los datos de entrenamiento y evaluar las predicciones para las instancias de entrenamiento y test. Los mejores valores con un menor tiempo de ejecución son para RIPPER con un valor de ranking de 1.67, seguido por C45R. El estadístico de Iman y Davenport para el

Tabla 8.13: Tiempo de ejecución

Tiempo(s)	ICRM	C45R	RIPPER	MPLCS	AntMiner+	CORE	ILGA	SLAVE
Appendicitis	0.78	**0.15**	0.19	3.44	2.67	5.22	3.89	3.44
Australian	2.26	**1.00**	1.19	18.56	55.93	35.41	56.89	32.26
Bupa	**0.54**	0.59	0.56	8.22	5.56	10.63	8.63	8.00
Contraceptive	**2.94**	11.52	3.59	45.22	66.74	42.89	55.15	194.19
Dermatology	134.59	0.59	**0.48**	48.70	118.44	60.96	144.59	51.81
Ecoli	26.49	0.59	**0.56**	13.30	32.07	22.70	15.41	29.63
Flare	1.44	**0.81**	3.52	38.37	178.56	59.67	73.11	17.67
Glass	24.45	**0.52**	**0.52**	9.30	11.74	13.96	15.52	26.52
Haberman	**0.12**	0.30	0.33	4.89	5.48	8.00	3.22	3.70
Heart	1.64	0.63	**0.48**	13.70	33.04	13.04	27.00	12.19
Iris	0.71	0.15	**0.11**	4.11	4.07	9.33	2.93	2.41
Lymphography	3.54	0.30	**0.26**	6.30	12.15	21.15	19.96	10.67
Monk-2	0.27	0.33	**0.19**	10.74	12.22	14.44	7.41	4.37
New-thyroid	1.49	0.22	**0.19**	5.41	7.52	10.81	5.70	5.11
Page-blocks	46.22	**10.26**	14.67	191.30	616.30	225.63	469.74	352.11
Pima	1.36	**0.96**	1.48	19.11	115.33	24.15	35.63	22.59
Saheart	1.08	**0.67**	1.04	11.48	28.78	13.63	33.96	20.74
Sonar	98.20	**0.70**	0.96	34.63	78.44	34.11	152.74	21.96
Tae	0.65	0.33	**0.26**	4.52	2.78	5.85	4.19	6.26
Thyroid	71.61	4.37	**4.15**	629.81	9671.81	893.44	813.78	331.07
Vehicle	32.74	**2.00**	2.63	85.85	274.22	73.67	158.52	204.15
Wine	13.86	**0.26**	**0.26**	9.41	12.78	16.48	14.59	7.19
Zoo	2.98	**0.19**	**0.19**	3.07	9.59	14.93	16.70	5.19
Promedio	20.43	**1.63**	1.64	53.02	493.75	70.87	93.01	59.71
Ranking	3.39	1.72	**1.67**	4.76	6.48	6.30	6.48	5.20

número medio de condiciones distribuído conforme a la distribución F con $K-1 = 7$ y $(K-1)(N-1) = 154$ grados de libertad es 46.922. El test establece un valor de distribución de 2.757 para un nivel de confianza con alpha igual a 0.01. Este valor es menor que el valor del estadístico obtenido 46.922. Por lo tanto, el test rechaza la hipótesis nula y por lo tanto se puede decir que existen diferencias significativas entre el tiempo de ejecución de los algoritmos comparados.

La Figura 8.12 muestra la aplicación del test de Bonferroni–Dunn para el tiempo de ejecución con un alpha igual a 0.01, cuya diferencia crítica es 2.303. Los algoritmos a la izquierda de la diferencia crítica son algoritmos significativamente peores que ICRM, y aquellos situados a la derecha del intervalo son peores. Observando esta figura, ningún algoritmo es significativamente más rápido que ICRM, puesto que Ripper y C45R sí son más rápidos al ser no evolutivos, pero su ranking está dentro de los límites del test de Bonferroni–Dunn. Por otro lado, ICRM es el más rápido de todos los algoritmos evolutivos, y es significativamente más rápido que AntMiner+, CORE e ILGA.

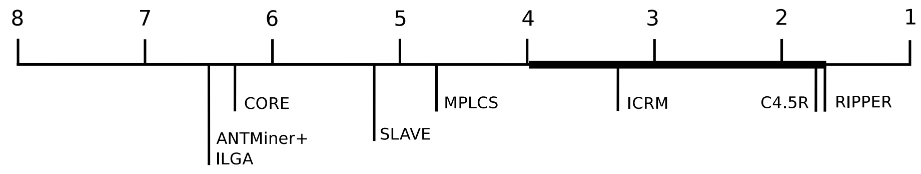


Figura 8.12: Test de Bonferroni–Dunn para el tiempo de ejecución

La Tabla 8.14 muestra los resultados del test de Wilcoxon para el tiempo de ejecución, realizando múltiples comparaciones por pares entre nuestra propuesta y los otros métodos. Los p-values devueltos por el test indican que existen diferencias significativas cuando se compara ICRM con todos los demás métodos evolutivos con una confianza superior al 99 %.

Tabla 8.14: Test de Wilcoxon para el tiempo de ejecución

ICRM vs	R^+	R^-	p-value
C45R	36.0	240.0	≥ 0.2
RIPPER	42.0	234.0	≥ 0.2
MPLCS	244.0	32.0	6.388E-4
AntMiner+	262.0	14.0	2.622E-5
CORE	258.0	18.0	6.032E-5
ILGA	262.0	14.0	2.622E-5
SLAVE	257.0	19.0	7.320E-5

Comparación general

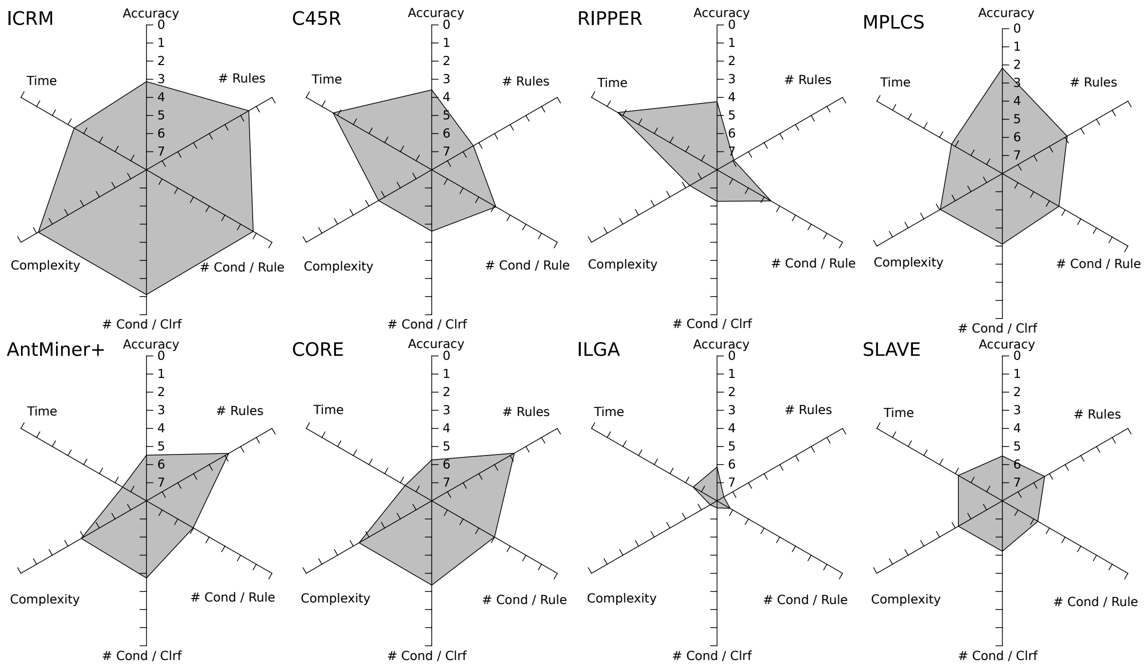


Figura 8.13: Ranking de los algoritmos

La Figura 8.13 resume los rankings obtenidos por los clasificadores generados por los algoritmos con respeto a las diferentes métricas. Cuanto mayor sea el área, mejores valores han obtenido los clasificadores. Esta figura muestra el gran desempeño general de nuestra propuesta ICRM y los malos resultados de ILGA. RIPPER y C45R son los algoritmos más rápidos puesto que son no evolutivos, pero proporcionan clasificadores con muchas reglas. MPLCS proporciona los clasificadores más precisos pero su tiempo de ejecución está entre los peores puesto que emplea una codificación Pittsburgh. AntMiner+ y CORE proporcionan un número relativamente bajo de reglas pero se ejecutan más lentos. ILGA y SLAVE son definitivamente los peores algoritmos de la comparativa. Concretamente ILGA alcanza el peor ranking sobre todas las métricas consideradas excepto para el número de reglas, para el cual RIPPER es el peor. Nuestra propuesta obtiene los mejores rankings de acuerdo al número de reglas, número de condiciones por regla, número de condiciones del clasificador y complejidad. Es también el segundo mejor algoritmo en exactitud y el tercero en tiempo de ejecución (primero de los evolutivos).

9. UN MODELO DE MINERÍA DE REGLAS MEDIANTE PG

La tarea de clasificación ha sido resulta con numerosos métodos de sistemas basados en reglas [25, 38], incluyendo algoritmos evolutivos de Programación Genética [29, 40, 76, 78].

La representación de individuos y reglas en algoritmos evolutivos siguen dos aproximaciones. La primera, individuo = regla, evoluciona cada individuo como la representación de una única regla, y el clasificador se obtiene de la agregación de varias reglas obtenidas a lo largo del proceso evolutivo, dentro de este paradigma se encuentran los métodos tipo Michigan, Iterative Rule Learning (IRL) y Genetic Cooperative-Competitive Learning (GCCL). La segunda, individuo = conjunto de reglas, es conocida como Pittsburgh [73].

La principal ventaja de los métodos Pittsburgh frente a los primeros es que permiten tratan con el problema de la cooperación-competición directamente, resolviendo la interacción entre las reglas durante el proceso evolutivo. Sin embargo, su mayor problema es controlar el número de reglas, puesto que el número total de reglas de toda la población puede crecer en gran medida, incrementando el coste computacional del algoritmo hasta hacerlo intratable.

En este capítulo se propone un método de Programación Genética basado en gramática (G3P) que aprende reglas en forma normal disjuntiva (DNF) mediante una gramática de contexto libre. Bajo un modelo Pittsburgh, cada individuo es una lista de decisión [72] conformada por varias reglas ordenadas. Será necesario diseñar operadores genéticos que permitan el cruce y mutación de reglas individuales, así como la mezcla entre los individuos.

Un proceso evolutivo tipo Pittsburgh con un número elevado de reglas introduce una alta carga computacional. Para resolver este problema y reducir el tiempo de ejecución, la fase de evaluación se paraleliza empleando GPUs. Además, se limita y controla el número máximo de reglas por individuo, lo que nos permite controlar la complejidad de los individuos y el tiempo de ejecución.

Descripción del modelo

En esta sección se describe la representación de los individuos, los operadores genéticos, la función objetivo, el procedimiento de inicialización de la población y el modelo generacional del algoritmo.

Representación de los individuos

Una de las ventajas de la Programación Genética es su flexibilidad a la hora de representar múltiples tipos de soluciones, como árboles de decisión, reglas de clasificación, funciones discriminantes, etc. [33]. En nuestro caso, los individuos representan clasificadores complejos donde cada individuo es un conjunto de reglas ordenadas generadas mediante una gramática de contexto libre cuyo árbol de expresión se compone por nodos terminales y no terminales. El clasificador, se expresa como un conjunto de reglas *Si–Entonces* ordenado en forma de lista de decisión. El antecedente de una regla representa una composición de comparaciones atributovaloradas y el consecuente especifica la clase predicha por la regla. En nuestro caso, emplearemos los operadores lógicos (AND, OR, NOT), los operadores relacionales (<, =, <>, >) y el operador de intervalo (IN). Es importante asegurar que el clasificador contiene al menos una regla por clase para garantizar en lo posible su completitud.

Operadores genéticos

Los operadores genéticos se usan tanto para mezclar material genético de los individuos de la población en la búsqueda de una solución mejor por explotación, como para buscar nuevos espacios en donde explorar nuevas soluciones.

Operador de cruce

El operador de cruce actúa a dos niveles, ya que un individuo representa un conjunto de reglas, y se puede tanto cruzar individuos como cruzar reglas.

Por un lado, el operador de cruce aplicado a las reglas toma dos reglas de un individuo, las cruza, y produce dos nuevas reglas. Para ello, se seleccionan dos nodos compatibles del árbol de derivación de la gramática de cada uno de los individuos y se intercambian las ramas encabezadas por dichos nodos.

Por otro lado, el operador de cruce intercambia reglas entre indidivuos completos. Dados dos individuos padres, se seleccionan dos puntos de cruce, uno por padre, de forma que las reglas se intercambian a partir de dichos puntos, generando dos nuevos individuos. El punto de selección deberá de garantizar que al menos una regla se intercambie.

Operador de mutación

El operador de mutación se puede aplicar también tanto a las reglas como a los individuos. La mutación de una regla actúa sobre un nodo del árbol de derivación seleccionado aleatoriamente. A partir de dicho nodo y empleando la gramática, se genera un nuevo subárbol creado aleatoriamente. La mutación de un individuo se determina por la eliminación aleatoria de una de las reglas de la lista de decisión con un cierto grado de probabilidad.

Función objetivo

El algoritmo emplea dos funciones objetivo para dirigir la optimización de las reglas y los clasificadores. La primera evalúa la calidad de cada regla independientemente. La segunda evalúa los individuos como clasificadores y busca maximizar la exactitud sobre las instancias del conjunto de entrenamiento. Primero es necesario evaluar las reglas de cada individuo y luego evaluar la exactitud de los clasificadores.

Como el tiempo de ejecución es uno de los mayores problemas de los métodos Pittsburgh, se ha paralelizado la fase de evaluación siguiendo el modelo descrito en el primer capítulo, solucionando este problema y consiguiendo un tiempo de ejecución razonable.

Función objetivo aplicada a las reglas

La función objetivo aplicada a las reglas es la propuesta por Bojarczuk et al. [40]. Cada regla se evalúa para ver qué clase es la que mejor la representa, variando la clase predicha por el consecuente. Para cada clase, se somete la regla al conjunto de instancias de entrenamiento y se observa la calidad de las predicciones de la regla. Concretamente, se obtienen los verdaderos positivos (T_P), verdaderos negativos (T_N), los falsos positivos (F_P) y los falsos negativos (F_N). Empleando estos valores, se construye la matriz de confusión y se obtienen dos indicadores, sensibilidad (Se) y especificidad (Sp).

$$Se = \frac{T_P}{T_P + F_N} \qquad Sp = \frac{T_N}{T_N + F_P} \qquad fitness = Se * Sp$$

Función objetivo aplicada al clasificador

La función objetivo aplicada al clasificador se ejecuta una vez que se ha calculado el mejor consecuente para cada regla. Evaluamos la calidad de un clasificador como la exactitud, el ratio de predicciones correctas frente al número de instancias. Cada instancia se somete a la lista de decisión que comprueba de forma ordenada si las reglas cubren a la instancia, en dicho caso, se predice la clase del consecuente de la regla. Si el consecuente coincide con la clase verdadera de la instancia es un acierto, en caso contrario ha sido una predicción errónea.

El proceso de activación de las reglas define el orden en que se comprueban las reglas para cubrir una instancia. En nuestro caso empleamos una lista de decisión como conjunto ordenado de reglas *Si–Entonces*. Una instancia se clasifica con la clase del consecuente de la primera regla que la cubra. Si ninguna regla cubre a la instancia, entonces se predice una clase por defecto, que será la clase de mayor frecuencia en el conjunto de datos.

Inicialización

La inicialización del algoritmo con una población genéticamente diversa es crucial para obtener buenos resultados. El problema de la función objetivo que utilizamos para evaluar las reglas es que una clase minoritaria puede ser ignorada. Por lo tanto, debemos asegurarnos de que los individuos contienen al menos una regla para cada clase. Una manera de hacer esto, una vez que los individuos se crean, es completar con nuevas reglas de las clases que aún no han sido cubiertas por al menos una regla. Como puede ser que sea muy costoso obtener al menos una regla para todas y cada una de las clases, el proceso de completar el clasificador se realiza un número de veces dependiendo del número de clases. De esta manera, se intenta que la mayor parte de las clases estén representadas en el clasificador. Por último, entre todas las reglas de todos los individuos, siempre se guarda la mejor regla para cada clase que nos ayudará más adelante para rellenar los clasificadores con las reglas de las clases que podría faltar.

Modelo generacional

En esta sección se describe el modelo generacional representado en la Figura 9.1. El cuadro de la izquierda representa la inicialización de los individuos descrita en la sección anterior. Una vez que la población se ha inicializado, el algoritmo procede de forma iterativa a aplicar los operadores de cruce y mutación descritos en la sección 9.1.2. Específicamente, el algoritmo realiza el cruce de individuos intercambiando subconjuntos de reglas. Los individuos pueden mutar eliminando algunas de sus reglas. Las reglas de cada individuo se cruzan entre sí y se mutan, obteniendo nuevas reglas de clasificación. Estas nuevas reglas deben ser evaluadas para obtener sus mejores consecuentes y su valor de aptitud.

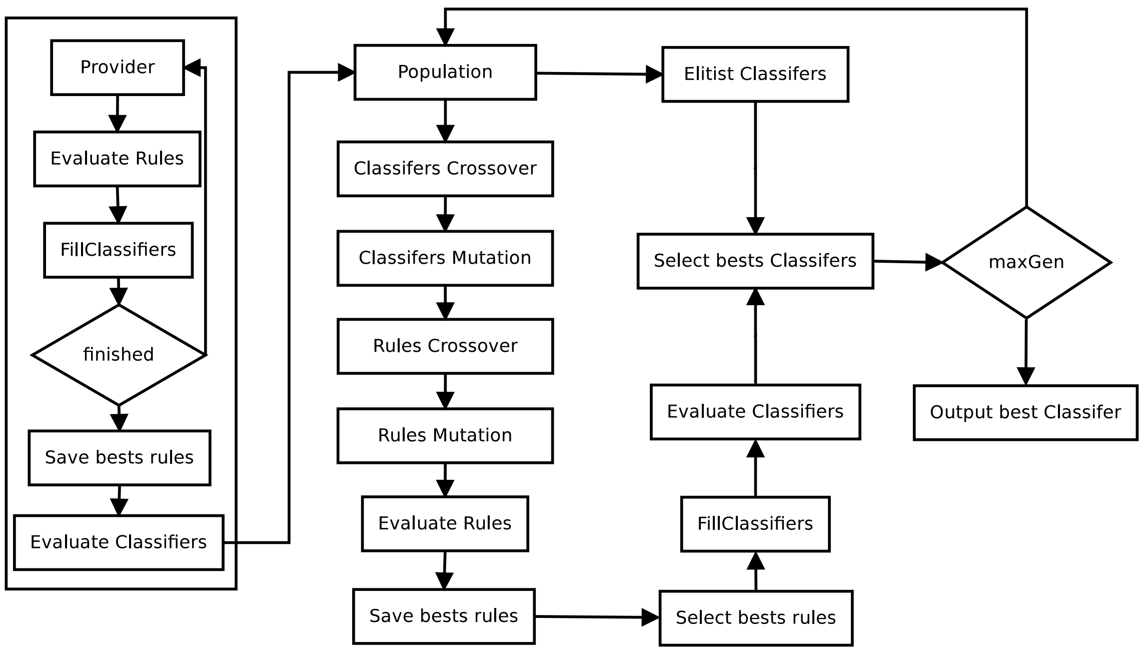


Figura 9.1: Diagrama del algoritmo PG

Para garantizar la supervivencia de las mejores reglas, el algoritmo guarda en cada generación y para cada clase, si una regla nueva es mejor que la mejor almacenada de esa clase, si es así la regla se sustituye por la nueva mejor. A medida que el operador de cruce puede haber creado las individuos que superen el número máximo de reglas permitas, se puede simplificar mediante la selección de los mejores subconjunto reglas.

El individuo debe ser completado por las reglas de las clases no cubiertas, tomando las mejores reglas guardadas para cubrir dichas clases. Una vez completado, cada individuo debe ser evaluado para obtener su aptitud con los datos de entrenamiento. El elitismo se encarga de mantener un subconjunto de los mejores individuos en cada generación para asegurar la supervivencia de los individuos más aptos. El algoritmo termina cuando se ha encontrado un individuo que clasifica correctamente todas las instancias de entrenamiento o cuando el algoritmo ha iterado un cierto número de generaciones.

Estudio experimental del modelo PG

Esta sección describe los detalles de los experimentos realizados en diferentes conjuntos de datos para evaluar las capacidades de la propuesta y compararla con otros métodos de clasificación. A continuación se presentan los conjuntos de datos empleados y los algoritmos utilizados para la comparación, la descripción de la metodología experimental y las pruebas estadísticas utilizadas para la validación de los resultados.

Los experimentos realizados comparan los resultados de 11 algoritmos de clasificación con 18 conjuntos de datos. Estos algoritmos están disponibles en el sitio web de JCLEC [80] y KEEL [4]. Los conjuntos de datos empleados han sido seleccionados de la página web del repositorio de KEEL [3]. Estos datos son muy variados teniendo en cuenta los diferentes grados de complejidad, el número de clases, el número de características y el número de instancias. Así, el número de clases es de hasta 10, el número de características oscila entre 4 y 60 y el número de instancias varía desde 101 hasta 58000.

Para evaluar adecuadamente el desempeño del algoritmo propuesto se considera hacer un estudio comparativo con algunos de los algoritmos evolutivos de aprendizaje de reglas para clasificación (De Falco et al. [29], Bojarczuk et al. [40], Tan et al. [76, 78], MPLCS [6], ILGA [46], CORE [77] y UCS [9]), dos algoritmos de reglas no evolutivos (PART [38] y RIPPER [25]) y el algoritmo clásico de árbol de decisión (C4.5 [70]). Para cada algoritmo, los valores de los parámetros se establecen a los valores óptimos proporcionados por los autores. El tamaño de la población y el número de generaciones para nuestra propuesta se establecen en 200, es decir, el algoritmo evoluciona con 200 clasificadores, y el número máximo de reglas por clase se establece en 3. Los resultados se validan utilizando 10-fold cross validation y las pruebas con algoritmos estocásticos se repiten 10 veces con semillas diferentes. Los resultados proporcionados en las tablas son el promedio de las 100 ejecuciones.

Resultados del modelo PG

En esta sección presentamos los resultados de los experimentos y la exactitud de los algoritmos en los conjuntos de datos seleccionados. En la Tabla 9.1 se muestran los resultados promedio de la exactitud, obtenidos al ejecutar cada algoritmo diez veces en cada uno de los conjuntos de datos y sus particiones. La penúltima fila indica los resultados medios del algoritmo y la última fila indica los rankings obtenidos.

El análisis de los resultados de la tabla indica de que el algoritmo propuesto obtiene los mejores resultados de exactitud y un mejor promedio de clasificación. Su exactitud es considerablemente mayor que la mayoría de los algoritmos y está muy cerca y por encima que los algoritmos MPLCS, UCS y C4.5. En los conjuntos de datos en los que nuestro algoritmo no logra los mejores resultados, su exactitud es generalmente muy competitiva. Por otra parte, el algoritmo no destaca negativamente en ninguno de los conjuntos de datos.

Con el fin de analizar los resultados y descubrir la existencia de diferencias significativas entre los algoritmos, una serie de tests estadísticos [42, 43] se han llevado a cabo. Usamos el test de Iman y Davenport para clasificar a los k algoritmos sobre los N conjuntos de datos. El estadístico de Iman y Davenport para la exactitud media de acuerdo a la distribución F con $k-1 = 11$ y $(k-1)(N-1) = 187$ grados de libertad es $30,1460$. Este valor no pertenece al intervalo crítico $[0, F = 2,34390{,}01, 11, 187]$ para $p = 0,01$. Por lo tanto, rechazamos la hipótesis nula de que todos los algoritmos funcionan igualmente bien. Con el fin de analizar si existen diferencias significativas entre los algoritmos, se utiliza el test de Bonferroni–Dunn para encontrar las diferencias con una distancia crítica (CD) con el valor de $p = 0,01$ igual a 3,9865.

Los resultados indican que un nivel de significa de p $= 0,01$ (es decir, con una confianza del 99 %), existen diferencias significativas entre nuestro algoritmo y el de Falco De et al. Bojarczuk et al., Tan et al., PART, IGLA y CORE, siendo los resultados de nuestro algoritmo estadísticamente significativamente mejores. Estas

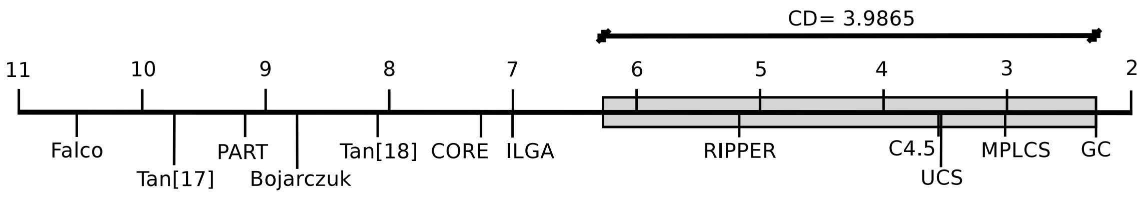


Figura 9.2: Test de Bonferroni–Dunn para la exactitud

diferencias se muestran en la Figura 9.2, que representa los rankings de los algoritmos y el intervalo de distancia crítica. En cuanto a los otros algoritmos, el test no indica diferencias significativas. Sin embargo, nuestra propuesta obtiene el ranking menor, lo que indica que teniendo en cuenta todos los datos, obtiene mejores resultados en un mayor número de casos que otras propuestas.

Tabla 9.1: Resultados de exactitud mediante 10-fold cross-validation

Dataset	PG	Falco	Bojarczuk	Tan [76]	Tan [78]	MPLCS	ILGA	CORE	UCS	C4.5	PART	RIPPER
Zoo	95.90 %	61.73 %	86.31 %	93.43 %	92.75 %	95.50 %	84.67 %	94.58 %	**96.50 %**	92.80 %	86.13 %	93.41 %
Iris	95.82 %	76.20 %	84.40 %	89.67 %	75.34 %	96.00 %	93.33 %	94.67 %	92.00 %	**96.67 %**	33.33 %	94.00 %
Hepatitis	**88.51 %**	68.85 %	74.44 %	73.50 %	83.43 %	85.15 %	77.92 %	83.43 %	80.74 %	85.66 %	84.17 %	79.09 %
Wine	94.58 %	63.72 %	79.33 %	75.81 %	66.24 %	91.54 %	89.28 %	**95.49 %**	91.57 %	94.90 %	63.98 %	93.79 %
Sonar	75.09 %	61.90 %	66.31 %	60.57 %	59.05 %	76.81 %	71.57 %	53.38 %	**77.83 %**	70.54 %	60.53 %	72.45 %
Glass	**69.39 %**	31.79 %	39.98 %	36.88 %	48.85 %	65.88 %	52.40 %	50.97 %	66.07 %	68.86 %	46.50 %	63.47 %
New-thyroid	**94.40 %**	69.18 %	75.54 %	73.65 %	78.66 %	91.65 %	90.71 %	91.21 %	93.55 %	93.52 %	77.22 %	93.05 %
Heart	81.85 %	66.78 %	74.37 %	72.55 %	76.30 %	**83.70 %**	64.44 %	71.11 %	80.37 %	78.14 %	57.77 %	76.29 %
Dermatology	94.96 %	45.61 %	73.59 %	77.29 %	76.56 %	95.53 %	60.81 %	43.86 %	**96.36 %**	94.35 %	73.12 %	94.42 %
Haberman	**73.82 %**	65.92 %	68.43 %	52.70 %	70.18 %	72.17 %	71.89 %	70.88 %	72.18 %	71.19 %	73.53 %	46.72 %
Ecoli	78.92 %	51.72 %	57.63 %	50.48 %	65.19 %	**80.67 %**	63.10 %	65.78 %	79.49 %	77.37 %	44.67 %	72.63 %
Australian	85.41 %	72.42 %	84.29 %	75.77 %	78.41 %	**86.96 %**	85.07 %	83.33 %	86.09 %	85.21 %	60.72 %	81.44 %
Pima	75.01 %	69.82 %	67.59 %	62.98 %	66.68 %	74.60 %	73.19 %	72.28 %	**76.04 %**	72.53 %	65.11 %	69.53 %
Vehicle	70.59 %	31.11 %	41.67 %	36.04 %	41.52 %	71.15 %	57.24 %	40.05 %	**72.45 %**	66.66 %	37.85 %	70.44 %
Contraceptive	**55.61 %**	40.75 %	42.68 %	40.37 %	44.00 %	55.47 %	43.59 %	45.01 %	49.49 %	51.19 %	42.90 %	50.78 %
Thyroid	99.22 %	68.05 %	51.39 %	52.92 %	92.43 %	94.72 %	94.10 %	68.00 %	96.99 %	**99.56 %**	92.58 %	99.37 %
Penbased	83.32 %	25.54 %	40.29 %	35.76 %	44.90 %	91.80 %	53.25 %	15.69 %	14.28 %	94.89 %	15.86 %	**96.15 %**
Shuttle	**99.97 %**	61.16 %	75.51 %	63.55 %	89.51 %	99.60 %	93.67 %	91.60 %	99.62 %	99.96 %	99.60 %	99.96 %
Promedio (%)	**84.02 %**	57.34 %	65.76 %	62.44 %	69.44 %	83.82 %	73.34 %	68.40 %	78.98 %	83.00 %	61.97 %	80.38 %
Ranking	**2.22**	10.56	8.72	9.72	8.08	3.03	7.00	7.25	3.56	3.58	9.19	5.14

IV

CLASIFICACIÓN GRAVITATORIA

10. MODELOS BASADOS EN GRAVEDAD

El vecino más cercano (NN) [27] es un método basado en instancias y podría ser el modelo de clasificación más sencillo, que consiste en asignar a un ejemplo la clase del ejemplo más cercano. La versión extendida del vecino más cercano a los k vecinos y sus derivados se consideran una de las familias de algoritmos más influyentes y han demostrado su eficacia en muchos ámbitos [54]. Sin embargo, el principal problema de estos métodos es que sufren gravemente de datos con ruido y de problemas de alta dimensionalidad, ya que su rendimiento es muy lento y su exactitud tiende a deteriorarse a medida que aumenta la dimensionalidad.

En los últimos años, se han propuesto nuevos métodos basados en instancias basados en gravitación que resuelven estos problemas de los clasificadores del vecino más cercano [68, 81, 91].

Estos modelos se inspiran en la ley de gravitación universal de Newton, simulando la fuerza gravitatoria entre los datos para realizar la clasificación. Estos modelos amplían el concepto de vecindad a la ley de gravitación entre los objetos en el mundo real. El principio de la clasificación se basa en calcular la gravedad de una instancia frente a todas las demás, concluyendo que el grado de pertenecer a una clase aumenta conforme la gravedad de una instancia a esa clase es más fuerte.

Newton publicó en 1687 la ley de gravitación universal, que establece que cada objeto en el universo atrae al resto de elementos con una fuerza que es directamente proporcional al producto de sus masas e inversamente proporcional al cuadrado de la distancia entre ellos. Esta fuerza de atracción se calcula como:

$$F = G\frac{m_1 m_2}{r^2}$$

donde F es la fuerza entre los objetos, G es la constante de gravitación unviersal, m_1 es la masa del primer objeto, m_2 es la masa del segundo objeto, y r es la distancia entre los objetos.

La relatividad general generaliza la ley de Newton, proporcionando una descripción unificada de la gravedad como una propiedad geométrica del espacio–tiempo.

La curvatura del espacio–tiempo se suele representar mediante diagramas de incrustación [44]. Estos diagramas muestran el campo gravitatorio que rodea a un objeto en el espacio. Cada punto en el espacio tendrá un valor de potencial gravitatorio proporcional a la función potencial:

$$\Phi = G\frac{m}{r^2}$$

El primer algoritmo de aprendizaje automático inspirado en el modelo gravitatorio fue propuesto en 1977 por W. E. Wright [85] para la realización de análisis de clusters de datos en el espacio euclídeo. Más recientemente, Y. Endo y H. Iwata [32] propusieron en 2005 un algoritmo de clustering dinámico basado en la gravitación universal, que emplea tanto la información global y local de los datos. Aunque ambas propuestas se centran en la clustering, se basan en el principio de gravitación.

En cuanto a los modelos de clasificación basados en gravedad, C. Wang y Chen Y. Q [81] presentaron en 2005 una mejora del vecino más cercano (NN) utilizando el concepto de colapso gravitacional para reducir y definir los límites de la distribución de cada clase, ya que el rendimiento del clasificador NN se reduce significativamente con el aumento de la superposición de la distribución de diferentes clases. Su algoritmo genera prototipos para el clasificador NN por la migración de las instancias siguiendo la órbita gravitatoria.

Hay más contribuciones que aplican la teoría de la gravitación a clasificación, como la propuesta de B. Yang et al. [87] para la detección de intrusiones en la red o la de Y. Zong-Chang [91], que propuso en 2007 un modelo gravitatorio vectorial derivado del análisis geométrico del clasificador lineal.

El trabajo más reciente y completo en relación con la clasificación de datos mediante gravedad fue presentada por L. Peng et al. [68] en 2009. Emplean pesos para describir la importancia de las características del modelo de clasificación. Los pesos están optimizados por un algoritmo iterativo. La fuerza de la gravedad de una partícula a las diferentes clases se calcula utilizando la masa de las partículas y la distancia a su centro de gravedad, que representa el centro de masa de una clase. El algoritmo genera partículas de datos utilizando el principio de la máxima distancia. Sin embargo, este método reduce la exactitud, especialmente en las zonas alejadas del centro de gravedad de las partículas de datos.

11. IGCA. UN MODELO GRAVITATORIO MEJORADO

En este capítulo se proporciona una descripción de la propuesta del modelo gravitatorio llamado IGCA (improved gravitation-based classification algorithm). Primero, es necesario definir los conceptos y principios fundamentales de la clasificación basada en gravedad, que consiste en asignar a una instancia la clase con el máximo potencial gravitatorio. Por lo tanto, dada una instancia desconocida $\bar{x}$ y un conjunto de datos con k clases, el algoritmo calcula la gravedad de las otras instancias para las diferentes clases hasta el punto en el que la instancia $\bar{x}$ se sitúa. Finalmente, el modelo predice la clase del máximo potencial gravitatorio.

$$Clase(\bar{x}) = max\left\{g(\bar{x}, c_1), g(\bar{x}, c_2), \ldots, g(\bar{x}, c_k)\right\}$$

donde $g(\bar{x}, c)$ es la fuerza de gravedad ejercida sobre una partícula $\bar{x}$ y debida a la atracción de las instancias de la clase c.

Definición de distancia

La gravedad es inversamente propocional al cuadrado de la distancia. La gravedad de Newton se formula en base a la geometría euclídea, que es la más natural. La distancia euclídea entre dos objetos $\bar{x_1}$ y $\bar{x_2}$ se define como:

$$d(\bar{x_1}, \bar{x_2}) = \sqrt{\sum_{i=1}^{f} (x_{1i} - x_{2i})^2}$$

donde el número de atributos (dimensiones) es f. Todos los atributos numéricos deben ser normalizados al intervalo $[0, 1]$ para un cálculo equitativo de la distancia. Con el objetivo de encontrar un criterio razonable para calcular la distancia entre las etiquetas de los atributos nominales se emplea la métrica de solapamiento que se define como:

$$\delta(x_{1i}, x_{2i}) = \begin{Bmatrix} 0 & \text{si} & x_{1i} = x_{2i} \\ 1 & \text{si} & x_{1i} \neq x_{2i} \end{Bmatrix}$$

Esta definición de distancia considera a todos los atributos igualmente relevantes, pero la selección de características ha demostrado mejorar los resultados de los algoritmos de clasificación [58]. La selección de características permite la detección y filtrado de atributos irrelevantes, considerando la entropía de la distribución de las instancias conforme a los atributos y clases.

Peng et al. [68] demostraron el buen funcionamiento del ponderamiento del peso de los atributos en el cálculo de la distancia mediante un algoritmo iterativo. Sin embargo, el peso de cada atributo puede ser distinto para cada clase, puesto que la distribución de las instancias no es la misma para todas las clases. Por lo tanto, en vez de emplear un vector de pesos de atributos de longitud f, nuestro algoritmo emplea un vector de pesos atributo–clase $\bar{w}$ de longitud $f * k$, donde k es el número de clases. Finalmente, la función de distancia se reescribe para tener en cuenta los pesos de los atributos para cada clase:

$$\bar{w} = \langle w_{1,1}, w_{1,2}, \ldots, w_{1,f}, w_{2,1}, w_{2,2}, \ldots, w_{k,f} \rangle$$

$$d(\bar{x_1}, \bar{x_2}, c) = \sqrt{\sum_{i=1}^{f} w_{c,i} \cdot (x_{1i} - x_{2i})^2}$$

Optimización de los pesos atributo–clase

La optimización del vector de pesos atributo–clase $\bar{w}$ es un problema de optimización real cuya dimensión, $f * k$, depende del número de atributos y del número de clases. Uno de los algoritmos evolutivos más potentes en la resolución de problemas reales mono-objetivo no lineales y no convexos es CMA-ES [47], que es una estrategia evolutiva de adaptación de la matriz de covarianza. Las principales ventajas de CMA-ES residen en sus propiedades invariantes, conseguidas con un cuidadoso diseño de los operadores de selección y genéticos, y en su eficiente autoadaptación en la distribución de la mutación. CMA-ES no necesita una complicada parametrización puesto que la búsqueda de los mejores parámetros es considerada parte del diseño del algoritmo, y no de su aplicación. Por lo tanto, nuestro modelo emplea el algoritmo CMA-ES para optimizar los valores reales del vector de pesos atributo–clase.

Desempeño con datos no balanceados

El problema del aprendizaje con datos no balanceados en un reto relativamente reciente que ha atraído la atención de la investigación [48]. El problema se debe a que la representación de los datos no es equitativa, existen grandes diferencias en la cantidad de datos que representan a cada una de las clases. Uno de los graves problemas extraídos de las conclusiones del artículo de Peng et al. [68] es el mal funcionamiento de su algoritmo gravitatorio en este tipo de conjuntos de datos no balanceados. Este problema sería aumentado en nuestro algoritmo, puesto que todas las instancias de entrenamiento participan en la decisión de la clase de una nueva instancia. La atracción gravitatoria de las instancias de las clases minoritarias sería eclipsada por aquellas de las clases mayoritarias. Por lo tanto, la gravedad de una instancia hacia una clase debe ser ponderada con respecto al número de instancias de esa clase y el número total de instancias. De esta forma, se consigue un balanceo y compromiso entre las fuerzas gravitatorias de las clases minoritarias y mayoritarias. Finalmente, la función de atracción gravitatoria de una instancia $\bar{x}$ a una clase c se define como:

$$g(\bar{x}, c) = (1 - \frac{n_c - 1}{n}) \cdot \sum_{i=1}^{n} \frac{1}{d(\bar{x}_i, \bar{x}, c)^2} \quad | \; \bar{x}_i \in c$$

donde n_c es el número de instancias de la clase c y n es el número total de instancias del conjunto de datos.

Estudio experimental del modelo IGCA

Esta sección describe los detalles de los experimentos realizados en diferentes conjuntos de datos para evaluar las capacidades de la propuesta y compararla con otros métodos de clasificación. A continuación se presentan los conjuntos de datos empleados, las medidas de desempeño y los algoritmos utilizados para la comparación, la descripción de la metodología experimental y las pruebas estadísticas utilizadas para la validación de los resultados [19].

Conjuntos de datos

Los conjuntos de datos utilizados en los experimentos han sido seleccionados de la página web de repositorio de KEEL [3] y son muy variados en su grado de complejidad, el número de clases, el número de atributos, y el número de instancias. El número de clases es de hasta 11, el número de atributos entre dos y 60, y el número de instancias varía de 148 a 12.690. La Tabla 11.1 resume la información acerca de estos conjuntos de datos. Los conjuntos de datos se dividen en 10 particiones siguiendo el método de validación cruzada 10-fold cross validation [53, 82] y están disponibles para facilitar las comparaciones futuras con otros métodos.

Tabla 11.1: Complejidad de los conjuntos de datos

Dataset	#Instancias	#Atributos	#Clases
Balance	625	4	3
Banana	5300	2	2
Bupa	345	6	2
Car	1728	6	4
Dermatology	366	34	6
Ecoli	336	7	8
Flare	1066	11	6
German	1000	20	2
Glass	214	9	7
Haberman	306	3	2
Hayes-Roth	160	4	3
Heart	270	13	2
Hepatitis	155	19	2
Ionosphere	351	33	2
Iris	150	4	3
Lymphography	148	18	4
Monk-2	432	6	2
New-thyroid	215	5	3
Nursery	12690	8	5
Page-blocks	5472	10	5
Phoneme	5404	5	2
Pima	768	8	2
Sonar	208	60	2
Tae	151	5	3
Thyroid	7200	21	3
Tic-tac-toe	958	9	2
Vehicle	846	18	4
Vowel	990	13	11
Wine	178	13	3
Yeast	1484	8	10

Métricas

Existen muchas medidas para evaluar la calidad de los clasificadores. La selección de las más apropiadas depende del tipo del problema de clasificación al que nos enfrentemos [75]. Las métricas se basan en los valores de la matriz de confusión, donde la suma de cada columna representa el número de instancias predichas para una clase, mientras que la suma de cada fila representa el número verdadero de instancias de dicha clase.

La métrica estándar de evaluación de un clasificador es la exactitud (accuracy), que es el número de predicciones correctas con respecto al número total de predicciones. Sin embargo, la exactitud puede dar lugar a valoraciones no del todo correctas, especialmente cuando se trabaja con datos no balanceados, puesto que un clasificador que únicamente clasifique como clase mayoritaria, ignorando la clase minoritaria, obtendrá un resultado de exactitud muy elevado, pero probablemente no es lo que se desea, sino que resulta más interesante obtener mejores predicciones en las instancias de la clase minoritaria. Los problemas del mundo real tratan habitualmente con datos no balanceados. Por lo tanto, la evaluación del clasificador necesita otro criterio mejor que la exactitud para evitar este tipo de problemas.

El coeficiente kappa de Cohen [8] es una medida alternativa a la exactitud que penaliza las predicciones poco fundamentadas. El coeficiente kappa evalúa el mérito del clasificador, es decir, que los aciertos puedan ser verdaderamente atribuidos al clasificador, y no ser meros aciertos de predicciones al azar. El estadístico del coeficiente de kappa se valora en el rango desde -1 (ninguna concordancia), pasando por 0 (clasificación aleatoria), hasta 1 (concordancia total). Se calcula empleando los valores de la matriz de confusión de la siguiente forma:

$$Kappa = \frac{N \displaystyle\sum_{i=1}^{k} x_{ii} - \sum_{i=1}^{k} x_{i.} x_{.i}}{N^2 - \displaystyle\sum_{i=1}^{k} x_{i.} x_{.i}}$$

donde x_{ii} es el número de casos en la diagonal de la matriz de confusión (predicciones correctas), N es el número de instancias, y $x_{.i}$, $x_{i.}$ son la suma de los valores de las columnas y filas, respectivamente. El coeficiente kappa de Cohen también penaliza las predicciones mayoritarias, lo que le hace una medida apropiada para evaluar clasificadores con datos no balanceados. El coeficiente kappa es muy útil en problemas multiclase, evaluando la exactitud del clasificador, y a la vez, compensando las predicciones correctas aleatorias.

Por otro lado, existen otras características interesantes que observar en el comportamiento de los algoritmos evolutivos, tales como estudiar la convergencia de la solución al problema a lo largo de las generaciones, puesto que los algoritmos evolutivos tienen una tendencia a converger a soluciones locales en muchos problemas. El algoritmo CMA-ES incorpora técnicas para mejorar la convergencia, especialemente bajo malas definiciones de funciones objetivo, y obtiene resultados altamente competitivos, con buenas soluciones locales y globales. Por lo tanto, en la sección 11.5.5 se estudia la convergencia del modelo a la solución a lo largo de las generaciones.

Configuración de la experimentación

En esta sección se describe la configuración de la experimentación, los algoritmos empleados en la comparativa y sus parámetros, que se han obtenido de la herramienta KEEL [4]. En total, 6 algoritmos, evolutivos y no evolutivos, han sido seleccionados y comparados con nuestra propuesta con el objetivo de determinar si el funcionamiento del nuestro es competitivo en distintos conjuntos de datos. A continuación, se describen los distintos algoritmos empleados:

- DGC [68]: Algoritmo gravitatorio propuesto por Peng et al. que emplea pesos para describir la importancia de cada atributo en el modelo de clasificación gravitatorio. Emplea cúmulos o partículas para agrupar las instancias más cercanas, con el objetivo de reducir el número de instancias y la complejidad del conjunto de datos. La fuerza gravitatoria de una partícula a cada clase se calcula usando el centro de gravedad o centro de masas del cúmulo y el número de instancias representadas por dicho cúmulo.

- C-SVM [26]: Máquina de vector soporte que mapea los valores de entrada a un espacio de alta dimensionalidad a través de un kernel elegido a priori. En este espacio se construye una solución lineal con la propiedad que proporciona una buena generalización. El tipo de kernel seleccionado para la comparativa es el polinómico, puesto que proporciona mejores resultados que el kernel lineal o RBF.

- SMO [69]: Implementa el algoritmo de optimización mínimo secuencial de Platt's para entrenar a una máquina de vector soporte usando un kernel polinómico o RBF. Esta implementación reemplaza los valores omitidos y transforma los atributos nominales a binarios, y los normaliza. Los problemas multiclase se resuelven mediante clasificación por pares de clases. El tipo de kernel seleccionado para la comparativa es el polinómico, puesto que proporciona mejores resultados que el kernel RBF.

- KNN [62]: El clásico clasificador de los k vecinos más cercanos clasifica una instancia con la clase del mayor número de sus vecinos. El criterio de vecindad se define como las k instancias con menor distancia a la instancia a clasificar. El número de vecinos seleccionado es 3 puesto que proporciona los mejores resultados.

- NNEP [61]: Este método obtiene el diseño de una red neuronal y simultáneamente estima los pesos adecuados del modelo empleando un algoritmo de programación evolutiva. De esta forma, el modelo de red neuronal se obtiene de los datos de entrenamiento y se compara con los datos de test para evaluar su generalización.

- SFLSDE [79]: Este método basado en evolución diferencial genera instancias prototipo artificiales a partir de las verdaderas instancias de entrenamiento. Una vez ajustada la ubucicación de los prototipos, se evalúa mediante un clasificador del vecino más cercano.

Los autores de los modelos establecieron en sus respectivos trabajos diferentes configuraciones de los parámetros de sus métodos. Los parámetros que nosotros hemos empleado en el estudio comparativo son los parámetros óptimos encontrados por los autores en sus respectivos estudios experimentales.

Nuestra propuesta ha sido implementada en el software JCLEC [80] y sus únicos parámetros son los requeridos por el algoritmo CMA-ES. Precisamente, CMA-ES no necesita de una parametrización compleja, puesto que encontrar los valores óptimos se considera como parte del diseño del algoritmo, y no de su aplicación. Para la ejecución de CMA-ES, hemos empleado los valores por defecto y sólo es necesario detallar algunos de ellos a modo informativo. CMA-ES emplea un cromosoma cuya dimensionalidad en nuestro problema es (C_L), resultado del producto del número de clases y del número de atributos. Cada gen del cromosoma representa el peso de un atributo para una clase. Los valores iniciales del cromosoma se establecen a 0,5 y la desviación inicial a 0,3. El criterio de parada es el estancamiento de los valores de la función objetivo por debajo de $1E^{-13}$ y el máximo número de generaciones se sitúa en 500. El tamaño de población se adapta a la complejidad del problema y se calcula como $4 * \log {C_L}^2$. Para CMA-ES, el tamaño de población puede ser libremente elegido, porque previene de una convergencia prematura, incluso para poblaciones pequeñas. Valores de población bajos conducen habitualmente soluciones locales más rápidamente, mientras que poblaciones más grandes pueden conducir al óptimo global. El número de reinicios es dos y el tamaño de la población se mantiene constante.

Todos los experimentos se han repetido con cinco semillas diferentes para los métodos estocásticos, y los resultados medios son los que se muestran en las tablas. Todos los algoritmos se han ejecutado sobre los conjuntos de datos empleando validación cruzada 10-fold cross validation. Los experimentos se han ejecudado en un PC con un procesador Intel core i7 a 2.66 GHz, 12 GB DDR3 de memoria, dos gráficas NVIDIA GTX 480 para acelerar el cálculo de la función objetivo bajo el modelo de programación NVIDIA CUDA. El sistema operativo era GNU/Linux Ubuntu 11.04 64 bit.

Análisis estadístico

Para poder analizar los resultados de los experimentos es necesario realizar algunos tests estadísticos no paramétricos con el objetivo de validar los resultados y las conclusiones [42, 43]. Para evaluar si existen diferencias significativas en los resultados de los distintos algoritmos, primero se realizar el test de Iman y Davenport. Este útil test no paramétrico fue recomendado por Demsar [30], y se aplica para obtener el ranking de los K algoritmos sobre los N conjuntos de datos (en nuestro caso hay siete algoritmos y 30 conjuntos de datos) conforme a una F-distribución. Cuando el test de Iman y Davenport indica que existen diferencias significativas, es necesario realizar un test post hoc, en nuestro caso, el test de Bonferroni–Dunn [31] se emplea para encontrar las diferencias significativas entre los algoritmos de la compartiva múltiple. El test asume que los resultados de dos clasificadores es significativamente distinto si difieren en su ranking al menos un cierto valor que se denomina distancia crítica. Finalmente, el test de pares de Wilcoxon [83] se emplea para evaluar múltiples comparaciones para cada par de algoritmos.

Resultados del modelo IGCA

Esta sección presenta y analiza los resultados de los experimentos. Primero, se exponen los resultados de exactitud y del coeficiente kappa. Posteriormente, se procede a su validación mediante test estadísticos no paramétricos. Finalmente, se muestra el análisis de convergencia de la propuesta y los pesos encontrados por el algoritmo.

Exactitud

La exactitud determina el porcentaje de ejemplos correctamente clasificados por el modelo. La Tabla 11.2 muestra los resultados medios de exactitud para los diferentes conjuntos de datos (filas) y algoritmos (columnas). El algoritmo gravitacional propuesto mejora los resultados de los otros métodos en 14 de los 30 conjuntos de datos y obtiene resultados competitivos en el resto. A continuación, realizaremos algunas apreciaciones para resaltar los resultados de algunos métodos y conjuntos de datos.

Banana es un conjunto de datos artificial (2 clases y 2 atributos) donde las instancias pertenecen a clusters con una forma de banana pero contiene ruido. Las máquinas de vector soporte, C-SVM y SMO, fallan en sus predicciones en este conjunto de datos, prediciendo todos los ejemplos de una única clase.

Hayes-Roth es otro conjunto de datos artificial originalmente creado para evaluar el rendimiento de los clasificadores basados en prototipos. Contiene un atributo generado aleatoriamente, que le añade ruido a los datos. En este caso, el clasificador KNN es el que peor se comporta, puesto que no es capaz de filtrar el ruido de este atributo. Por otro lado, ambos modelos gravitatorios evitan este problema mucho mejor que el resto de los métodos, gracias a la ponderación de los pesos de los atributos, proporcionando unos buenos resultados.

El conjunto de datos de glass identification es un problema del mundo real y procede del servicio de ciencia forense de EEUU, contiene seis tipos diferentes de glass que pueden encontrarse en una escena de crimen. Nuestra propuesta gravitatoria mejora los resultados de las máquinas de vector soporte y mejora levemente los resultados del KNN. Resultados similares se han obtenido con el conjunto de datas Sonar (Minas vs Rocas), que contiene 60 atributos reales en el rango [0.0, 1.0].

Tabla 11.2: Resultados de exactitud mediante 10-fold cross-validation

Data set	IGCA	DGC	CSVM	SMO	KNN	NNEP	SFLSDE
Balance	0.8966	0.8999	0.9168	0.8790	0.8337	**0.9669**	0.8789
Banana	**0.8952**	0.8931	0.5517	0.5517	0.8864	0.7411	0.8789
Bupa	0.6744	0.6527	0.7014	0.5789	0.6066	**0.7240**	0.6225
Car	**0.9523**	0.9126	0.8519	0.9385	0.9231	0.9126	0.8474
Contraceptive	0.4945	0.4954	0.5126	0.5116	0.4495	**0.5318**	0.4476
Dermatology	0.9544	0.9170	0.9635	**0.9718**	0.9690	0.9403	0.9541
Ecoli	**0.8217**	0.7672	0.7592	0.7709	0.8067	0.7324	0.7921
German	0.7322	0.7020	0.7360	**0.7450**	0.6960	0.7313	0.7157
Glass	**0.7036**	0.6893	0.6259	0.5924	0.7011	0.6319	0.6679
Haberman	0.7171	0.7277	0.7287	0.7353	0.7058	**0.7489**	0.7100
Hayes-Roth	**0.8400**	0.7738	0.6062	0.5271	0.2500	0.6792	0.6479
Heart	**0.8452**	0.8119	0.8444	0.8444	0.7741	0.8173	0.8136
Hepatitis	0.8628	0.8343	0.8356	**0.8900**	0.8251	0.8431	0.8514
Ionosphere	**0.9311**	0.6724	0.8803	0.8889	0.8518	0.9165	0.8765
Iris	0.9533	0.9533	**0.9667**	0.9600	0.9400	0.9444	0.9444
Lymphography	0.8140	0.8033	0.8398	**0.8477**	0.7739	0.7693	0.7725
Monk-2	**0.9995**	0.9982	0.8061	0.8061	0.9629	0.9931	0.8910
New-thyroid	0.9786	0.8684	**0.9816**	0.9024	0.9537	0.9725	0.9663
Nursery	**0.9696**	0.9378	0.7407	0.9313	0.9254	0.9037	0.7084
Page-blocks	0.9508	0.9268	**0.9671**	0.9270	0.9591	0.9476	0.9476
Phoneme	0.8718	0.8471	0.7733	0.7734	**0.8849**	0.7903	0.8164
Pima	0.7451	0.6662	**0.7710**	**0.7710**	0.7319	0.7692	0.7388
Sonar	**0.8487**	0.7694	0.7726	0.7729	0.8307	0.7731	0.7948
Tae	**0.6715**	0.6709	0.5504	0.5175	0.4113	0.5461	0.5553
Thyroid	**0.9704**	0.9256	0.9332	0.9380	0.9389	0.9424	0.9351
Tic-tac-toe	0.8549	0.6906	0.7045	**0.9833**	0.7756	0.7770	0.7665
Vehicle	0.7116	0.6572	**0.7990**	0.7389	0.7175	0.6745	0.6333
Vowel	**0.9824**	0.9788	0.7970	0.6923	0.9778	0.4431	0.5542
Wine	0.9731	0.9706	0.9438	**0.9775**	0.9549	0.9625	0.9548
Yeast	**0.5926**	0.5151	0.5552	0.5714	0.5317	0.5180	0.5814
Avg. values	**0.8403**	0.7976	0.7805	0.7845	0.7850	0.7881	0.7755
Avg. ranks	**2.1833**	4.6667	3.9667	3.6667	4.6000	4.0167	4.9000

Coeficiente kappa de Cohen

El coeficiente kappa de Cohen [8] es una medida alternativa a la exactitud que compensa los aciertos aleatorios y evalúa el mérito atribuible al clasificador. El estadístico del coeficiente de kappa se valora en el rango desde -1 (ninguna concordancia), pasando por 0 (clasificación aleatoria), hasta 1 (concordancia total). La Tabla 11.3 muestra los resultados medios del coeficiente kappa obtenidos para los diferentes conjuntos de datos (filas) y algoritmos (columnas). El modelo gravitatorio

Tabla 11.3: Resultados de coeficiente kappa mediante 10-fold cross-validation

Data set	IGCA	DGC	CSVM	SMO	KNN	NNEP	SFLSDE
Balance	0.8083	0.8144	0.8603	0.7756	0.7004	**0.9417**	0.7806
Banana	**0.7873**	0.7825	0.0000	0.0000	0.7699	0.4585	0.7543
Bupa	0.3076	0.2220	0.3699	0.0000	0.1944	**0.4180**	0.2211
Car	**0.8981**	0.8025	0.6547	0.8673	0.8264	0.8093	0.6572
Contraceptive	0.1994	0.1952	0.2402	0.2391	0.1358	**0.2670**	0.1484
Dermatology	0.9425	0.8939	0.9540	**0.9644**	0.9608	0.9250	0.9423
Ecoli	**0.7498**	0.6585	0.6621	0.6729	0.7300	0.6179	0.7116
German	0.2812	0.0092	0.2748	**0.3489**	0.2194	0.2958	0.2401
Glass	0.5834	0.5548	0.4718	0.4037	**0.5887**	0.4791	0.5400
Haberman	0.0558	0.0315	0.0085	0.0000	0.1362	**0.2275**	0.1086
Hayes-Roth	**0.7468**	0.6294	0.3851	0.2523	-0.2326	0.4960	0.4278
Heart	**0.6826**	0.6094	0.6809	0.6818	0.5420	0.6278	0.6206
Hepatitis	0.5512	0.2000	0.3545	**0.6081**	0.4683	0.4351	0.4976
Ionosphere	**0.8487**	0.1142	0.7255	0.7459	0.6494	0.8137	0.7233
Iris	0.9300	0.9300	**0.9500**	0.9400	0.9100	0.9167	0.9167
Lymphography	0.6289	0.6026	0.6954	**0.7027**	0.5507	0.5576	0.5710
Monk-2	**0.9991**	0.9963	0.6114	0.6114	0.9254	0.9862	0.7806
New-thyroid	0.9544	0.6566	**0.9622**	0.7576	0.8957	0.9413	0.9265
Nursery	**0.9551**	0.9083	0.6146	0.8991	0.8907	0.8575	0.5732
Page-blocks	0.7152	0.4457	**0.8138**	0.4361	0.7655	0.6731	0.6705
Phoneme	0.6898	0.5889	0.4411	0.4406	**0.7172**	0.4971	0.5548
Pima	0.4063	0.0702	**0.4686**	0.4602	0.3892	0.4634	0.3978
Sonar	**0.6943**	0.5187	0.5374	0.5406	0.6554	0.5406	0.5869
Tae	**0.5062**	0.5049	0.3263	0.2774	0.1171	0.3209	0.3346
Thyroid	**0.7709**	-0.0002	0.1777	0.2867	0.4012	0.4101	0.2970
Tic-tac-toe	0.6607	0.1472	0.3242	**0.9620**	0.4149	0.4961	0.4339
Vehicle	0.6152	0.5437	**0.7320**	0.6520	0.6233	0.5661	0.5115
Vowel	**0.9807**	0.9767	0.7767	0.6615	0.9756	0.3874	0.5096
Wine	0.9590	0.9552	0.9145	**0.9655**	0.9318	0.9430	0.9313
Yeast	**0.4695**	0.3309	0.4086	0.4327	0.3942	0.3602	0.4566
Avg. values	**0.6793**	0.5231	0.5465	0.5529	0.5749	0.5910	0.5609
Avg. ranks	**2.1500**	4.8167	4.1333	3.9833	4.4000	3.9000	4.6167

propuesto obtiene los mejores resultados de kappa en 13 de los 30 conjuntos de datos y obtiene resultados competitivos en el resto. A continuación, realizaremos algunas apreciaciones para resaltar los resultados de algunos métodos y conjuntos de datos.

Anteriormente hemos visto como las máquinas de vector soporte eran incapaces de realizar predicciones sobre el conjunto de datos banana, clasificando a todas las instancias como de una única clase. El coeficiente kappa es capaz de detectar este comportamiento y lo penaliza, otorgándole un valor de 0.0. Comportamientos similares se detectan con SMO y los conjuntos de datos Bupa y Haberman. También, mencionamos el mal funcionamiento del método KNN sobre el conjunto de datos Hayes-Roth, debido a un atributo ruidoso. Es interesante ver como el kappa le otorga en este caso una valoración negativa, es decir, las predicciones son realmente completamente erróneas y peores que si fuesen aleatorias.

Otra apreciación interesante con respecto al kappa se encuentra con el conjunto de datos thyroid, donde los resultados de exactitud de la Tabla 11.2 no sugerían una diferencia de comportamiento importante entre los resultados de los métodos. Sin embargo, el kappa sí indica que existen diferencias considerables. DGC obtiene 0.9256, C-SVM 0.9332, y nuestro modelo 0.9704 en exactitud, pero obtienen -0.0002, 0.1777, y 0.7709 para el kappa, respectivamente. Estas diferencias significativas se deben al elevado ratio de desbalanceo en este conjunto de datos, en el que los clasificadores habitualmente fallan la clase minoritaria. Este comportamiento se agrava especialmente en el método DGC, que sufría severamente en problemas no balanceados.

Análisis estadístico

En este apartado se muestran los resultados de los tests estadísticos no paramétricos descritos en la sección 11.4.4.

El estadístico de Iman y Davenport (distribuído conforme a una F-distribución con seis y 174 grados de libertad) reporta un valor de 6.3461 para la exactitud. El test establece un valor de F-distribución igual a 2.1510 para un nivel de significancia de alpha = 0.05. Este valor es menor que el valor crítico reportado de 6.3461. Entonces, el test rechaza la hipótesis nula y por lo tanto se puede decir que existen diferencias significativas entre los resultados de exactitud de los algoritmos. Concretamente, el p-vale reportado por el test de Iman y Davenport es de $4{,}667E^{-6}$.

La Figura 11.1 muestra la aplicación del test de Bonferroni–Dunn a la exactitud con alpha = 0.05, cuya distancia crítica es de 1.4714. Esta gráfica representa los valores de los rankings de los algoritmos. La distancia crítica se representa mediante una barra horizontal gruesa. Los valores que exceden esta línea son algoritmos que obtienen resultados significativamente diferentes que los del algoritmo de control, que en nuestro caso es el modelo gravitatorio propuesto. De esta forma, se evalúan las diferencias de los otros métodos frente a nuestra propuesta. Los algoritmos situados a la derecha del intervalo crítico son métodos significativamente peores que el algorimo de control. Observando la figura, todos los demás métodos obtienen resultados peores y significamente diferentes que nuestra propuesta. SMO obtiene los segundos mejores resultados de ranking, pero aun así, se sitúa más allá de la distancia crítica.

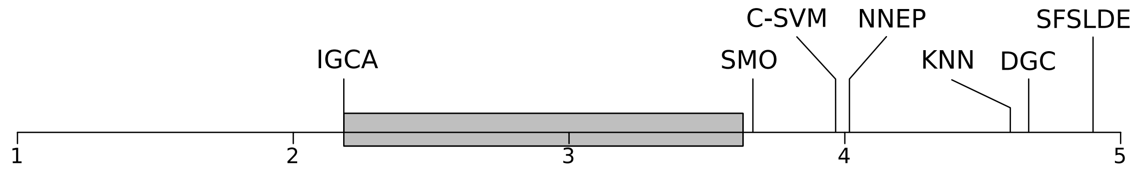


Figura 11.1: Test de Bonferroni–Dunn para la exactitud

La Tabla 11.4 muestra los resultados del test de Wilcoxon para la exactitud con el objetivo de evaluar múltiples comparaciones entre cada par de algoritmos. SMO es el segundo de los mejores métodos, pero comparado 1 vs 1 con la propuesta obtiene resultados significativamente peores con un p-value de 0.0113. Por otro lado, SFLSDE es el peor de todos los métodos y nunca obtiene resultados de exactitud mejores que nuestra propuesta. El test reporta un p-value en este caso de $1{,}862E^{-9}$.

Tabla 11.4: Test de Wilcoxon para la exactitud

IGCA vs	R^+	R^-	p-value
DGC	419.0	16.0	6.296E-7
C-SVM	367.0	98.0	0.004664
SMO	354.0	111.0	0.011304
KNN	441.0	24.0	1.419E-6
NNEP	389.0	76.0	7.978E-4
SFLSDE	465.0	0.0	1.862E-9

El estadístico de Iman y Davenport (distribuído conforme a una F-distribución con seis y 174 grados de libertad) reporta un valor de 5.7867 para el coeficiente kappa. El test establece un valor de F-distribución igual a 2.1510 para un nivel de significancia de alpha = 0.05. Este valor es menor que el valor crítico reportado de 5.7867. Entonces, el test rechaza la hipótesis nula y por lo tanto se puede decir que existen diferencias significativas entre los resultados del coeficiente kappa de los algoritmos. Concretamente, el p-vale reportado por el test de Iman y Davenport es de $1{,}621E^{-5}$.

La Figura 11.2 muestra la aplicación del test de Bonferroni–Dunn al coeficiente kappa con alpha = 0.05, cuya distancia crítica es de 1.4714. De forma similar a la exactitud, esta gráfica muestra los rankings y distancias entre los algoritmos. Los algoritmos a la derecha de la distancia crítica son significativamente peores que el algoritmo de control. Observando la figura, todos los métodos obtienen resultados significativamente peores que nuestro modelo. NNEP en este caso obtiene el segundo mejor ranking, pero se encuentra fuera del intervalo crítico.

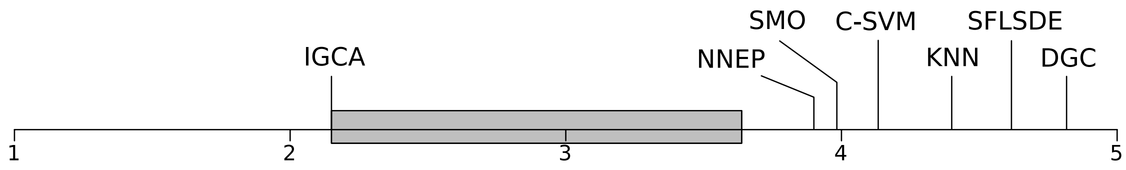


Figura 11.2: Test de Bonferroni–Dunn para el kappa

La Tabla 11.5 muestra los resultados del test de Wilcoxon para el coeficiente kappa con el objetivo de evaluar múltiples comparaciones entre cada par de algoritmos. SMO es el segundo mejor método, pero nuestra propuesta supera a esta máquina de vector soporte con un p-value mejor que 0.01, es decir, existen diferencias significativas entre los dos algoritmos con un nivel de confianza superior al 99 %. Por otro lado, DGC es el peor método comparado con nuestra propuesta, y el test reporta un p-value de $2{,}048E^{-7}$.

Tabla 11.5: Test de Wilcoxon para el kappa

IGCA vs	R^+	R^-	p-value
DGC	428.0	7.0	7.078E-8
C-SVM	370.0	95.0	0.003744
SMO	363.0	102.0	0.006194
KNN	411.0	54.0	8.856E-5
NNEP	375.0	90.0	0.002560
SFLSDE	451.0	14.0	2.048E-7

Comparativa

En este apartado resumimos los resultados obtenidos de los experimentos. La Tabla 11.6 muestra los resultados de kappa y exactitud tanto en las particiones de entrenamiento como de validación para los diferentes métodos. Nuestra propuesta obtiene los mejores resultados en todos los casos, mejorando un 6 % los resultados de exactitud en entrenamiento y validación, un 10 % los resultados de kappa de entrenamiento y un 12 % los resultados de kappa de validación. Además, las desviaciones de los resultados son mejores para nuestro algoritmo. Los resultados del método KNN son similares en entrenamiento y validación, como era esperado, pero las mayores diferencias entre entrenamiento y validación ocurren con el método de generación de prototipos SFLSDE.

Tabla 11.6: Comparativa y resumen de resultados

Algoritmo	Exactitud		Kappa	
	Train	Test	Train	Test
IGCA	**0.8821 ± 0.1137**	**0.8403 ± 0.1302**	**0.7602 ± 0.2204**	**0.6793 ± 0.2475**
DGC	0.7926 ± 0.1527	0.7976 ± 0.1360	0.5172 ± 0.3324	0.5231 ± 0.3219
C-SVM	0.8064 ± 0.1438	0.7805 ± 0.1365	0.6010 ± 0.2864	0.5465 ± 0.2708
SMO	0.8035 ± 0.1551	0.7845 ± 0.1543	0.5894 ± 0.2992	0.5529 ± 0.2908
KNN	0.7861 ± 0.1810	0.7850 ± 0.1830	0.5751 ± 0.3045	0.5749 ± 0.3074
NNEP	0.8330 ± 0.1539	0.7881 ± 0.1515	0.6847 ± 0.2417	0.5910 ± 0.2332
SFLSDE	0.8495 ± 0.1215	0.7755 ± 0.1404	0.7061 ± 0.1985	0.5609 ± 0.2309

Análisis de convergencia

En esta sección se analiza la convergencia de los individuos de la población a lo largo de las generaciones para cuatro conjuntos de datos interesantes: dermatology (seis clases, 34 atributos), vowel (11 clases, 13 atributos), sonar (dos clases, 60 atributos), y nursery (cinco clases, ocho atributos). Estos conjuntos de datos han sido seleccionados debido a su alta dimensionalidad en cuanto a número de clases y de atributos, proporcionando un vector de considerable longitud a CMA-ES, cuya dimensionalidad (producto del número de clases por el número de atributos) es 204, 143, 120 y 40, respectivamente. La Figura 11.3 muestra el mejor *fitness* a lo largo de las generaciones. Los individuos de la generación 0 indican los resultados de exactitud iniciales en entrenamiento cuando todos los atributos eran igualmente considerados con el mismo peso. El algoritmo CMA-ES itera encontrando los mejores pesos para cada atributo y clase a lo largo de las generaciones, logrando mejorar el *fitness* de los individuos. CMA-ES logra efectivamente evitar la convergencia prematura y consigue una buena convergencia a un óptimo.

Pesos atributo–clase

En este apartado se muestran los pesos obtenidos por CMA-ES para nuestro modelo gravitatorio con el objetivo de encontrar cuáles son los atributos más relevantes a la hora de clasificar cada clase de los conjuntos de datos. A continuación exponemos algunos comentarios interesantes sobre algunos de los conjuntos de datos mencionados en la sección anterior.

El histograma de la Figura 11.4 es una representación gráfica que muestra la distribución de los pesos aprendidos para los conjuntos de datos dermatology, vowel, sonar y nursery. El número de pesos con valor bajo es especialmente elevado en

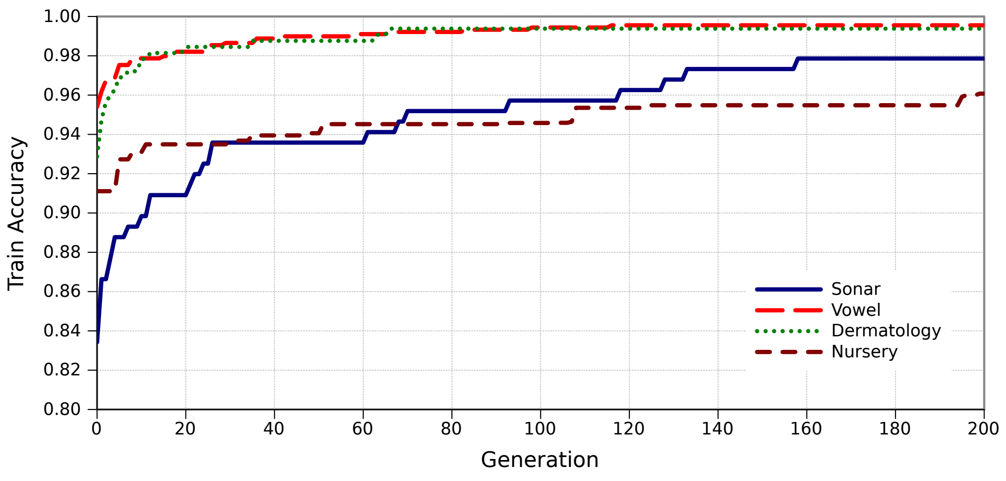
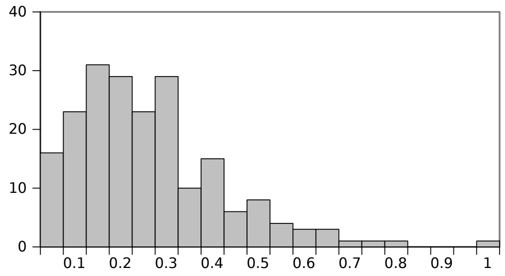
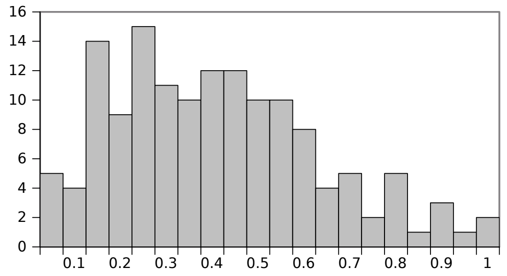


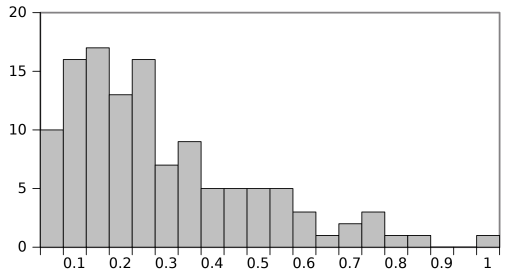
Figura 11.3: Convergencia CMA-ES

dermatology y sonar, luego nuestro método es capaz de ignorar esos atributos al considerarlos irrelevantes. Los pesos para cada atributo y clase se muestran en las Figuras 11.5, 11.6, 11.7, 11.8.

Los atributos más relevantes detectados en sonar son las bandas de frecuencia 4, 44, 46 y 49 para la clase de rocas, y 17 y 19 para la clase de minas, mientras que las bandas de 5, 33, 51, 57 y 58 no se consideran para cualquier clase. En cuanto a vowel, los atributos F0 y F1 son los más relevantes para clasificar a la clase 10 y clase 5, respectivamente. Por otro lado, F5 es la característica menos relevante para la clase 9. Los pesos de nuersery indican que el comportamiento social no se considera excepto para la predicción de la especificación de clase spec_prior, mientras que la salud es más revelante excepto para la clase prioridad. Finalmente, la alta desviación entre las barras del conjunto de datos dermatology indican que el algoritmo ha sido capaz de discriminar muchas de las características, que son ponderadas con valores inferiores a 0,2. Por otro lado, algunas características se han considerado verdaderamente relevantes y sus pesos son superiores a 0,5.


(a) Dermatology


(b) Vowel


(c) Sonar

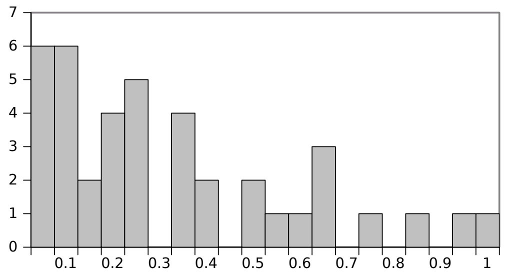

(d) Nursery

Figura 11.4: Histograma de pesos atributo–clase

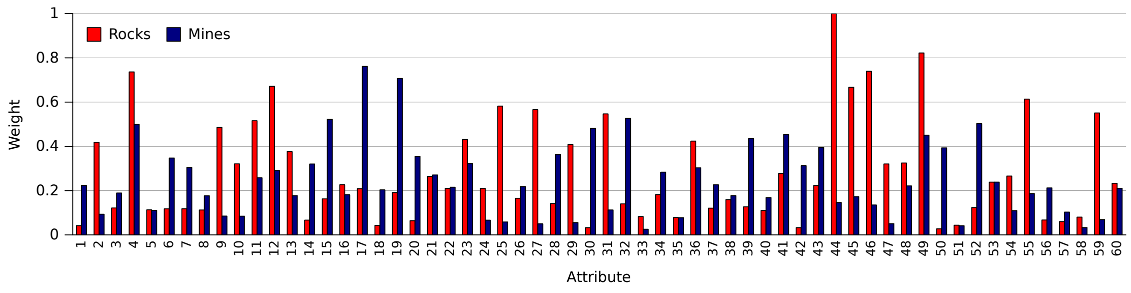


Figura 11.5: Pesos atributo–clase para Sonar

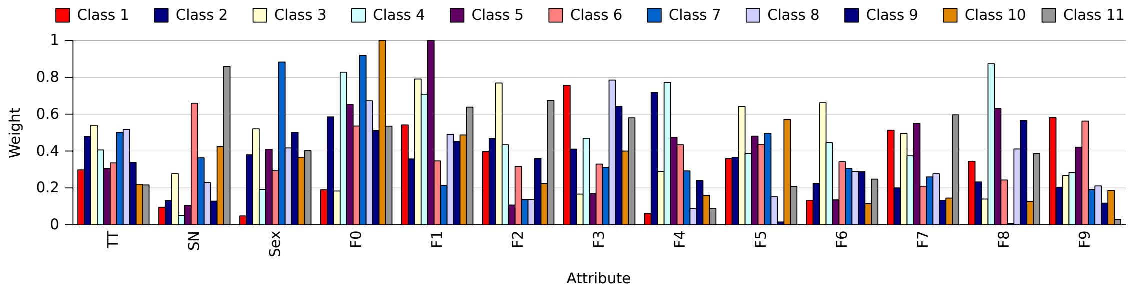


Figura 11.6: Pesos atributo–clase para Vowel

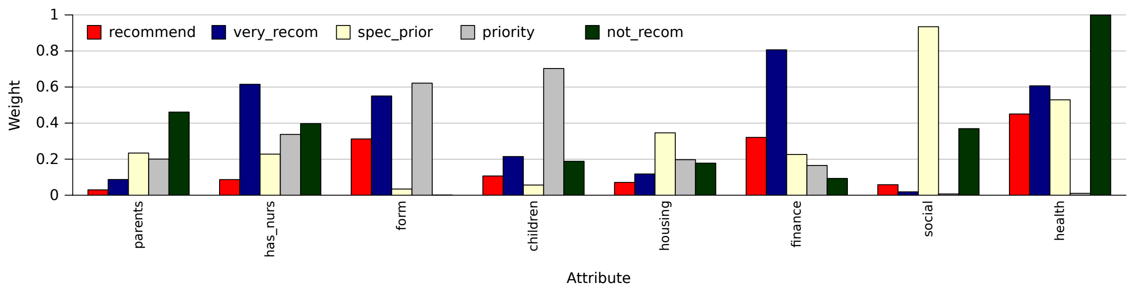


Figura 11.7: Pesos atributo–clase para Nursery

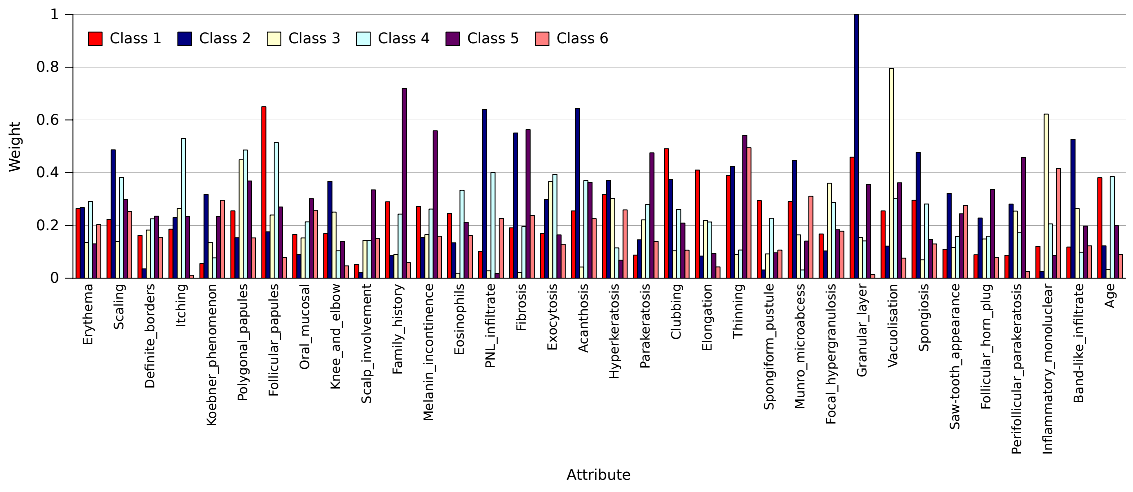


Figura 11.8: Pesos atributo–clase para Dermatology

V

CONCLUSIONES Y TRABAJO FUTURO

12. CONCLUSIONES

La tarea de clasificación ha demostrado ser un verdadero reto con problemas abiertos a nuevas investigaciones para mejorar la eficiencia y exactitud de los clasificadores mediante nuevos modelos.

En la primera parte, hemos abordado el problema del tiempo de ejecución para clasificadores basados en reglas mediante algoritmos evolutivos y hemos aportado un modelo paralelo de evaluación de las reglas mediante GPUs. Los resultados obtenidos con este modelo, aplicado a tres algoritmos de clasificación sobre múltiples conjuntos de datos, aportan aceleraciones de hasta 820 veces empleando una centésima parte del coste económico de un sistema de alto rendimiento tradicional. El modelo ha demostrado una buena escalabilidad frente al número de instancias y permite el uso de múltiples GPUs, aumentando los límites del problema abordables por el modelo. El uso de la GPU permite abordar a bajo coste nuevos retos en clasificación ya que hace viables algunos problemas previamente dados por imposible debido a su elevado coste computacional.

En la segunda parte de este trabajo nos hemos centrado en el problema de la interpretabilidad vs precisión de los clasificadores basados en reglas con el objetivo de maximizar ambos parámetros. La comprensibilidad del clasificador es una cualidad requerida en muchos ámbitos como diagnóstico médico y evaluación de riesgos, pero la interpretabilidad de los sistemas basados en reglas puede crear un problema a la exactitud de los resultados. Dicho problema es complejo, ya que ambos objetivos son contrapuestos. Por un lado, hemos presentado el modelo ICRM, destinado a obtener clasificadores muy interpretables con el menor número de reglas y condiciones posible. Nuestra propuesta obtiene los clasificadores más comprensibles manteniendo los resultados de la calidad de la exactitud en un estudio experimental que incluye algunos de los algoritmos para clasificadores basados en reglas más importantes y recientes en la literatura.

Por otro lado, hemos presentado un modelo de programación genética destinado a optimizar la exactitud del clasificador. Se trata de un modelo Pittsburgh eficiente que obtenga mejores resultados mediante una representación individuo = conjunto de reglas, que permite una mayor flexibilidad en los operadores genéticos en la búsqueda de mejores soluciones y es capaz de tratar con el problema de la competición y cooperación de las reglas dentro del proceso evolutivo.

Finalmente, en la tercera parte nos centramos en modelos de caja negra donde lo importante es obtener la mejor exactitud. En ese sentido, proponemos un modelo basado en gravedad que aprovecha la teoría gravitatoria de Newton para calcular la distancia entre las instancias y de esta forma, seleccionar la clase de una instancia por similitud a la de sus vecinas. El modelo incorpora un proceso de selección de los pesos más apropiados para cada atributo y clase en el cálculo de las distancias mediante CMA-ES, que ha demostrado ser un procedimiento realmente eficaz en la mejora de la exactitud. Nuestro modelo gravitatorio supera las limitaciones y problemas del modelo gravitatorio de Peng et al. [68] en cuanto a la calidad de los resultados y comportamiento en conjuntos de datos no balanceados. Esto ha podido ser evaluado mediante el coeficiente kappa de Cohen, que aporta una medida de la calidad de los resultados de la clasificación que tiene en cuenta el desbalanceo entre clases. El estudio experimental demuestra el buen funcionamiento de nuestra propuesta frente a otros modelos reconocidos de clasificación, incluyendo SMVs y ANNs.

13. TRABAJO FUTURO

El trabajo realizado y el bagaje adquirido a lo largo del último año en el máster ha abierto nuevos problemas y oportunidades para abordar con nuevos modelos de clasificación con objetivo de realizar la tesis doctoral.

El uso de GPUs para acelerar el desempeño de modelos de aprendizaje automático ha demostrado ser muy eficaz, prueba de ello es la tendencia de artículos que abordan dicha temática. En los próximos años las GPUs se aplicarán para resolver de forma más eficiente todo tipo de problemas y además permitirá abordar nuevos problemas. La escalabidad de los modelos con GPUs será clave para ello, por lo tanto será necesario tener en cuenta el diseño eficiente de sistemas de alto rendimiento con múltiples GPUs distribuidas en múltiples máquinas interconectadas. De esta forma se podrán resolver de una manera más intuitiva y eficiente nuevos problemas, como los algoritmos evolutivos con múltiples poblaciones distribuídas en islas, que evolucionan independientemente y cada cierto tiempo migran individuos entre ellas.

Los sistemas basados en reglas para clasificación están muy explotados [33] y ya existen buenas soluciones para tratar con los objetivos de la interpretabilidad y exactitud [22]. Por ello, será necesario abordar nuevos problemas más específicos y proponer nuevos modelos que los resuelvan, tales como su comportamiento frente a datos no balanceados [48], tanto en clasificación binaria como multiclase, reglas para small disjuncts [35], data streaming [1], aprendizaje semisupervisado [23], etc.

El modelo gravitatorio permite ser extendido fácilmente al dominio de la clasificación multi-etiqueta, ya que dicho modelo proporciona la atracción gravitatoria que sufre una instancia para cada clase. La clasificación clásica toma una única etiqueta como clase, a partir de la clase con mayor atracción gravitatoria. Sin embargo, en el dominio multi-etiqueta, las instancias pueden pertenecer a varias clases, de forma que también se podría tomar las etiquetas de las clases con las siguientes fuerzas gravitatorias mayores. Ésta es una de las aplicaciones del modelo gravitatorio que actualmente estamos estudiando y en la que esperamos obtener buenos resultados.

BIBLIOGRAFÍA

[1] C. Aggarwal, editor. *Data Streams – Models and Algorithms*. Springer, 2007.

[2] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

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

[4] J. Alcalá-Fdez, L. Sánchez, S. García, M. del Jesus, S. Ventura, J. Garrell, J. Otero, C. Romero, J. Bacardit, V. Rivas, J. Fernández, and F. Herrera. KEEL: A Software Tool to Assess Evolutionary Algorithms for Data Mining Problems. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 13:307–318, 2009.

[5] R.M. Axelrod. *The Complexity of Cooperation: Agent-based Models of Competition and Collaboration*. Princeton University Press, 1997.

[6] J. Bacardit and N. Krasnogor. Performance and efficiency of memetic pittsburgh learning classifier systems. *Evolutionary Computation*, 17(3):307–342, 2009.

[7] N. Barakat and A. P. Bradley. Rule extraction from support vector machines: A review. *Neurocomputing*, 74(1-3):178–190, 2010.

[8] A. Ben-David. Comparison of classification accuracy using cohen's weighted kappa. *Expert Systems with Applications*, 34(2):825–832, 2008.

[9] E. Bernadó-Mansilla and J. M. Garrell. Accuracy-based learning classifier systems: Models and analysis and applications to classification tasks. *Evolutionary Computation*, 11(3):209–238, 2003.

[10] C. C. Bojarczuk, H. S. Lopes, A. A. Freitas, and E. L. Michalkiewicz. A constrained-syntax genetic programming system for discovering classification

rules: application to medical data sets. *Artificial Intelligence in Medicine*, 30(1):27–48, 2004.

[11] C. Campbell. Kernel methods: A survey of current techniques. *Neurocomputing*, 48:63–84, 2002.

[12] A. Cano, J. M. Luna, and S. Ventura. High performance evaluation of evolutionary-mined association rules on gpus. *Journal of Supercomputing*, 66(3):1438–1461, 2013.

[13] A. Cano, J. M. Luna, A. Zafra, and S. Ventura. A classification module for genetic programming algorithms in jclec. *Journal of Machine Learning Research*, 16:491–494, 2015.

[14] A. Cano, J.L. Olmo, and S. Ventura. Programación automática con colonias de hormigas multi-objetivo en gpus. In *IX Congreso Espa nol sobre Metaheurísticas and Algoritmos Evolutivos y Bioinspirados(MAEB)*, pages 288–297, 2013.

[15] A. Cano, A. Zafra, and S. Ventura. Solving classification problems using genetic programming algorithms on GPUs. In *Proceedings of the 5th International Conference on Hybrid Artificial Intelligent Systems (HAIS). Lecture Notes in Computer Science*, volume 6077 LNAI, pages 17–26, 2010.

[16] A. Cano, A. Zafra, and S. Ventura. A parallel genetic programming algorithm for classification. In *Proceedings of the 6th International Conference on Hybrid Artificial Intelligent Systems (HAIS). Lecture Notes in Computer Science*, volume 6678 LNAI, pages 172–181, 2011.

[17] A. Cano, A. Zafra, and S. Ventura. Speeding up the evaluation phase of GP classification algorithms on GPUs. *Soft Computing*, 16(2):187–202, 2012.

[18] A. Cano, A. Zafra, and S. Ventura. An interpretable classification rule mining algorithm. *Information Sciences*, 240:1–20, 2013.

[19] A. Cano, A. Zafra, and S. Ventura. Weighted data gravitation classification for standard and imbalanced data. *IEEE Transactions on Cybernetics*, 43(6):1672–1687, 2013.

[20] A. Cano, A. Zafra, and S. Ventura. Parallel evaluation of pittsburgh rule-based classifiers on gpus. *Neurocomputing*, 126:45–57, 2014.

[21] A. Cano, A. Zafra, and S. Ventura. Speeding up multiple instance learning classification rules on gpus. *Knowledge and Information Systems*, 44(1):127–145, 2015.

[22] J.R. Cano, F. Herrera, and M. Lozano. Evolutionary Stratified Training Set Selection for Extracting Classification Rules with trade off Precision-Interpretability. *Data and Knowledge Engineering*, 60(1):90–108, 2007.

[23] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning (Adaptive Computation and Machine Learning)*. MIT Press, 2006.

[24] M. S. Chen, J. Han, and P. S. Yu. Data mining: An overview from a database perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866–883, 1996.

[25] W. W. Cohen. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, 1995.

[26] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[27] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[28] N. L. Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 183–187, 1985.

[29] I. De Falco, A. Della Cioppa, and E. Tarantino. Discovering interesting classification rules with genetic programming. *Applied Soft Computing*, 1(4):257–269, 2001.

[30] J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Machine Learning Research*, 7:1–30, 2006.

[31] O. J. Dunn. Multiple Comparisons Among Means. *Journal of the American Statistical Association*, 56(293):52–64, 1961.

[32] Y. Endo and H. Iwata. Dynamic clustering based on universal gravitation model. *Lecture Notes in Computer Science*, 3558:183–193, 2005.

[33] P. G. Espejo, S. Ventura, and F. Herrera. A Survey on the Application of Genetic Programming to Classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 40(2):121–144, 2010.

[34] U. Fayyad, G. Piatetsky, and P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 1996.

[35] A. Fernández, S. García, and F. Herrera. Addressing the classification with imbalanced data: Open problems and new challenges on class distribution. *Lecture Notes in Computer Science*, 6678:1–10, 2011.

[36] D. Fisch, B. Kühbeck, B. Sick, and S. J. Ovaska. So near and yet so far: New insight into properties of some well-known classifier paradigms. *Information Sciences*, 180(18):3381–3401, 2010.

[37] M. A. Franco, N. Krasnogor, and J. Bacardit. Speeding up the evaluation of evolutionary learning systems using GPGPUs. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1039–1046, 2010.

[38] E. Frank and I.H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the 15th International Conference on Machine Learning*, pages 144–151, 1998.

[39] W. J. Frawley, G. Piatetsky, and C. J. Matheus. Knowledge discovery in databases: An Overview. *AI Magazine*, 13, 1992.

[40] A. A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag New York, Inc., 2002.

[41] S. García, A. Fernández, J. Luengo, and F. Herrera. A Study of Statistical Techniques and Performance Measures for Genetics-based Machine Learning: Accuracy and Interpretability. *Soft Computing*, 13(10):959–977, 2009.

[42] S. García, A. Fernández, J. Luengo, and F. Herrera. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10):2044–2064, 2010.

[43] S. García, D. Molina, M. Lozano, and F. Herrera. A Study on the use of Non-parametric Tests for Analyzing the Evolutionary Algorithms' Behaviour: A Case Study. *Journal of Heuristics*, 15:617–644, 2009.

[44] J. Giblin, D. Marolf, and R. Garvey. Spacetime embedding diagrams for spherically symmetric black holes. *General Relativity and Gravitation*, 36:83–99, 2004.

[45] A. González and R. Perez. Selection of Relevant Features in a Fuzzy Genetic Learning Algorithm. *IEEE Transactions on Systems and Man and and Cybernetics and Part B: Cybernetics*, 31(3):417–425, 2001.

[46] S. U. Guan and F. Zhu. An incremental approach to genetic-algorithms-based classification. *IEEE Transactions on Systems and Man and Cybernetics, Part B*, 35(2):227–239, 2005.

[47] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[48] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.

[49] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51:141–154, 2011.

[50] U. Johansson, R. König, and L. Niklasson. Automatically Balancing Accuracy and Comprehensibility in Predictive Modeling. In *7th International Conference on Information Fusion*, volume 2, pages 1554–1560, 2005.

[51] Kazarlis, S.A. and Papadakis, S.E. and Theocharis, J.B. and Petridis, V. Microgenetic algorithms as generalized hill-climbing operators for GA optimization. *IEEE Transactions on Evolutionary Computation*, 5(3):204–217, 2001.

[52] D. Kirk, W. W. Hwu, and J. Stratton. Reductions and their implementation. Technical report, University of Illinois, Urbana-Champaign, 2009.

[53] R. Kohavi. A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the 14th international joint conference on Artificial intelligence*, volume 2, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[54] I. Kononenko and M. Kukar. *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Horwood Publishing Limited, 2007.

[55] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. In *Proceeding of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 3–24, 2007.

[56] J. R. Koza. *Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems.* Stanford University, Stanford, CA, USA, 1990.

[57] M. Li and Z. Wang Z. A hybrid coevolutionary algorithm for designing fuzzy classifiers. *Information Sciences*, 179(12):1970–1983, 2009.

[58] H. Liu and L. Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491–502, 2005.

[59] B. Mak and T. Munakata. Rule extraction from expert heuristics: A comparative study of rough sets with neural networks and id3. *European Journal of Operational Research*, 136(1):212–229, 2002.

[60] D. Martens, B. Baesens, T. Van Gestel, and J. Vanthienen. Comprehensible Credit Scoring Models using Rule Extraction from Support Vector Machines. *European Journal of Operational Research*, 183(3):1466–1476, 2007.

[61] F. J. Martínez-Estudillo, C. Hervás-Martínez, P. A. Gutiérrez, and A. C. Martínez-Estudillo. Evolutionary product-unit neural networks classifiers. *Neurocomputing*, 72(1-3):548–561, 2008.

[62] G. J. McLachlan. *Discriminant Analysis and Statistical Pattern Recognition.* Wiley-Interscience, 2004.

[63] D. D. Nauck. Measuring Interpretability in Rule-Based Classification Systems. In *Proceedings of IEEE International Conference on Fuzzy Systems*, pages 196–201, 2002.

[64] D. J. Newman and A. Asuncion. UCI machine learning repository, 2007.

[65] A. Palacios, L. Sánchez, and I. Couso. Extending a Simple Genetic Cooperative-Competitive Learning Fuzzy Classifier to Low Quality Datasets. *Evolutionary Intelligence*, 2:73–84, 2009.

[66] M. Paliwal and U.A. Kumar. Neural Networks and Statistical Techniques: A Review of Applications. *Expert Systems with Applications*, 36(1):2–17, 2009.

[67] R.S. Parpinelli, H.S. Lopes, and A.A. Freitas. Data Mining With an Ant Colony Optimization Algorithm. *IEEE Transactions on Evolutionary Computation*, 6(4):321–332, 2002.

[68] L. Peng, B. Peng, Y. Chen, and A. Abraham. Data gravitation based classification. *Information Sciences*, 179(6):809–819, 2009.

[69] J. Platt. *Machines using Sequential Minimal Optimization.* MIT Press, 1998.

[70] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kauffman Publishers, 1993.

[71] D. Richards. Two Decades of Ripple Down Rules Research. *Knowledge Engineering Review*, 24(2):159–184, 2009.

[72] R. L. Rivest. Learning decision lists. *Machine Learning*, 2:229–246, 1987.

[73] S. F. Smith. *A Learning System Based on Genetic Adaptive Algorithms.* Phd thesis, University of Pittsburgh, 1980.

[74] S. F. Smith. *A learning system based on genetic adaptive algorithms.* PhD thesis, Pittsburgh, PA, USA, 1980.

[75] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, 45(4):427–437, 2009.

[76] K. C. Tan, A. Tay, T. H. Lee, and C. M. Heng. Mining multiple comprehensible classification rules using genetic programming. In *Proceedings of the Evolutionary Computation on 2002. CEC '02*, volume 2, pages 1302–1307, 2002.

[77] K. C. Tan, Q. Yu, and J. H. Ang. A coevolutionary algorithm for rules discovery in data mining. *International Journal of Systems Science*, 37(12):835–864, 2006.

[78] K. C. Tan, Q. Yu, C. M. Heng, and T. H. Lee. Evolutionary computing for knowledge discovery in medical diagnosis. *Artificial Intelligence in Medicine*, 27(2):129–154, 2003.

[79] I. Triguero, S. García, and F. Herrera. Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification. *Pattern Recognition*, 44(4):901–916, 2011.

[80] S. Ventura, C. Romero, A. Zafra, J. A. Delgado, and C. Hervás. JCLEC: a Java framework for evolutionary computation. *Soft Computing*, 12:381–392, 2007.

[81] C. Wang and Y. Q. Chen. Improving nearest neighbor classification with simulated gravitational collapse. *Lecture Notes in Computer Science*, 3612:845–854, 2005.

[82] T.S. Wiens, B.C. Dale, M.S. Boyce, and G.P. Kershaw. Three way k-fold cross-validation of resource selection functions. *Ecological Modelling*, 212(3-4):244–255, 2008.

[83] F. Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[84] M. L. Wong and K. S. Leung. *Data Mining Using Grammar-Based Genetic Programming and Applications*. Kluwer Academic Publishers, 2000.

[85] W. E. Wright. Gravitational clustering. *Pattern Recognition*, 9(3):151–166, 1977.

[86] N. Xie and Y. Liu. Review of Decision Trees. In *Proceedings - 3rd IEEE International Conference on Computer Science and Information Technology, ICCSIT*, volume 5, pages 105–109, 2010.

[87] B. Yang, L. Peng, Y. Chen, H. Liu, and R. Yuan. A DGC-based data classification method used for abnormal network intrusion detection. *Lecture Notes in Computer Science*, 4234:209–216, 2006.

[88] X. Yao, Y. Liu, and G. Lin. Evolutionary Programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102, 1999.

[89] X. Yu and M. Gen. *Introduction to Evolutionary Algorithms*. Springer, 2010.

[90] NVIDIA CUDA Zone. Nvidia programming and best practices guide 3.0.

[91] Y. Zong-Chang. A vector gravitational force model for classification. *Pattern Analysis and Applications*, 11(2):169–177, 2008.

APÉNDICES

Solving Classification Problems Using Genetic Programming Algorithms on GPUs

Alberto Cano, Amelia Zafra, and Sebastián Ventura

Department of Computing and Numerical Analysis, University of Córdoba
14071 Córdoba, Spain
{i52caroa,azafra,sventura}@uco.es

Abstract. Genetic Programming is very efficient in problem solving compared to other proposals but its performance is very slow when the size of the data increases. This paper proposes a model for multi-threaded Genetic Programming classification evaluation using a NVIDIA CUDA GPUs programming model to parallelize the evaluation phase and reduce computational time. Three different well-known Genetic Programming classification algorithms are evaluated using the parallel evaluation model proposed. Experimental results using UCI Machine Learning data sets compare the performance of the three classification algorithms in single and multithreaded Java, C and CUDA GPU code. Results show that our proposal is much more efficient.

1 Introduction

Evolutionary Algorithms (EA) are a good method, inspired by natural evolution, to find a reasonable solution for data mining and knowledge discovery [1], but they can be slow at converging and have complex and great dimensional problems. Their parallelization has been an object of intensive study. Concretely, we focus on the Genetic Programming (GP) paradigm.

GP has been parallelized in many ways to take advantage both of different types of parallel hardware and of different features in particular problem domains. Most parallel algorithms during the last two decades deal with implementation in clusters or Massively Parallel Processing architectures. More recently, the studies on parallelization focus on using graphic processing units (GPUs) which provide fast parallel hardware for a fraction of the cost of a traditional parallel system.

The purpose of this paper is to improve the efficiency of GP classification models in solving classification problems. Once it has been demonstrated that the evaluation phase is the one that requires the most computational time, the proposal is to parallelize this phase generically, to be used by different algorithms. An evaluator is designed using GPUs to speed up the performance, receiving a classifier and returning the confusion matrix of that classifier to a database. Three of the most popular GP algorithms are tested using the parallel evaluator proposed. Results greatly speed algorithm performance up to 1000 times with respect to a non-parallel version executed sequentially.

The remainder of this paper is organized as follows. Section 2 provides an overview of GPU architecture and related experiences with Evolutionary Computation in GPUs. Section 3 analyzes GP classification algorithms to find out which parts of the algorithms are more susceptible to performance improvement, and discusses their parallelization and implementation. Section 4 describes the experimental setup and presents computational results for the models benchmarked. Conclusions of our investigation and future research tasks are summarized in Section 5.

2 Overview

In this section, first, the CUDA programming model on GPU will be specified. Then, the more important previous studies of GPU on GP will be described.

2.1 CUDA Programming Model on GPU

The CUDA programming model [13] executes kernels as batches of parallel threads in a Single Instruction Multiple Data (SIMD) programming style. These kernels comprise thousands to millions of lightweight GPU threads for each kernel invocation.

CUDA threads are organized into a two-level hierarchy represented in Figure 1. At the higher one, all the threads in a data-parallel execution phase form a grid. Each call to a kernel initiates a grid composed of many thread groupings, called thread blocks. All the blocks in a grid have the same number of threads, with a maximum of 512. The maximum number of thread blocks is 65535 x 65535, so each device can run up to 65535 x 65535 x 512 = $2 \cdot 10^{12}$ threads.

To properly identify threads, each thread in a thread block has a unique ID in the form of a three-dimensional coordinate, and each block in a grid also has a unique two-dimensional coordinate.

Thread blocks are executed in streaming multiprocessors (SM). A SM can perform zero-overhead scheduling to interleave warps and hide the latency of long-latency arithmetic and memory operations.

There are four different main memory spaces: global, constant, shared and local. GPU's memories are specialized and have different access times, lifetimes and output limitations.

2.2 Related GP Works with GPUS

Several studies have been done on GPUs and Massively Parallel Processing architectures within the framework of evolutionary computation. Concretely, we can cite some studies on Genetic Programming on GPUs [7].

D. Chitty [4] describes the technique of general purpose computing using graphics cards and how to extend this technique to Genetic Programming. The

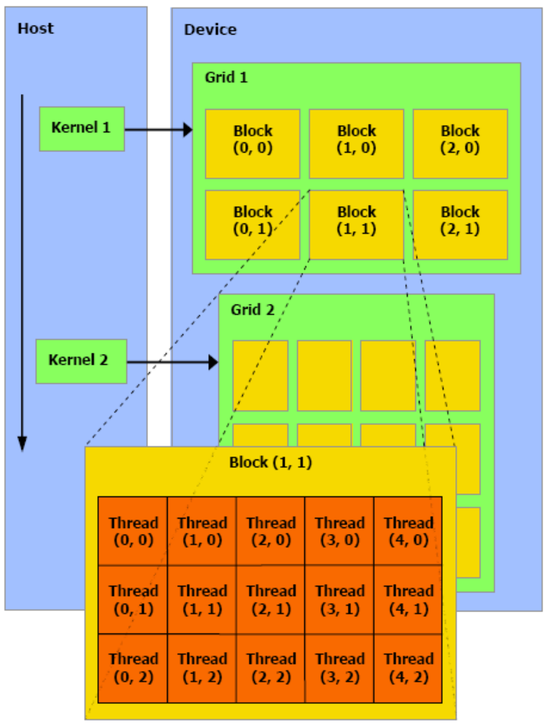


Fig. 1. CUDA threads and blocks

improvement in the performance of Genetic Programming on single processor architectures is also demonstrated. S. Harding [8] goes on to report on how exactly the evaluation of individuals on GP could be accelerated.

Previous research on GPUs have shown evaluation phase speedups for large training case sets like the one in our proposal. Furthermore, D. Robilliard [14] proposes a parallelization scheme to exploit the power of the GPU on small training sets. To optimize with a modest-sized training set, instead of sequentially evaluating the GP solutions parallelizing the training cases, the parallel capacity of the GPU are shared by the GP programs and data. Thus, different GP programs are evaluated in parallel, and a cluster of elementary processors are assigned to each of them to treat the training cases in parallel. A similar technique but using an implementation based on the Single Program Multiple Data (SPMD) model is proposed by W. Landgdon and A. Harrison [12]. The use of SPMD instead of Single Instruction Multiple Data (SIMD) affords the opportunity to achieve increased speedups since, for example, one cluster can interpret the *if* branch of a test while another cluster treats the *else* branch independently. On the other hand, performing the same computation inside a cluster is also possible, but the two branches are processed sequentially in order to respect the SIMD constraint: this is called divergence and is, of course, less efficient.

These reports have helped us to design and optimize our proposal to achieve maximum performance in data sets with different dimensions and population sizes.

3 Genetic Programming Algorithms on GPU

In this section the computation time of the different phases of the GP generational algorithm is evaluated to determine the most expensive part of algorithm. Then, it is specified how parallelize using GPU the evaluation phase of a GP algorithm.

3.1 GP Classification Algorithm

Genetic Programming, introduced by Koza [10], is a learning methodology belonging to the family of EA [17]. Among successful EA implementations, GP retains a significant position due to such valuable characteristics as: its flexible variable length solution representation, the fact that a priori knowledge is not needed about the statistical distribution of the data (data distribution free), data in their original form can be used to operate directly on them, unknown relationships that exist among data can be detected and expressed as a mathematical expression and finally, the most important discriminative features of a class can be discovered. These characteristics convert these algorithms into a paradigm of growing interest both for obtaining classification rules [2],[18], and for other tasks related to prediction, such as feature selection and the generation of discriminant functions.

The evolutionary process of Genetic Programming algorithms [6], similar to other EA paradigms, is represented in Figure 2 and consists of the following steps. The initial population is created randomly and evaluated. For each generation, the algorithm performs a selection of the best parents. This subset of individuals is recombined and mutated by applying different genetic operators,

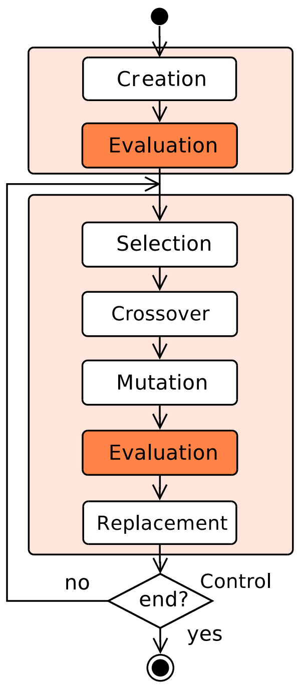


Fig. 2. GP Evolution Model

Table 1. CPU GP classification time test

Phase	Time (ms)	Percentage
Initialization	8647	8.96%
Creation	382	0.39%
Evaluation	8265	8.57%
Generation	87793	91.04%
Selection	11	0.01%
Crossover	13	0.01%
Mutation	26	0.03%
Evaluation	82282	85.32%
Replacement	26	0.03%
Control	5435	5.64%
Total	96440	100 %

thus obtaining offspring. These new individuals must now be evaluated using the fitness function. Different replacement strategies can be employed on parents and offspring so that the next generation population size remains constant. The algorithm performs a control stage in which it terminates if it finds acceptable solutions or the generation count reaches a limit. The consecutive processes of selection of parents, crossover, mutation, evaluation, replacement and control constitute a generation of the algorithm.

Experiments have been conducted to evaluate the computation time of the different phases of the generational algorithm. The experiment using GP algorithms represented in table 1 proves that around 94% of the time is taken up by the evaluation phase. This percentage is mainly linked to the population size and the number of patterns, increasing up to 98% in large problems. Thus, the most significant improvement is obtained by accelerating the evaluation phase, and this is what we do in our proposal. The most computationally expensive phase is evaluation since it involves the test of all individuals generated throughout all the patterns. Each individual's expression must be interpreted or translated into an executable format which is then evaluated for each training pattern. The result of the individual's evaluation throughout all the patterns is used to build the confusion matrix. The confusion matrix allows us to apply different quality indexes to get the individual's fitness.

The evaluation process of individuals within the population consists of two loops, where each individual iterates each pattern and checks if the rule covers that pattern. These two loops make the algorithm really slow when the population or pattern size increases.

3.2 Implementation on GPU

To parallelize the evaluation phase, the designed implementation is meant to be as generic as possible to be employed by different classification algorithms. The implementation receives a classifier and returns the confusion matrix that is used by most algorithms for the calculation of the fitness function.

To take advantage of GPU architecture [15], all individuals are evaluated throughout all the patterns simultaneously. An easy way to do that, is to create a grid of thread blocks sized as follows: one dimension is sized as the number of individuals and the other dimension is sized as the number of patterns in the data set. This organization means that one thread is the evaluation of one individual in one pattern. To achieve full performance, we have to maximize multiprocessor occupancy, so each block represents the evaluation of one individual in 128 patterns. This way, each thread within the block computes one single evaluation, then the size of the second dimension of the grid is the number of patterns divided by 128. This configuration allows up to 65536 individuals and 8.388.608 patterns per GPU and kernel call, large enough for all the data sets tested. Larger populations can be evaluated using several GPUs, up to 4 devices per host or iterating kernel calls.

Before running the evaluator on the GPU, the individuals's rules must be copied to the memory device using the PCI-E bus. Full performance can be obtained by copying the rules into constant memory and pointing all the threads in a warp toward the same rule, resulting in a single memory instruction access. Constant memory provides a 64 KB cache written by the host and read by GPU threads.

The evaluation takes place in two steps. In the first kernel, each thread checks if the antecedent of the rule covers the pattern and the consequent matches the pattern class and stores a value, generally a hit or a miss. This implementation allows reusability for different classification models by changing the value stored depending on whether the antecedent does or does not cover the pattern, and the resulting matches or non-matches of the pattern class.

The kernel function must analyze the expression working with Polish notation, also known as prefix notation. Its distinguishing feature is that it places operators to the left of their operands. If the arity of the operators is fixed, the result is a syntax lacking parentheses or other brackets. While there remain tokens to be analyzed in the individual's expression, it checks what it has to do next by using a stack in order to store numerical values. Finally, we check the top value of the stack. If this value is true, that means the antecedent was true, so depending on the algorithm used, we compare this value to the known class given for the pattern.

The second kernel performs a reduction [5] and counts the results by subsets of 128, to get the total number of hits and misses. Using the total number of hits and misses, the proposed kernel performs the calculation of the confusion matrix on GPU, which could be used by any implementation of GP classification algorithms or even any genetic algorithm. This way, the kernel calculates the fitness of individuals in parallel using the confusion matrix and the quality metrics required by the classification model. Finally, results are copied back to the host memory and set to individuals for the next generation.

4 Experimental Results

Experiments carried out compare the performance of three different GP algorithms in single and multithreaded Java, C and CUDA GPU code. This section explains several experimentation details related with the data sets and the algorithms and then the speedup of the different implementations is compared.

4.1 Experimental Setup

This paper presents an implementation of a GPU GP evaluator for data classification using JCLEC [16]. JCLEC is a software system for Evolutionary Computation (EC) research, developed in Java programming language. It provides a high-level software environment for any kind of Evolutionary Algorithm, with support for Genetic Algorithms, Genetic Programming and Evolutionary Programming. We have selected two databases from the UCI repository for benchmarks, shuttle and poker hand inverse. The shuttle data set contains 9 attributes, 58000 instances and 7 classes. The poker hand inverse data set contains 11 attributes, 10^6 instances and 10 classes.

Experiments were run on a PC equipped with an Intel Core i7 processor running at 2.66GHz with two NVIDIA GeForce 285 GTX video cards equipped with 2GB of GDDR3 video RAM. No overclocking was done for any of the hardware.

Three different Grammar-Guided Genetic-Programming classification algorithms that were proposed in the literature are tested using our evaluator proposal. The evaluation concerning each algorithm is detailed for parallelization purposes.

Falco, Della and Tarantino [9] propose a method to achieve rule fitness by evaluating the antecedent throughout all the patterns within the data set. The adjustment function calculates the difference between the number of examples where the rule does or does not correctly predict the membership of the class and number of examples; if the opposite occurs, then the prediction is wrong. Specifically it measures the number of true positives t_p, false positives f_p, true negatives t_n and false negatives f_n. Finally fitness is expressed as:

$$fitness = I - ((t_p + t_n) - (f_p + f_n)) + \alpha * N \tag{1}$$

where I is the total number of examples from all training, α is a value between 0 and 1 and N is the number of nodes.

Tan, Tay, Lee and Heng [11] propose a fitness function that combines two indicators that are commonplace in the domain, namely sensitivity (Se) and specifity (Sp), defined as follows:

$$Se = \frac{t_p}{t_p + f_n} \qquad Sp = \frac{t_n}{f_p + t_n} \tag{2}$$

Thus, fitness is defined by the product of these two parameters.

$$fitness = Se * Sp \tag{3}$$

The proposal by Bojarczuk, Lopes and Freitas [3] presents a method in which each rule is simultaneously evaluated for all the classes in a pattern. The classifier is formed by taking the best individual for each class generated during the evolutionary process.

GP does not produce simple solutions. The comprehensibility of a rule is inversely proportional to its size. Therefore Bojarczuk defines the simplicity (Sy) and then the fitness of a rule:

$$Sy = \frac{maxnodes - 0.5 * numnodes - 0.5}{maxnodes - 1} \qquad fitness = Se * Sp * Sy \tag{4}$$

4.2 Comparing the Performance of GPU and Other Proposals

The results of the three GP classification algorithms are benchmarked using two UCI data sets that are shown in Tables 2 and 3. The rows represent the

Table 2. Shuttle generation speedup results

		Tan Model				Falco Model				Bojarczuk Model			
	Pop	100	200	400	800	100	200	400	800	100	200	400	800
Speed up	Java	1	1	1	1	1	1	1	1	1	1	1	1
	C1	2,8	3,1	3,2	2,9	5,4	8,1	5,2	5,0	18,8	12,5	11,2	9,5
	C2	5,5	6,1	6,3	5,7	10,6	15,9	10,5	10,1	35,9	24,6	22,1	18,1
	C4	10,1	11,5	12,5	10,7	19,7	30,3	20,5	19,8	65,2	47,3	40,1	33,7
	C8	11,1	12,4	13,4	10,3	19,9	30,1	21,2	20,6	65,7	46,8	40,5	34,6
	GPU	218	267	293	253	487	660	460	453	614	408	312	269
	GPUs	436	534	587	506	785	1187	899	867	1060	795	621	533

Table 3. Poker-I generation speedup results

		Tan Model				Falco Model				Bojarczuk Model			
	Pop	100	200	400	800	100	200	400	800	100	200	400	800
Speed up	Java	1	1	1	1	1	1	1	1	1	1	1	1
	C1	2,7	3,2	3,1	3,0	4,6	5,0	5,6	4,9	5,5	5,7	5,8	4,7
	C2	5,5	6,5	6,7	5,5	9,0	9,8	11,1	9,7	10,6	11,0	11,6	11,3
	C4	10,3	11,1	12,8	10,5	16,8	18,9	21,6	18,9	20,3	20,7	22,5	22,3
	C8	11,2	12,9	14,0	10,2	18,5	20,5	23,3	26,4	21,8	22,0	24,2	23,8
	GPU	155	174	234	221	688	623	648	611	142	147	148	142
	GPUs	288	336	500	439	1275	1200	1287	1197	267	297	288	283

speedup compared to Java execution time. Each column is labeled with the algorithm execution configuration from left to right: Population size, Java single CPU thread, C single CPU thread, C two CPU threads, C four CPU threads, C eight CPU threads, 1 GPU device, 2 GPU devices.

Benchmark results prove the ability of GPUs to solve GP evaluation. Intel i7 quadcore performs linear scalability from 1 to 2 and 4 threads, but not any further. After that point, GPUs perform much better. Their parallelized model allows the time spent on a classification problem to drop from one month to only one hour. Real classification training usually needs dozens of evaluations to get an accurate result, so the absolute time saved is actually a great deal of time. The greatest speedup is obtained with the Falco model which increases performance by a factor up to 1200 over the Java solution and up to 150 over the C single threaded CPU implementation. The speed up obtained is actually similar when considering different population sizes, for example a population of 200 individuals is selected to represent the figures. Fig. 3 and Fig. 4 display the speed up obtained by the different proposals with respect to the sequential Java version in the Shuttle and Poker data sets respectively. Note the progessive

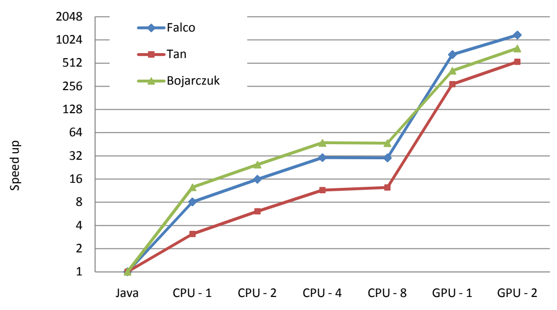


Fig. 3. Shuttle data set speed up

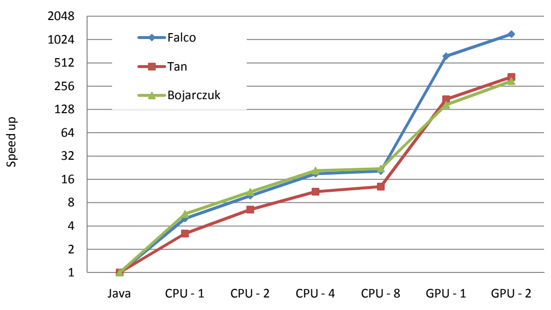


Fig. 4. Poker-I data set speed up

improvement obtained by threading C implementation and the significant increase that occurs when using GPUs.

5 Conclusions

Massive parallelization using the NVIDIA CUDA framework provides a hundred or thousand time speedup over Java and C implementation. GPUs are best for massive multithreaded tasks where each thread does its job but all of them collaborate in the execution of the program.

The CPU solution is lineal and complex. However, the GPU groups the threads into a block; then a grid of blocks is executed in SMs multiprocessors, one per SM. Thus, linearity is approximated by the number of 30-block grids. This implementation allows future scalability for GPUs with more processors. Next the NVIDIA GPU code-named Fermi doubles the number of cores available to 512 and performs up to 1.5 TFLOPs in single precision. It is noteworthy that i7 CPU scores are 2.5 times faster than 3.0 GHz PIV in our benchmarks.

Further work will implement the whole algorithm inside the GPU, so that selection, crossover, mutation, replacement and control phases will be parallelized, reducing data memory transfers between CPU and GPU devices.

Acknowledgments

The authors gratefully acknowledge the financial support provided by the Spanish department of Research under TIN2008-06681-C06-03, P08-TIC-3720 Projects and FEDER funds.

References

1. Freitas, A.A.: Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer, Heidelberg (2002)
2. Tsakonas, A.: A comparison of classification accuracy of four Genetic Programming-evolved intelligent structures. Information Sciences 176(6), 691–724 (2006)

3. Bojarczuk, C.C., Lopes, H.S., Freitas, A.A., Michalkiewicz, E.L.: A constrained-syntax Genetic Programming system for discovering classification rules: application to medical data sets. Artificial Intelligence in Medicine 30(1), 27–48 (2004)
4. Chitty, D.: A data parallel approach to Genetic Programming using programmable graphics hardware. In: GECCO 2007: Proceedings of the Conference on Genetic and Evolutionary Computing, pp. 1566–1573 (2007)
5. Kirk, D., Hwu, W.-m.W., Stratton, J.: Reductions and Their Implementation. University of Illinois, Urbana-Champaign (2009)
6. Deb, K.: A population-based algorithm-generator for real-parameter optimization. Soft Computing 9(4), 236–253 (2005)
7. Genetic Programming on General Purpose Graphics Processing Units, GP GP GPU, `http://www.gpgpgpu.com`
8. Harding, S., Banzhaf, W.: Fast Genetic Programming and artificial developmental systems on GPUS. In: HPCS 2007: Proceedings of the Conference on High Performance Computing and Simulation (2007)
9. De Falco, I., Della Cioppa, A., Tarantino, E.: Discovering interesting classification rules with Genetic Programming. Applied Soft Computing Journal 1(4), 257–269 (2002)
10. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
11. Tan, K.C., Tay, A., Lee, T.H., Heng, C.M.: Mining multiple comprehensible classification rules using Genetic Programming. In: CEC 2002: Proceedings of the Evolutionary Computation on 2002, pp. 1302–1307 (2002)
12. Langdon, W., Harrison, A.: GP on SPMD parallel graphics hardware for mega bioinformatics data mining. Soft Computing. A Fusion of Foundations, Methodologies and Applications 12(12), 1169–1183 (2008)
13. NVIDIA Programming and Best Practices Guide 2.3, NVIDIA CUDA Zone, `http://www.nvidia.com/object/cuda_home.html`
14. Robilliard, D., Marion-Poty, V., Fonlupt, C.: Genetic programming on graphics processing units. Genetic Programming and Evolvable Machines 10(4), 447–471 (2009)
15. Ryoo, S., Rodrigues, C.I., Baghsorkhi, S.S., Stone, S.S., Kirk, D.B., Hwu, W.-m.W.: Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In: PPoPP 2008: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, pp. 73–82 (2008)
16. Ventura, S., Romero, C., Zafra, A., Delgado, J.A., Hervás, C.: JCLEC: A Java framework for evolutionary computation. Soft Computing 12(4), 381–392 (2007)
17. Back, T., Fogel, D., Michalewicz, Z.: Handbook of Evolutionary Computation. Oxford University Press, Oxford (1997)
18. Lensberg, T., Eilifsen, A., McKee, T.E.: Bankruptcy theory development and classification via Genetic Programming. European Journal of Operational Research 169(2), 677–697 (2006)

Soft Comput (2012) 16:187–202
DOI 10.1007/s00500-011-0713-4

FOCUS

Speeding up the evaluation phase of GP classification algorithms on GPUs

Alberto Cano · Amelia Zafra · Sebastián Ventura

Published online: 3 May 2011

Abstract The efficiency of evolutionary algorithms has become a studied problem since it is one of the major weaknesses in these algorithms. Specifically, when these algorithms are employed for the classification task, the computational time required by them grows excessively as the problem complexity increases. This paper proposes an efficient scalable and massively parallel evaluation model using the NVIDIA CUDA GPU programming model to speed up the fitness calculation phase and greatly reduce the computational time. Experimental results show that our model significantly reduces the computational time compared to the sequential approach, reaching a speedup of up to 820×. Moreover, the model is able to scale to multiple GPU devices and can be easily extended to any evolutionary algorithm.

Keywords Evolutionary algorithms · Genetic programming · Classification · Parallel computing · GPU

1 Introduction

Evolutionary algorithms (EAs) are search methods inspired by natural evolution to find a reasonable solution for data mining and knowledge discovery (Freitas 2002). Genetic programming (GP) is a specialization of EAs, where each individual represents a computer program. It is a machine learning technique used to optimize a population of computer programs according to a fitness function that determines the program's ability to perform a task. Recently, GP has been applied to different common data mining tasks such as classification (Espejo et al. 2010), feature selection (Landry et al. 2006), and clustering (De Falco et al. 2004). However, they perform slowly with complex and high-dimensional problems. Specifically, in the case of classification, this slowness is due to the fact that the model must be evaluated according to a fitness function and training data. Many studies, using different approaches (Harding 2010; Schmitz et al. 2003), have focused on solving this problem by improving the execution time of these algorithms. Recently, the use of GPUs has increased for solving high-dimensional and parallelizable problems and in fact, there are already EA models that take advantage of this technology (Franco et al. 2010). The main shortcoming of these models is that they do not provide a general purpose model: they are too specific to the problem domain or their efficiency could be significantly improved.

In this paper, we present an efficient and scalable GPU-based parallel evaluation model to speed up the evaluation phase of GP classification algorithms that overcomes the shortcomings of the previous models. In this way, our proposal is presented as a general model applicable to any domain within the classification task regardless of its complexity and whose philosophy is easily adaptable to any other paradigm.

The proposed model parallelizes the evaluation of the individuals, which is the phase that requires the most computational time in the evolutionary process of EAs. Specifically, the efficient and scalable evaluator model designed uses GPUs to speed up the performance, receiving a classifier and returning the confusion matrix of that

A. Cano · A. Zafra · S. Ventura (✉)
Department of Computing and Numerical Analysis,
University of Córdoba, 14071 Córdoba, Spain
e-mail: sventura@uco.es

A. Cano
e-mail: i52caroa@uco.es

A. Zafra
e-mail: azafra@uco.es

classifier on a database. To show the generality of the model proposed, it is applied to some of the most popular GP classification algorithms and several datasets with distinct complexity. Thus, among the datasets used in experiments, there are some widely used as benchmark datasets on the classification task characterized by its simplicity and with others that have not been commonly addressed to date because of their extremely high complexity when applied to previous models. The use of these datasets of varied complexity allows us to demonstrate the performance of our proposal on any problem complexity and domain. Experimental results show the efficiency and generality of our model, which can deal with a variety of algorithms and application domains, providing a great speedup of the algorithm's performance, of up to 820 times, compared to the non-parallel version executed sequentially. Moreover, the speedup obtained is higher when the problem complexity increases. The proposal is compared with other different GPU computing evolutionary learning system called BioHEL (Franco et al. 2010). The comparison results show that the efficiency obtained is far better using our proposal.

The remainder of this paper is organized as follows. Section 2 provides an overview of previous works related to evolutionary classification algorithms and GPU implementations. Section 3 discusses the GP model and analyzes the computational cost associated with its different phases. Section 4 describes the GPU architecture and the CUDA programming model. Section 5 explains our proposal and its advantages as a scalable and efficient evaluation model. Section 6 describes the experimental study. In Sect. 7, the results will be announced and finally the last section presents the final remarks of our investigation and outlines future research work.

2 Related works

GP has been parallelized in multiple ways to take advantage both of different types of parallel hardware and of different features of particular problem domains. Most of the parallel approaches during the last decades deal with the implementation over CPU machine clusters. More recently, works about parallelization have been focusing on using graphics processing units (GPUs) which provide fast parallel hardware for a fraction of the cost of a traditional parallel system. GPUs are devices with multicore architectures and parallel processor units. The GPU consists of a large number of processors and recent devices operate as multiple instruction multiple data (MIMD) architectures. Today, GPUs can be programmed by any user to perform general purpose computation (GPGPU) (General-Purpose Computation on Graphics Hardware 2010). The use of GPUs has been already studied for speeding up algorithms within the framework of evolutionary computation. Concretely, we can cite some studies about the evaluation process in genetic algorithms and GP on GPUs (Harding 2010).

Previous investigations have focused on two evaluation approaches (Banzhaf et al. 1998): population parallel or fitness parallel and both methods can exploit the parallel architecture of the GPU. In the fitness parallel method, all the fitness cases are executed in parallel with only one individual being evaluated at a time. This can be considered an SIMD approach. In the population parallel method, multiple individuals are evaluated simultaneously. These investigations have proved that for smaller datasets or population sizes, the overhead introduced by uploading individuals to evaluate is larger than the increase in computational speed (Chitty et al. 2007). In these cases there is no benefit in executing the evaluation on a GPU. Therefore, the larger the population size or the number of instances are, the better the GPU implementation will perform. Specifically, the performance of the population parallel approaches is influenced by the size of the population and the fitness case parallel approaches are influenced by the number of fitness cases, i.e., the number of training patterns from the dataset.

Next, the more relevant proposals presented to date are discussed. Chitty et al. (2007) describes the technique of general purpose computing using graphics cards and how to extend this technique to GP. The improvement in the performance of GP on single processor architectures is also demonstrated. Harding and Banzhaf (2007) goes on to report on how exactly the evaluation of individuals on GP could be accelerated; both proposals are focused on population parallel.

Robilliard et al. (2009) proposes a parallelization scheme to exploit the performance of the GPU on small training sets. To optimize with a modest-sized training set, instead of sequentially evaluating the GP solutions parallelizing the training cases, the parallel capacity of the GPU is shared by the GP programs and data. Thus, different GP programs are evaluated in parallel, and a cluster of elementary processors are assigned to each of them to treat the training cases in parallel. A similar technique, but using an implementation based on the single program multiple data (SPMD) model, is proposed by Langdon and Harrison (2008). They implement the evaluation process of GP trees for bioinformatics purposes using GPGPUs, achieving a speedup of around 8×. The use of SPMD instead of SIMD affords the opportunity to achieve increased speedups since, for example, one cluster can interpret the *if* branch of a test while another cluster treats the *else* branch independently. On the other hand, performing the same computation inside a cluster is also possible, but the two branches are processed sequentially in order to respect the SIMD constraint: this is called divergence and is, of course, less efficient. Moreover, Maitre et al. (2009) presented an implementation of a genetic algorithm which performs the evaluation function using a GPU. However, they have a training function instead of a training set, which they run in parallel over different individuals. Classification fitness computation is based on learning from a training set within the GPU device which implies memory occupancy, while other proposals use a mathematical representation function as the fitness function.

Franco et al. (2010) introduce a fitness parallel method for computing fitness in evolutionary learning systems using the GPU. Their proposal achieves speed-ups of up to 52× in certain datasets, performing a reduction function (Hwu 2009) over the results to reduce the memory occupancy. However, this proposal does not scale to multiple devices and its efficiency and its spread to other algorithms or to more complex problems could be improved.

These works are focused on parallellizing the evaluation of multiple individuals or training cases and many of these proposals are limited to small datasets due to memory constraints where exactly GPU are not optimal. By contrast, our proposal is an efficient hybrid population and fitness parallel model, that can be easily adapted to other algorithms, designed to achieve maximum performance solving classification problems using datasets with different dimensions and population sizes.

3 Genetic programming algorithms

This section introduces the benefits and the structure of GP algorithms and describes a GP evolutionary system for discovering classification rules in order to understand the execution process and the time required.

GP, the paradigm on which this paper focuses, is a learning methodology belonging to the family of evolutionary algorithms (Bäck et al. 1997) introduced by Koza (1992). GP is defined as an automated method for creating a working computer program from a high-level formulation of a problem. GP performs automatic program synthesis using Darwinian natural selection and biologically inspired operations such as recombination, mutation, inversion, gene duplication, and gene deletion. It is an automated learning methodology used to optimize a population of computer programs according to a fitness function that determines their ability to perform a certain task. Among successful evolutionary algorithm implementations, GP retains a significant position due to such valuable characteristics as: its flexible variable length solution representation, the fact that a priori knowledge is not needed about the statistical distribution of the data (data distribution free), data in their original form can be used to operate directly on them, unknown relationships that exist among data can be detected and expressed as mathematical expressions, and, finally, the most important discriminative features of a class can be discovered. These characteristics suit these algorithms to be a paradigm of growing interest both for obtaining classification rules (De Falco et al. 2001; Freitas 2002; Tan et al. 2002) and for other tasks related to prediction, such as feature selection (Landry et al. 2006) and the generation of discriminant functions (Espejo et al. 2010).

3.1 GP classification algorithms

In this section we will detail the general structure of GP algorithms before proceeding to the analysis of its computational cost.

Individual representation GP can be employed to construct classifiers using different kinds of representations, e.g., decision trees, classification rules, discriminant functions, and many more. In our case, the individuals in the GP algorithm are classification rules whose expression tree is composed by terminal and non-terminal nodes. A classifier can be expressed as a set of IF-antecedent-THEN-consequent rules, in which the antecedent of the rule consists of a series of conditions to be met by an instance in order to consider that it belongs to the class specified by the consequent.

The rule consequent specifies the class to be predicted for an instance that satisfies all the conditions of the rule antecedent. The terminal set consists of the attribute names and attribute values of the dataset being mined. The function set consists of logical operators (AND, OR, NOT), relational operators ($<, \leq, =, <>, \geq, >$) or interval range operators (IN, OUT). These operators are constrained to certain data type restrictions: categorical, real or boolean. Figure 1 shows an example of the expression tree of an individual.

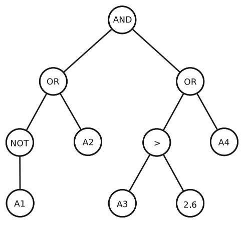


Fig. 1 Example of an individual expression tree

Generational model The evolution process of GP algorithms (Deb 2005), similar to other evolutionary algorithms, consists of the following steps:

1. An initial population of individuals is generated using an initialization criterion.
2. Each individual is evaluated based on the fitness function to obtain a fitness value that represents its ability to solve the problem.
3. In each generation, the algorithm selects a subset of the population to be parents of offspring. The selection criterion usually picks the best individuals to be parents, to ensure the survival of the best genes.
4. This subset of individuals is crossed using different crossover operators, obtaining the offspring.
5. These individuals may be mutated, applying different mutation genetic operators.
6. These new individuals must be evaluated using the fitness function to obtain their fitness values.
7. Different strategies can be employed for the replacement of individuals within the population and the offspring to ensure that the population size in the next generation is constant and the best individuals are kept.
8. The algorithm performs a control stage that determines whether to finish the execution by finding acceptable solutions or by having reached a maximum number of generations, if not the algorithm goes back to step 3 and performs a new iteration (generation).

The pseudo-code of a simple generational algorithm is shown in Algorithm 1.

Algorithm 1 Generational Algorithm

```
Require: max_generations

1: Initialize (P)
2: Evaluate (P)
3: num_generations ← 0
4: while num_generations < max_generations do
5:     P ← Parent selection (Population)
6:     C ← Crossover (P)
7:     M ← Mutation (C)
8:     Evaluate (M)
9:     Population ← Replacement (M ∪ Population)
10:    num_generations++
11: end while
```

Evaluation: fitness function The evaluation stage is the evaluation of the fitness function over the individuals. When a rule or individual is used to classify a given training instance from the dataset, one of these four possible values can be obtained: true positive t_p false positive f_p true negative t_n and false negative f_n. The true positive and true negative are correct classifications, while the false positive and false negative are incorrect classifications.

- *True positive* The rule predicts the class and the class of the given instance is indeed that class.
- *False positive* The rule predicts a class but the class of the given instance is not that class.
- *True negative* The rule does not predict the class and the class of the given instance is indeed not that class.
- *False negative* The rule does not predict the class but the class of the given instance is in fact that class.

The results of the individual's evaluations over all the patterns from a dataset are used to build the confusion matrix which allows us to apply different quality indexes to get the individual's fitness value and its calculation is usually the one that requires more computing time. Therefore, our model will also perform this calculation so that each algorithm can apply the most convenient fitness function.

The main problem of the evaluation is the computational time required for the match process because it involves comparing all the rules with all the instances of the dataset. The number of evaluations is huge when the population size or the number of instances increases, thus the algorithm must perform up to millions of evaluations in each generation.

Evaluation: computational study Several previous experiments have been conducted to evaluate the computational time of the different stages of the generational algorithm. These experiments execute the different algorithms described in Sect. 6.1 over the problem domains proposed in Sect. 6.3. The population size was set to 50, 100 and 200 individuals, whereas the number of generations was set to 100 iterations. The results of the average execution time of the different stages of the algorithms among all the configurations are shown in Table 1.

Table 1 GP classification execution time

Phase	Percentage
Initialization	**8.96**
Creation	0.39
Evaluation	8.57
Generation	**91.04**
Selection	0.01
Crossover	0.01
Mutation	0.03
Evaluation	85.32
Replacement	0.03
Control	5.64
Total	**100.00**

The experience using these GP algorithms proves that on average around 94% of the time is taken by the evaluation stage. This percentage is mainly linked to the algorithm, the population size and the number of patterns, increasing up to 99% on large problems. Anyway, the evaluation phase is always more expensive regardless of the algorithm or its parameters. We can conclude that evaluation takes most of the execution time so the most significant improvement would be obtained by accelerating this phase. Therefore, we propose a parallel GPU model detailed in Sect. 5 to speed up the evaluation phase.

4 CUDA programming model

Computer unified device architecture (CUDA) (NVIDIA 2010) is a parallel computing architecture developed by NVIDIA that allows programmers to take advantage of the computing capacity of NVIDIA GPUs in a general purpose manner. The CUDA programming model executes kernels as batches of parallel threads in a SIMD programming style. These kernels comprise thousands to millions of lightweight GPU threads per each kernel invocation.

CUDA's threads are organized into a two-level hierarchy represented in Fig. 2: at the higher one, all the threads in a data-parallel execution phase form a grid. Each call to a kernel execution initiates a grid composed of many thread groupings, called thread blocks. All the blocks in a grid have the same number of threads, with a maximum of 512. The maximum number of thread blocks is 65,535 $\times$ 65,535, so each device can run up to 65,535 $\times$ 65,535 $\times$ 512 $= 2 \times 10^{12}$ threads per kernel call.

To properly identify threads within the grid, each thread in a thread block has a unique ID in the form of a three-dimensional coordinate, and each block in a grid also has a unique two-dimensional coordinate.

Thread blocks are executed in streaming multiprocessors. A stream multiprocessor can perform zero overhead scheduling to interleave warps and hide the overhead of long-latency arithmetic and memory operations.

There are four different main memory spaces: global, constant, shared and local. These GPU memories are specialized and have different access times, lifetimes and output limitations.

- *Global memory* is a large, long-latency memory that exists physically as an off-chip dynamic device memory. Threads can read and write global memory to share data and must write the kernel's output to be readable after the kernel terminates. However, a better way to share data and improve performance is to take advantage of shared memory.
- *Shared memory* is a small, low-latency memory that exists physically as on-chip registers and its contents are only maintained during thread block execution and are discarded when the thread block completes. Kernels that read or write a known range of global memory with spatial or temporal locality can employ shared memory as a software-managed cache. Such caching potentially reduces global memory bandwidth demands and improves overall performance.
- *Local memory* each thread also has its own local memory space as registers, so the number of registers a thread uses determines the number of concurrent threads executed in the multiprocessor, which is called multiprocessor occupancy. To avoid wasting hundreds of cycles while a thread waits for a long-latency global-memory load or store to complete, a common technique is to execute batches of global accesses, one per thread, exploiting the hardware's warp scheduling to overlap the threads' access latencies.
- *Constant memory* is specialized for situations in which many threads will read the same data simultaneously.

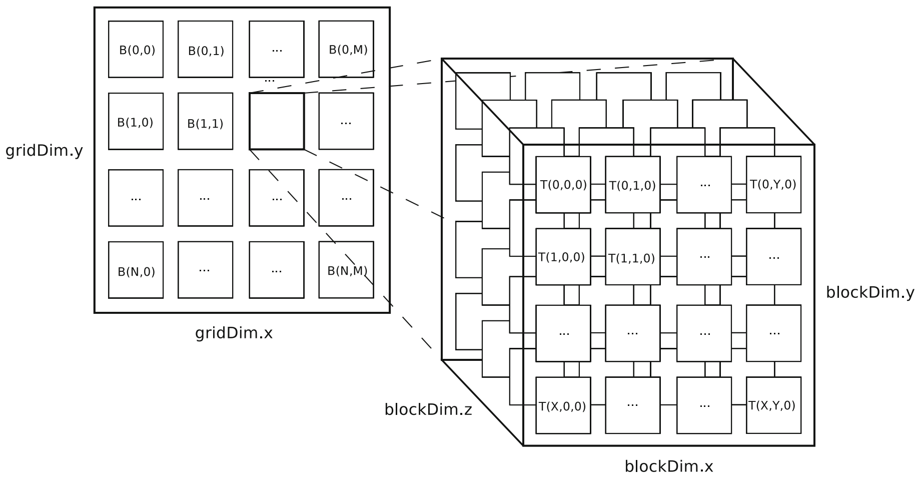


Fig. 2 CUDA threads and blocks model

This type of memory stores data written by the host thread, is accessed constantly and does not change during the execution of the kernel. A value read from the constant cache is broadcast to all threads in a warp, effectively serving 32 loads from memory with a single-cache access. This enables a fast, single-ported cache to feed multiple simultaneous memory accesses. The amount of constant memory is 64 KB.

For maximum performance, these memory accesses must be coalesced as with accesses to global memory. Global memory resides in device memory and is accessed via 32, 64, or 128-byte segment memory transactions. When a warp executes an instruction that accesses global memory, it coalesces the memory accesses of the threads within the warp into one or more of these memory transactions depending on the size of the word accessed by each thread and the distribution of the memory addresses across the threads. In general, the more transactions are necessary, the more unused words are transferred in addition to the words accessed by the threads, reducing the instruction throughput accordingly.

To maximize global memory throughput, it is therefore important to maximize coalescing by following the most optimal access patterns, using data types that meet the size and alignment requirement or padding data in some cases, for example, when accessing a two-dimensional array. For these accesses to be fully coalesced, both the width of the thread block and the width of the array must be multiple of the warp size.

5 Model description

This section details an efficient GPU-based evaluation model for fitness computation. Once it has been proved that the evaluation phase is the one that requires the most of the computational time, this section discusses the procedure of the fitness function to understand its cost in terms of runtime and memory occupancy. We then employ this knowledge to propose an efficient GPU-based evaluation model in order to maximize the performance based on optimization principles (Ryoo et al. 2008) and the recommendations of the NVIDIA CUDA programming model guide (NVIDIA 2010).

5.1 Evaluation complexity

The most computationally expensive phase is evaluation since it involves the match of all the individuals generated over all the patterns. Algorithm 2 shows the pseudo-code of the fitness function. For each individual its genotype must be interpreted or translated into an executable format and then it is evaluated over the training set. The evaluation process of the individuals is usually implemented in two loops, where each individual iterates each pattern and checks if the rule covers that pattern. Considering that the population size is P and the training set size is T the number of iterations is $O(P \times T)$. These two loops make the algorithm really slow when the population size or the pattern count increases because the total number of iterations is the product of these two parameters. This *one by one* iterative model is slow but it only requires 4 × *populationSize* × *size of(int)* bytes from memory, i.e., the four integer counters for t_p, t_n, f_p and f_n and values for each individual, this is $O(P)$ complex.

5.2 Efficient GPU-based evaluation

The execution of the fitness function over the individuals is completely independent from one individual to another. Hence, the parallelization of the individuals is feasible. A naive way to do this is to perform the evaluations in parallel using several CPU threads, one per individual. The main problem is that affordable PC systems today only run CPUs with 4 or 8 cores, thus larger populations will need to serialize its execution so the speedup would be limited up to the number of cores that there is. This is where GPGPU systems can exploit its massively parallel model.

Algorithm 2 Evaluation: fitness function

```
Require: population_size, number_instances

1: for each individual within the population do
2:    tp ← 0, fp ← 0, tn ← 0, fn ← 0
3:    for each instance from the dataset do
4:       if individual's rule covers actual instance then
5:          if the consequent matches predicted class then
6:             tp++
7:          else
8:             fp++
9:          end if
10:      else
11:         if the consequent matches predicted class then
12:            fn++
13:         else
14:            tn++
15:         end if
16:      end if
17:   end for
18:   fitnessValue ← computeFitness(tp,tn,fp,fn)
19: end for
```

Using the GPU, the fitness function can be executed over all individuals concurrently. Furthermore, the simple match of a rule over an instance is a self-dependent operation: there is no interference with any other evaluation. Hence, the matches of all the individuals over all the instances can be performed in parallel in the GPU. This means that one thread represents the single match of a pattern over an instance. The total number of GPU threads required would be equal to the number of iterations from the loop of the sequential version. Once each thread has obtained the result of its match in the GPU device, these results have to be copied back to the host memory and summed up to get the fitness values. This approach would be very slow because in every generation it will be necessary to copy a structure of size $O(P \times T)$, specifically $populationSize \times numberInstances \times sizeof(int)$ bytes from device memory to host memory, i.e, copying the match results obtained from the coverage of every individual over every pattern. This is completely inefficient because the copy transactions time would be larger than the speedup obtained. Therefore, to reduce the copy structure size it is necessary to calculate the final result for each individual inside the GPU and then only copy a structure size $O(P)$ containing the fitness values. Hence, our fitness calculation model involves two steps: matching process and reducing the results for fitness values computation. These two tasks correspond to two different kernels detailed in Sect. 5.2.2.

The source code of our model can be compiled into a shared library to provide the user the functions to perform evaluations in GPU devices for any evolutionary system. The schema of the model is shown in Fig. 3.

At first, the user must call a function to perform the dynamic memory allocation. This function allocates the memory space and loads the dataset instances to the GPU global memory. Moreover, it runs one host thread per GPU device because the thread context is mandatorily associated to only one GPU device. Each host thread runs over one GPU and performs the evaluation of a subset of the individuals from the population. The execution of the host threads stops once the instances are loaded to the GPU awaiting a trigger. The evaluate function call is the trigger that wakes the threads to perform the evaluations over their population's subset. Evaluating the present individuals require the copy of their phenotypes to a GPU memory space. The GPU constant memory is the best location for storing the individual phenotypes because it provides broadcast to all the device threads in a warp. The host threads execute the kernels as batches of parallel threads, first the match kernel obtains the results of the match process and then the reduction kernel calculates the fitness values from these results. The fitness values must be copied back to host memory and associated to the individuals. Once all the individuals from the thread have been evaluated, the host thread sends a signal to the main thread telling it its job has finished and the algorithm process can continue once all the host threads have sent the go-ahead. This stop and go model continues while more generations are performed. At the end, a free memory function must be called to deallocate the dynamic memory previously allocated.

5.2.1 *Data structures*

The scheme proposed attempts to make full use of global and constant memory. The purpose is to optimize memory usage to achieve maximum memory throughput. Global memory is employed to store the instances from the dataset, the results of the match process, and the fitness values.

The most common dataset structure is a 2D matrix where each row is an instance and each column is an attribute. Loading the dataset to the GPU is simple: allocate a 2D array of width *number Instances* $\times$ *number Attributes* and copy the instances to the GPU. This approach, represented in Fig. 4, is simple-minded and it works but for maximum performance, the memory accesses must be

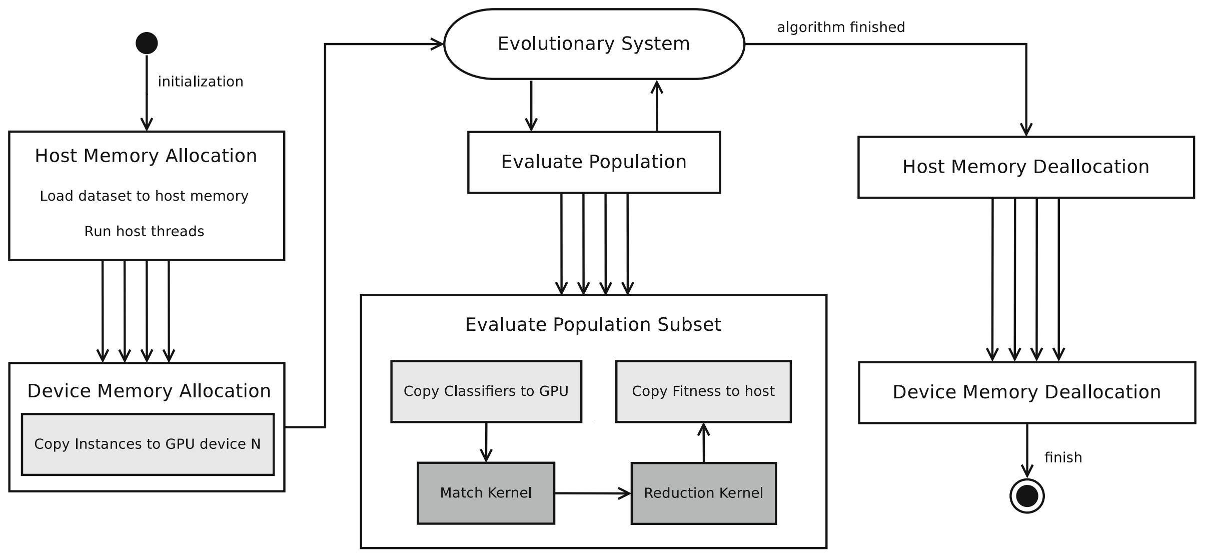


Fig. 3 Model schema

coalesced. When a warp executes an instruction that accesses global memory, it coalesces the memory accesses of the threads within the warp into one or more 32, 64, or 128-byte memory transactions depending on the size of the word accessed by each thread and the distribution of the memory addresses across the threads. In general, the more transactions are necessary, the more unused words are transferred in addition to the words accessed by the threads, reducing the instruction throughput accordingly.

The threads in the match kernel perform the match process of the classifiers over the instances. The *i* thread performs the match process over the *i* instance. Therefore, consecutive threads are executed over consecutive instances. The threads execute the phenotype of the individuals represented in reverse Polish notation using a stack and an interpreter. The phenotype follows the individual representation shown in Sect. 3.1. Thus, when attribute nodes have to obtain the attribute values for each instance covered by the threads within the warp, the attributes' addresses are spaced *numberAttributes* memory addresses (stride is *numberAttributes* × *sizeof* (*datatype*). Depending on the number of attributes, a memory transaction would transfer more or less useful values. Anyway, this memory access pattern shown in Fig. 5 is altogether inefficient because the memory transfer engine must split the memory request into many memory transactions that are issued independently.

The second approach shown in Fig. 6 for storing the instances is derived from the first one. The problem is the stride of the memory requests which is *numberAttributes*. The solution is to lower the stride to one transposing the 2D array that stores the instances. The length of the array remains constant but instead of storing all the attributes of an instance first, it stores the first attributes from all the instances. Now, the memory access pattern shown in Fig. 7 demands attributes which are stored in consecutive memory addresses. Therefore, a single 128-byte memory transaction would transfer the 32 integer or float attributes requested by the threads in the warp.

For these accesses to be fully coalesced, both the width of the thread block and the number of the instances must be a multiple of the warp size. The third approach for achieving fully coalesced accesses is shown in Figs. 8 and 9. Intra-array padding is necessary to align the addresses requested to the memory transfer segment sizes. Thus, the array must be expanded to *multiple*(*numberInstances*,32) × values.

The individuals to be evaluated must be uploaded in each generation to the GPU constant memory. The GPU has a read-only constant cache that is shared by all functional units and speeds up reads from the constant memory space, which resides in device memory. All the threads in a warp perform the match process of the same individual over different instances. Thus, memory requests point to

A(0,0)	A(1,0)	...	A(N,0)	A(0,1)	A(1,1)	...	A(N,1)	...	A(0,I)	...	A(N,I)

Fig. 4 Uncoalesced instances data array structure

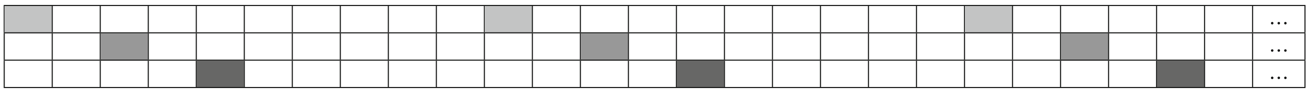


Fig. 5 Uncoalesced attributes request

A(0,0)	A(0,1)	...	A(0,I)	A(1,0)	A(1,1)	...	A(1,I)	...	A(N,0)	...	A(N,I)

Fig. 6 Coalesced instances data array structure

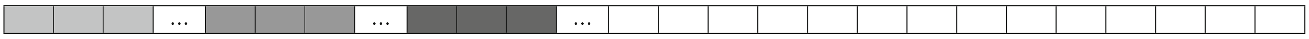


Fig. 7 Coalesced attributes request

A(0,0)	...	A(0,I)	X	A(1,0)	...	A(1,I)	X	...	A(N,0)	...	A(N,I)	X

Fig. 8 Fully coalesced intra-array padding instances data array structure

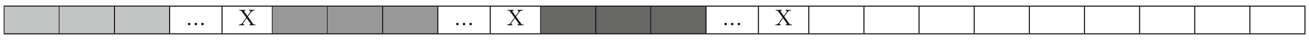


Fig. 9 Fully coalesced attributes request

R(0,0)	...	R(0,I)	X	R(1,0)	...	R(1,I)	X	...	R(N,0)	...	R(N,I)	X

Fig. 10 Results data array structure

the same node and memory address at a given time. Servicing one memory read request to several threads simultaneously is called broadcast. The resulting requests are serviced at the throughput of the constant cache in case of a cache hit, or at the throughput of device memory otherwise.

The results of the match process for each individual and instance must be stored in global memory for counting. Again, the memory accesses must be coalesced to device global memory. The best data structure is a 2D array *numberInstances* × *populationSize* shown in Fig. 10. Hence, the results write operations and the subsequent read operations for counting are both fully coalesced.

The fitness values calculated by the reduction kernel must be stored in global memory, then copied back to host memory and set to the individuals. A simple structure to store the fitness values of the individuals is a 1D array of length *populationSize*.

5.2.2 Evaluation process on GPU

The evaluation process on the GPU is performed using two kernel functions. The first kernel performs the match operations between the individuals and the instances storing a certain result. Each thread is in charge of a single match. The second kernel counts the results of an individual by a reduction operation. This 2-kernel model allows the user to perform the match processes and the fitness values' calculations completely independently. Once the results of the match process are obtained, any fitness function can be employed to calculate the fitness values. This requires copying back to global memory a large amount of data at the end of the first kernel. Franco et al. (2010) proposes to minimise the volume of data by performing a reduction in the first kernel. However, the experiments carried out indicate to us that the impact on the run-time of reducing data in the first kernel is larger than that of storing the whole data array because our approach allows the kernels to avoid synchronization between threads and unnecessary delays. Furthermore, the threads block dimensions can be ideally configured for each kernel independently.

Match kernel The first kernel performs in parallel the match operations between the classifiers and the instances. Algorithm 3 shows the pseudo-code for this kernel.

The number of matches and hence the total number of threads is *populationSize* × *numberInstances*. The maximum amount of threads per block is 512 or 1,024 depending on the device's computing capability. However, optimal values are multiples of the warp size. A GPU multiprocessor relies on thread-level parallelism to maximize utilization of its functional units. Utilization is therefore directly linked to the number of resident warps. At every instruction issue time, a warp scheduler selects a warp that is ready to execute, if any, and issues the next instruction to the active threads of the warp. The number of clock cycles it takes for a warp to be ready to execute its next instruction is called latency, and full utilization is achieved when the warp scheduler always has some instruction to issue for some warp at every clock cycle during that latency period, i.e., when the latency of each warp is completely hidden by other warps.

Algorithm 3 Match kernel

```
1: instance ← blockDim.y * blockIdx.y + threadIdx.y;
2: if instance < numberInstances then
3:     resultMemPosition ← blockIdx.x * numberInstancesAligned + instance;
4:     if covers(classifier[blockIdx.x],instance) then
5:         if classifiedClass == instancesClass[instance] then
6:             result[resultMemPosition] ← tp
7:         else
8:             result[resultMemPosition] ← fp
9:         end if
10:    else
11:        if classifiedClass != instancesClass[instance] then
12:            result[resultMemPosition] ← tn
13:        else
14:            result[resultMemPosition] ← fn
15:        end if
16:    end if
17: end if
```

The CUDA occupancy calculator spreadsheet allows computing the multiprocessor occupancy of a GPU by a given CUDA kernel. The multiprocessor occupancy is the ratio of active warps to the maximum number of warps supported on a multiprocessor of the GPU. The optimal number of threads per block obtained from the experiments carried out for this kernel is 128 for devices of compute capability 1×, distributed in 4 warps of 32 threads. The active thread blocks per multiprocessor is 8. Thus, the active warps per multiprocessor is 32. This means a 100% occupancy of each multiprocessor for devices of compute capability 1×. Recent devices of compute capability 2× requires 192 threads per block to achieve 48 active warps per multiprocesor and a 100% occupancy. The number of

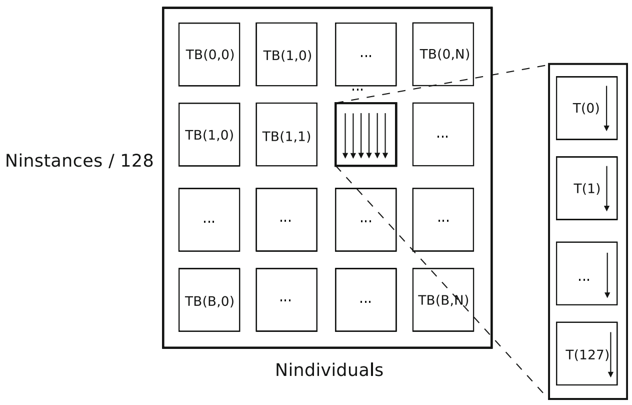


Fig. 11 Match kernel 2D grid of thread blocks

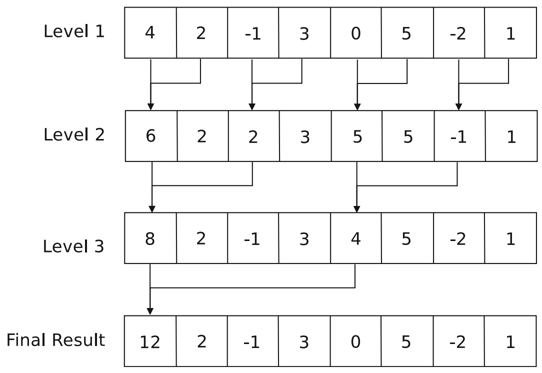


Fig. 12 Parallel reduction algorithm

threads per block does not matter, since the model is adapted to achieve maximum performance in any case.

The kernel is executed using a 2D grid of thread blocks as shown in Fig. 11. The first dimension length is *populationSize*. Using *N* threads per block, the number of thread blocks to cover all the instances is *ceil*(*number Instances*/*N*) in the second dimension of the grid. Thus, the total number of thread blocks is *populationSize* × *ceil*(*numberInstances*/*N*). This number is important as it concerns the scalability of the model in future devices. NVIDIA recommends that one run at least twice as many thread blocks as the number of multiprocessors.

Reduction kernel The second kernel reduces the results previously calculated in the first kernel and obtains the fitness value for each individual. The naive reduction operation shown in Fig. 12 sums in parallel the values of an array reducing iteratively the information. Our approach does not need to sum the values, but counting the number of t_p,f_p, t_n and f_n resulted for each individual from the match kernel. These four values are employed to build the confusion matrix. The confusion matrix allows us to apply different quality indexes defined by the authors to get the individual's fitness value.

Designing an efficient reduction kernel is not simple because it is the parallelization of a natively sequential task. In fact, NVIDIA propose six different approaches (NVIDIA 2010). Some of the proposals take advantage of the device shared memory. Shared memory provides a small but fast memory shared by all the threads in a block. It is quite desirable when the threads require synchronization and work together to accomplish a task like reduction.

A first approach to count the number of t_p, f_p, t_n and f_n in the results array using *N* threads is immediate. Each thread counts for the *N*th part of the array, specifically *numberInstances/numberThreads* items, and then the values for each thread are summed. This 2-level reduction is not optimal because the best would be the *N*/2-level reduction, but reducing each level requires the synchronization of the threads. Barrier synchronization can impact performance by forcing the multiprocessor to idle. Therefore, a 2 or 3-level reduction has been proved to perform the best.

To achieve a 100% occupancy, the reduction kernel must employ 128 or 192 threads, for devices of compute capability 1× or 2×, respectively. However, it is not trivial to organize the threads to count the items. The first approach involves the thread *i* counting *numberInstances/ numberThreads* items from the *thread Idx* × *numberInstances/numberThreads* item. The threads in a thread-warp would request the items spaced *numberInstances* memory addresses. Therefore, once again one has a coalescing and undesirable problem. Solving the memory requests pattern is naive. The threads would count again *numberInstances/ number Threads* items but for coalescing purposes the memory access pattern would be *iteration* × *number-Threads* + *thread Idx*. This way, the threads in a warp request consecutive memory addresses that can be serviced in fewer memory transactions. This second approach is shown in Fig. 13. The reduction kernel is executed using a

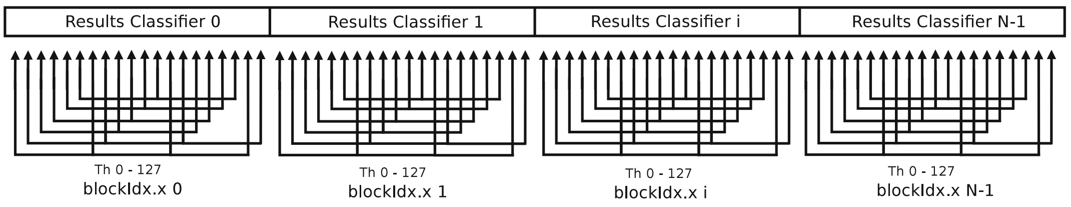


Fig. 13 Coalesced reduction kernel

1D grid of thread blocks whose length is *populationSize*. Using 128 or 192 threads per block, each thread block performs the reduction of the results for an individual. A shared memory array of length 4 × *numberThreads* keeps the temporary counts for all the threads. Once all the items have been counted, a synchronization barrier is called and the threads wait until all the threads in the thread block have reached this point and all global and shared memory accesses made by these threads prior to the synchronization point are visible to all threads in the block. Finally, only one thread per block performs the last sum, calculates the fitness value and writes it to global memory.

6 Experimental study

This section describes the details of the experiments, discusses the application domains, the algorithms used, and the settings of the tests.

6.1 Parallelized methods with our proposal

To show the flexibility and applicability of our model, three different GP classification algorithms proposed in the literature are tested using our proposal in the same way as could be applied to other algorithms or paradigms. Next, the major specifications of each of the proposals that have been considered in the study are detailed.

(1) De Falco et al. (2001) propose a method to get the fitness of the classifier by evaluating the antecedent over all the patterns within the dataset. Falco et al. uses the logical operators AND, OR, NOT, the relational operators $=, <, >, \leq, \geq$ and two numerical interval comparator operators IN and OUT. The evolution process is repeated as many times as classes holds the dataset. In each iteration the algorithm focuses on a class and keeps the best rule obtained to build the classifier.

The crossover operator selects two parent nodes and swaps the two subtrees. The crossover is constrained by two restrictions: the nodes must be compatible and the depth of the tree generated must not exceed a preset depth.

The mutation operator can be applied either to a terminal node or a non-terminal node. This operator selects a node randomly, if it is a terminal node, it is replaced by another randomly selected compatible terminal node, otherwise, the non-terminal node is replaced by another randomly selected compatible terminal node with the same arity and compatibility.

The fitness function calculates the difference between the number of instances where the rule correctly predicts the membership or not of the class and number of examples where the opposite occurs and the prediction is wrong. Finally, the fitness function is defined as:

$$fitness = nI - ((t_p + t_n) - (f_p + f_n)) + \alpha * N$$

where nI is the number of instances, α is a value between 0 and 1 and N is the number of nodes in the rule. The closer is α to 1, the more importance is given to simplicity.

(2) Tan et al. (2002) proposes a modified version of the steady-state algorithm (Banzhaf et al. 1998) which uses an external population and elitism to ensure that some of the best individuals of the current generation survive in the next generation.

The fitness function combines two indicators that are commonplace in the domain, namely the sensitivity (*Se*) and the specifity (*Sp*) defined as follows:

$$Se = \frac{t_p}{t_p + w_1 f_n} \quad Sp = \frac{t_n}{t_n + w_2 f_p}$$

The parameters w_1 and w_2 are used to weight the influence of the false negatives and false positives cases in the fitness calculation. This is very important because these values are critical in problems such as diagnosis. Decreasing $w1$ or increasing w_2 generally improves the results but also increases the number of rules. The range [0.2–1] for w_1 y [1–20] for w_2 is usually reasonable for the most cases. Therefore, the fitness function is defined by the product of these two parameters: $fitness = Se \times Sp$. The proposal of Tan et al. is similar to that of Falco et al. but the for OR operator because combinations of AND and NOT operators which can generate all the necessary rules. Therefore, the simplicity of the rules is affected. Tan et al. also introduces the token competition technique proposed by Wong and Leung (2000) and it is employed as an alternative niche approach to promote diversity. Most of the time, only a few rules are useful and cover most of the instances while most others are redundant. The token competition is an effective way to eliminate redundant rules.

(3) Bojarczuk et al. (2004) present a method in which each rule is evaluated for all of the classes simultaneously for a pattern. The classifier is formed by taking the best individual for each class generated during the evolutionary process. The Bojarczuk et al. algorithm does not have a mutation operator.

This proposal uses the logical operators AND, OR and NOT, although AND and NOT would be sufficient; this way the size of the generated rules is reduced. GP does not produce simple solutions. The comprehensibility of a rule is inversely proportional to its size. Therefore Bojarczuk et al. define the simplicity *Sy* of a rule:

$$Sy = \frac{maxnodes - 0.5 \times numnodes - 0.5}{maxnodes - 1}$$

where *maxnodes* is the maximum depth of the syntaxtree, *numnodes* is the number of nodes of the current rule, and *Se* and *Sp* are the sensitivity and the specifity parameters described in the Tan et al. with $w1$ and $w2$ equal to 1. The fitness value is the product of these three parameters.

$fitness = Se \times Sp \times Sy$

These three methods implement the match kernel and the reduction kernel. The match kernel obtains the results from the match processes of the prediction of the examples with their actual class. The reduction kernel counts the t_p, t_n, f_p and f_n values and computes the fitness values.

6.2 Comparison with other proposal

One of the most recent works and similar to our proposal is the one by Franco et al. (2010). This work speeds up the evaluation of the BioHEL system using GPGPUs. BioHEL is an evolutionary learning system designed to cope with large-scale datasets. They provide the results, the profiler information, the CUDA and the serial version of the software in the website http://www.cs.nott.ac.uk/~mxf/biohel. The experiments carried out compare our model and its speedup to the speedup obtained from the CUDA version of BioHEL system over several problem domains in order to demonstrate the improvements provided by the parallelization model proposed. The configuration settings of the BioHEL system were the provided by the authors in the configuration files.

6.3 Problem domains used in the experiments

To evaluate the performance of the proposed GP evaluation model, some datasets selected from the UCI machine learning repository (Newman and Asuncion 2007) and the KEEL website (Alcalá-Fdez et al. 2009) are benchmarked using the algorithms previously described. These datasets are very varied considering different degrees of complexity. Thus, the number of instances ranges from the simplest containing 150 instances to the most complex containing one million instances. Also, the number of attributes and classes are different in different datasets. This information is summarized in Table 2. The wide variety of datasets considered allows us to evaluate the model performance in both low and high problem complexity. It is interesting to note that some of these datasets such as KDDcup or Poker have not been commonly addressed to date because they are not memory and CPU manageable by traditional models.

Table 2 Complexity of the datasets tested

Dataset	#Instances	#Attributes	#Classes
Iris	150	4	3
New-thyroid	215	5	3
Ecoli	336	7	8
Contraceptive	1,473	9	3
Thyroid	7,200	21	3
Penbased	10,992	16	10
Shuttle	58,000	9	7
Connect-4	67,557	42	3
KDDcup	494,020	41	23
Poker	1,025,010	10	10

6.4 General experimental settings

The GPU evaluation code is compiled into a shared library and loaded into the JCLEC (Ventura et al. 2007) framework using JNI. JCLEC is a software system for evolutionary computation research developed in the Java programming language. Using the library, our model can be easily employed in any evolutionary learning system.

Experiments were run on two PCs both equipped with an Intel Core i7 quad-core processor running at 2.66GHz and 12 GB of DDR3 host memory. One PC features two NVIDIA GeForce 285 GTX video cards equipped with 2 GB of GDDR3 video RAM and the other one features two NVIDIA GeForce 480 GTX video cards equipped with 1.5 GB of GDDR5 video RAM. No overclock was made to any of the hardware. The operating system was GNU/Linux Ubuntu 10.4 64 bit.

The purpose of the experiments is to analyze the effect of the dataset complexity on the performance of the GPU evaluation model and the scalability of the proposal. Each algorithm is executed over all the datasets using a sequential approach, a threaded CPU approach, and a massively parallel GPU approach.

7 Results

This section discusses the experimental results. The first section compares the performance of our proposal over different algorithms. The second section provides the results of the BioHEL system and compares them with the obtained by our proposal.

7.1 Results obtained using our proposal

In this section we discuss the performance achieved by our proposal using three different GP algorithms. The execution time and the speedups of the three classification algorithms solving the various problems considered are shown in Tables 3, 4 and 5 where each column is labeled with the execution configuration indicated from left to right as follows: the dataset, the execution time of the native sequential version coded in Java expressed in seconds, the speedup of the model proposed using JNI and one CPU thread, two CPU threads, four CPU threads, one GTX 285 GPU, two GTX 285 GPUs, and with one and two GTX 480 GPUs. The results correspond to the executions of the algorithms with a population of 200 individuals and 100 generations.

Table 3 Falco et al. algorithm execution time and speedups

Execution time (s)		Speedup						
Dataset	Java	1 CPU	2 CPU	4 CPU	1 285	2 285	1 480	2 480
Iris	2.0	0.48	0.94	1.49	2.96	4.68	2.91	8.02
New-thyroid	4.0	0.54	1.03	1.99	4.61	9.46	5.18	16.06
Ecoli	13.7	0.49	0.94	1.38	6.36	10.92	9.05	17.56
Contraceptive	26.6	1.29	2.52	3.47	31.43	55.29	50.18	93.64
Thyroid	103.0	0.60	1.15	2.31	37.88	69.66	75.70	155.86
Penbased	1,434.1	1.15	2.26	4.37	111.85	207.99	191.67	391.61
Shuttle	1,889.5	1.03	2.02	3.87	86.01	162.62	182.19	356.17
Connect-4	1,778.5	1.09	2.14	3.87	116.46	223.82	201.57	392.86
KDDcup	154,183.0	0.91	1.77	3.30	136.82	251.71	335.78	653.60
Poker	108,831.6	1.25	2.46	4.69	209.61	401.77	416.30	820.18

Table 4 Bojarczuk et al. algorithm execution time and speedups

Execution time (s)		Speedup						
Dataset	Java	1 CPU	2 CPU	4 CPU	1 285	2 285	1 480	2 480
Iris	0.5	0.50	0.96	1.75	2.03	4.21	2.39	5.84
New-thyroid	0.8	0.51	1.02	1.43	2.99	5.85	3.19	9.45
Ecoli	1.3	0.52	1.02	1.15	3.80	8.05	5.59	11.39
Contraceptive	5.4	1.20	2.40	2.47	14.58	31.81	26.41	53.86
hyroid	27.0	0.56	1.11	2.19	25.93	49.53	56.23	120.50
Penbased	42.6	0.96	1.92	3.81	18.55	36.51	68.01	147.55
Shuttle	222.5	1.18	2.35	4.66	34.08	67.84	117.85	253.13
Connect-4	298.9	0.69	1.35	2.65	42.60	84.92	106.14	214.73
KDDcup	3,325.8	0.79	1.55	2.98	30.89	61.80	135.28	306.22
Poker	6,527.2	1.18	2.32	4.45	39.02	77.87	185.81	399.85

Table 5 Tan et al. algorithm execution time and speedups

Execution time (s)		Speedup						
Dataset	Java	1 CPU	2 CPU	4 CPU	1 285	2 285	1 480	2 480
Iris	2.6	0.44	0.80	1.01	2.94	5.44	4.90	9.73
New-thyroid	6.0	0.77	1.43	1.78	7.13	12.03	9.15	21.74
Ecoli	22.5	0.60	1.16	2.09	9.33	16.26	14.18	25.92
Contraceptive	39.9	1.28	2.44	3.89	40.00	64.52	60.60	126.99
Thyroid	208.5	1.10	2.11	2.66	64.06	103.77	147.74	279.44
Penbased	917.1	1.15	2.23	4.25	86.58	148.24	177.78	343.24
Shuttle	3,558.0	1.09	2.09	3.92	95.18	161.84	222.96	431.80
Connect-4	1,920.6	1.35	2.62	4.91	123.56	213.59	249.81	478.83
KDDcup	185,826.6	0.87	1.69	3.20	83.82	138.53	253.83	493.14
Poker	119,070.4	1.27	2.46	4.66	158.76	268.69	374.66	701.41

The results in the tables provide useful information that in some cases, the external CPU evaluation is inefficient for certain datasets such as Iris, New-thyroid or Ecoli. This is because the time taken to transfer the data from the Java virtual machine memory to the native memory is higher than just doing the evaluation in the Java virtual machine. However, in all the cases, regardless of the size of the dataset, the native GPU evaluation is always considerably faster. If we

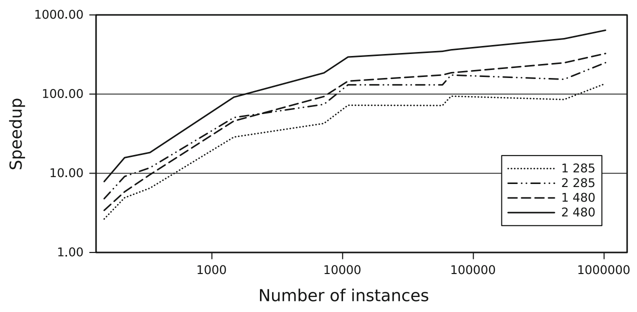


Fig. 14 Average speedups

look at the results of the smallest datasets such as Iris, New-thyroid and Ecoli, it can be seen that its speedup is acceptable and specifically Ecoli performs up to 25× faster. Speeding up these small datasets would not be too useful because of the short run time required, but it is worthwhile for larger data sets. On the other hand, if we focus on complex datasets, the speedup is greater because the model can take full advantage of the GPU multiprocessors' offering them many instances to parallelize. Notice that KDDcup and Poker datasets perform up to 653× and 820× faster, respectively. We can also appreciate that the scalability of the proposal is almost perfect, since doubling the number of threads or the graphics devices almost halves the execution time. Figure 14 summarizes the average speedup, depending on the number of instances.

The fact of obtaining significant enhancements in all problem domains (both small and complex datasets) as has been seen is because our proposal is a hybrid model that takes advantage of both the parallelization of the individuals and the instances. A great speedup is not only achieved by classifying a large number of instances but by a large enough population. The classification of small datasets does not require many individuals but high-dimensional problems usually require a large population to provide diversity in the population genetics. Therefore, a great speedup is achieved by maximizing both parameters. These results allow us to determine that the proposed model achieves a high speedup in the algorithms employed. Specifically, the best speedup is 820× when using the Falco et al. algorithm and the poker dataset; hence, the execution time can be impressively reduced from 30 h to only 2 min.

7.2 Results of the other proposal

This section discusses the results obtained by BioHEL. The results for the BioHEL system are shown in Table 6 where the first column indicates the execution time of the serial version expressed in seconds, the second column shows the speedup of the CUDA version using a NVIDIA GTX 285 GPU, and the third using a NVIDIA GTX 480 GPU. These results show that for a dataset with a low number of instances, the CUDA version of BioHEL performs slower

Table 6 BioHEL execution time and speedups

Execution time (s)		Speedup	
Dataset	Serial	1 285	1 480
Iris	0.5	0.64	0.66
New-thyroid	0.9	0.93	1.32
Ecoli	3.7	1.14	6.82
Contraceptive	3.3	3.48	3.94
Thyroid	26.4	2.76	8.70
Penbased	147.9	5.22	20.26
Shuttle	418.4	11.54	27.84
Connect-4	340.4	10.18	12.24
KDDcup	503.4	14.95	28.97
Poker	3,290.9	11.93	34.02

than the serial version. However, the speedup obtained is higher when the number of instances increases, achieving a speedup of up to 34 times compared to the serial version.

The speedup results for the BioHEL system shown in Table 6 compared with the results obtained by our proposal shown in Tables 3, 4 and 5 demonstrate the better performance of our model. One of the best advantages of our proposal is that it scales to multiple GPU devices, whereas BioHEL does not. Both BioHEL and our proposal employ a 2-kernel model. However, we do not to perform a one-level parallel reduction in the match kernel, in order to avoid synchronization between threads and unnecessary delays even if it means storing the whole data array. Thus, the memory requirements are larger but the reduction performs faster as the memory accesses are fully coalesced and synchronized. Moreover, our proposal improves the instruction throughput upto 1.45, i.e., the number of instructions that can be executed in a unit of time. Therefore, our proposal achieves 1 Teraflops performance using two GPUs NVIDIA GTX 480 with 480 cores running at 700 MHz. This information is provided in the CUDA profiler available in the respective websites.

Additional information of the paper such as the details of the kernels, the datasets employed, the experimental results and the CUDA profiler information are published in the website: http://www.uco.es/grupos/kdis/kdiswiki/SOCOGPU.

8 Conclusions and future work

The classification of large datasets using EAs is a time consuming computation as the problem complexity increases. To solve this problem, many studies have aimed at optimizing the computational time of EAs. In recent years, these studies have focused on the use of GPU devices whose main advantage over previous proposals are their massively parallel MIMD execution model that allows researchers to perform parallel computing where million threads can run concurrently using affordable hardware.

In this paper there has been proposed a GPU evaluation model to speed up the evaluation phase of GP classification algorithms. The parallel execution model proposed along with the computational requirements of the evaluation of individuals, creates an ideal execution environment where GPUs are powerful. Experimental results show that our GPU-based proposal greatly reduces the execution time using a massively parallel model that takes advantage of fine-grained and coarse-grained parallelization to achieve a good scalability as the number of GPUs increases. Specifically, its performance is better in high-dimensional problems and databases with a large number of patterns where our proposal has achieved a speedup of up to 820× compared to the non-parallel version.

The results obtained are very promising. However, more work can be done in this area. Specifically, the development of hybrid models is interesting from the perspective of evolving in parallel a population of individuals. The classical approach of genetic algorithms is not completely parallelizable because of the serialization of the execution path of certain pieces of code. There have been several proposals to overcome these limitations achieving excellent results. The different models used to perform distributed computing and parallelization approaches focus on two approaches (Dorigo and Maniezzo 1993): the islands model, where several isolated subpopulations evolve in parallel and periodically swap their best individuals from neighboring islands, and the neighborhood model that evolves a single population and each individual is placed in a cell of a matrix.

These two models are available for use in MIMD parallel architectures in the case of islands and SIMD models for the neighborhood. Therefore, both perspectives can be combined to develop multiple models of parallel and distributed algorithms (Harding and Banzhaf 2009), which take advantage of the parallel threads in the GPU, the use of multiple GPUs, and the distribution of computation across multiple machines networked with these GPUs.

Acknowledgments This work has been financed in part by the TIN2008-06681-C06-03 project of the Spanish Inter-Ministerial Commission of Science and Technology (CICYT), the P08-TIC-3720 project of the Andalusian Science and Technology Department, and FEDER funds.

References

Alcalá-Fdez J, Sánchez LS, García S, del Jesus M, Ventura S, Garrell J, Otero J, Romero C, Bacardit J, Rivas V, Fernández J, Herrera F (2009) KEEL: KEEL: a software tool to assess evolutionary algorithms for data mining problems. Soft Comput Fusion Found Methodol Appl 13:307–318

Bäck TB, Fogel D, Michalewicz Z (1997) Handbook of evolutionary computation. Oxford University Press

Banzhaf W, Nordin P, Keller RE, Francone FD (1998) Genetic programming—an introduction: on the automatic evolution of computer programs and its applications. Morgan Kaufmann, San Francisco, CA

Bojarczuk CC, Lopes HS, Freitas AA, Michalkiewicz EL (2004) A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets. Artif Intell Med 30(1):27–48

Chitty DM, Malvern Q (2007) A data parallel approach to genetic programming using programmable graphics hardware. In: GECCO G07: Proceedings of the 9th annual conference on genetic and evolutionary computation. ACM Press, pp 1566–1573

De Falco I, Della Cioppa A, Tarantino E (2001) Discovering interesting classification rules with genetic programming. Appl Soft Comput 1(4):257–269

De Falco I, Della Cioppa A, Fontanella F, Tarantino E (2004) An innovative approach to genetic programming-based clustering. In: 9th online world conference on soft computing in industrial applications

Deb K (2005) A population-based algorithm-generator for real-parameter optimization. Soft Comput 9:236–253

Dorigo M, Maniezzo V (1993) Parallel genetic algorithms: introduction and overview of current research. In: Parallel genetic algorithms: theory and applications. IOS Press, Amsterdam, The Netherlands, pp 5–42

Espejo PG, Ventura S, Herrera F (2010) A survey on the application of genetic programming to classification. IEEE Trans Syst Man Cybern Part C (Appl Rev) 40(2):121–144

Franco MA, Krasnogor N, Bacardit J (2010) Speeding up the evaluation of evolutionary learning systems using GPGPUs. In: Proceedings of the 12th annual conference on genetic and evolutionary computation, GECCO '10. ACM, New York, NY, pp 1039–1046

Freitas AA (2002) Data mining and knowledge discovery with evolutionary algorithms. Springer, New York

General-purpose computation on graphics hardware. http://www.gpgpu.org. Accessed November 2010

Harding S (2010) Genetic programming on graphics processing units bibliography. http://www.gpgpgpu.com. Accessed November 2010

Harding S, Banzhaf W (2007) Fast genetic programming and artificial developmental systems on gpus. In: High performance computing systems and applications, 2007. HPCS, pp 2-2

Harding S, Banzhaf W (2009) Distributed genetic programming on GPUs using CUDA. In: Workshop on parallel architectures and bioinspired algorithms. Raleigh, USA

Hwu WW (2009) Illinois ECE 498AL: programming massively parallel processors, lecture 13: reductions and their implementation. http://nanohub.org/resources/7376

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection (complex adaptive systems). The MIT Press, Cambridge

Langdon WB, Harrison AP (2008) GP on SPMD parallel graphics hardware for mega bioinformatics data mining. Soft Comput 12(12):1169–1183

Landry J, Kosta LD, Bernier T (2006) Discriminant feature selection by genetic programming: towards a domain independent multi-class object detection system. J Syst Cybern Inform 3(1)

Maitre O, Baumes LA, Lachiche N, Corma A, Collet P (2009) Coarse grain parallelization of evolutionary algorithms on gpgpu cards with easea. In: Proceedings of the 11th annual conference on genetic and evolutionary computation, GECCO '09. ACM, New York, NY, USA, pp 1403–1410

Newman DJ, Asuncion A (2007) UCI machine learning repository. University of California, Irvine

NVIDIA (2010) NVIDIA programming and best practices guide. http://www.nvidia.com/cuda. Accessed November 2010

Robilliard D, Marion-Poty V, Fonlupt C (2009) Genetic programming on graphics processing units. Genetic Program Evolvable Mach 10:447–471

Ryoo S, Rodrigues CI, Baghsorkhi SS, Stone SS, Kirk DB, Hwu WW (2008) Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In: Proceedings of the 13th ACM SIGPLAN symposium on principles and practice of parallel programming, PPoPP '08. ACM, New York, NY, pp 73–82

Schmitz T, Hohmann S, Meier K, Schemmel J, Schürmann F (2003) Speeding up hardware evolution: a coprocessor for evolutionary algorithms. In: Proceedings of the 5th international conference on evolvable systems: from biology to hardware, ICES'03. Springer, pp 274–285

Tan KC, Tay A, Lee TH, Heng CM (2002) Mining multiple comprehensible classification rules using genetic programming. In: Proceedings of the evolutionary computation on 2002. CEC '02. Proceedings of the 2002 Congress, volume 2 of CEC '02. IEEE Computer Society, Washington, DC, pp 1302–1307

Ventura S, Romero C, Zafra A, Delgado JA, Hervás C (2007) JCLEC: a Java framework for evolutionary computation. Soft Comput 12:381–392

Wong ML, Leung KS (2000) Data mining using grammar-based genetic programming and applications. Kluwer Academic Publishers, Norwell, MA

Information Sciences 240 (2013) 1–20

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

An interpretable classification rule mining algorithm

Alberto Cano, Amelia Zafra, Sebastián Ventura *

Department of Computer Science and Numerical Analysis, University of Córdoba, Spain

A R T I C L E I N F O

Article history:
Received 9 October 2011
Received in revised form 14 February 2013
Accepted 16 March 2013
Available online 6 April 2013

Keywords:
Classification
Evolutionary programming
Interpretability
Rule mining

A B S T R A C T

Obtaining comprehensible classifiers may be as important as achieving high accuracy in many real-life applications such as knowledge discovery tools and decision support systems. This paper introduces an efficient Evolutionary Programming algorithm for solving classification problems by means of very interpretable and comprehensible IF-THEN classification rules. This algorithm, called the Interpretable Classification Rule Mining (ICRM) algorithm, is designed to maximize the comprehensibility of the classifier by minimizing the number of rules and the number of conditions. The evolutionary process is conducted to construct classification rules using only relevant attributes, avoiding noisy and redundant data information. The algorithm is evaluated and compared to nine other well-known classification techniques in 35 varied application domains. Experimental results are validated using several non-parametric statistical tests applied on multiple classification and interpretability metrics. The experiments show that the proposal obtains good results, improving significantly the interpretability measures over the rest of the algorithms, while achieving competitive accuracy. This is a significant advantage over other algorithms as it allows to obtain an accurate and very comprehensible classifier quickly.

1. Introduction

Discovering knowledge in large amounts of data collected over the last decades has become significantly challenging and difficult, especially in large-scale databases. Data mining (DM) [60] involves the use of data analysis tools to discover previously unknown, valid patterns and relationships in large data sets. Classification and regression are two forms of data analysis which can be used to extract models describing important data classes or to predict future data trends. Classification predicts categorical labels whereas regression models predict continuous-valued functions.

The data analysis tools used for DM include statistical models, mathematical methods, and machine learning algorithms. Classification is a common task in supervised machine learning with the search for algorithms that learn from training examples to produce predictions about future examples.

Classification has been successfully solved using several approaches [26]. On the one hand, there are approaches such as artificial neural networks (ANN) [46], support vector machines (SVM) [16], and instance-based learning methods [2]. These approaches obtain accurate classification models but they must be regarded as black boxes, i.e., they are opaque to the user. Opaque predictive models prevent the user from tracing the logic behind a prediction and obtaining interesting knowledge previously unknown from the model. These classifiers do not permit human understanding and inspection, they are not directly interpretable by an expert and it is not possible to discover which are the relevant attributes to predict the class of an example. This opacity prevents them from being used in many real-life knowledge discovery applications where both accuracy and comprehensibility are required, such as medical diagnosis [55], credit risk evaluation [42], and decision support systems [6], since the prediction model must explain the reasons for classification.

* Corresponding author.
E-mail addresses: acano@uco.es (A. Cano), azafra@uco.es (A. Zafra), sventura@uco.es (S. Ventura).

0020-0255/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2013.03.038

On the other hand, there are machine learning approaches which overcome this limitation and provide transparent and comprehensible classifiers such as decision trees [62] and rule-based systems [49]. Evolutionary Algorithms [65], and specifically Evolutionary Programming (EP) [13,64] and Genetic Programming (GP) [25], have been successfully applied to build decision trees and rule-based systems easily. Rule-based systems are especially user-friendly and offer compact, understandable, intuitive and accurate classification models. To obtain comprehensibility, accuracy is often sacrificed by using simpler but transparent models, achieving a trade-off between accuracy and comprehensibility. Even though there are many rule based classification models, it has not been until recently that the comprehensibility of the models is becoming a more relevant objective. Proof of this trend is found in recent studies of issue [18,27,34,57], i.e, the comprehensibility of the models is a new challenge as important as accuracy. This paper focuses on the interpretability, trying to reach more comprehensible models than most of the current proposals and thus covering the needs of many application domains that require greater comprehensibility than the provided by current methods.

This paper presents an EP approach applied to classification problems to obtain comprehensible rule-based classifiers. This algorithm, called ICRM (Interpretable Classification Rule Mining), is designed to obtain a base of rules with the minimum number of rules and conditions, in order to maximize its interpretability, while obtaining competitive accuracy results. The algorithm uses an individual = rule representation, following the Iterative Rule Learning (IRL) model. Individuals are constructed by means of a context-free grammar [33,61], which establishes a formal definition of the syntactical restrictions of the problem to be solved and its possible solutions, so that only grammatically correct individuals are generated. Next, the most important characteristics of the algorithm are detailed. Firstly, the algorithm guarantees obtaining the minimum number of rules. This is possible because it generates one rule per class, together with a default class prediction, which is assigned when none of the available rules are triggered. Moreover, it is guaranteed that there are no contradictory or redundant rules, i.e., there is no pair of rules with the same antecedents and different consequents. Finally, it also guarantees the minimum number of conditions forming the antecedents of these rules, which is achieved by selecting only the most relevant and discriminating attributes that separate the classes in the attribute domains.

The experiments carried out on 35 different data sets and nine other algorithms show the competitive performance of our proposal in terms of predictive accuracy and execution time, obtaining significantly better results than all the other algorithms in terms of all the interpretability measures considered: the minimum number of rules, minimum number of conditions per rule, and minimum number of conditions of the classifier. The experimental study includes a statistical analysis based on the Bonferroni–Dunn [24] and Wilcoxon [59] non-parametric tests [28,29] in order to evaluate whether there are statistically differences in the results of the algorithms.

This paper is structured as follows. Section 2 briefly reviews the related background works. Section 3 describes the ICRM algorithm. Section 4 describes the experimental study whose results are discussed in Section 5. Finally, Section 6 draws some conclusions raised from the work.

2. Background

This section introduces the accuracy vs interpretability problem and discusses the interpretability definition and metrics. Finally, it briefly reviews the most important works related to genetic rule-based classification systems in recent years.

2.1. Accuracy vs interpretability

Classification with rule-based systems comes with two contradictory requirements in the obtained model: the interpretability, capability the behavior of the real system in an understandable way, and the accuracy, capability to faithfully represent the real system. Obtaining high degrees of interpretability and accuracy is a contradictory purpose and, in practice, one of the two properties prevails over the other. To find the best trade-off between them is an optimization problem that is very difficult to solve efficiently [36].

In contrast, when looking for interpretability, fuzzy-based systems are usually considered [5,41]. These systems comprise very interpretable rules since they employ linguistic variables to address the vagueness of human language [44]. One of these systems is the algorithm by *González and Pérez*, SLAVE (Structural Learning Algorithm on Vague Environment) [30], which is a genetic learning algorithm that uses the iterative approach to learn fuzzy rules. Another fuzzy rule-based algorithm is GFS-GP [52], which combines genetic programming operators with simulated annealing search. The use of evolutionary programming, genetic programming and grammars to construct classification rules has been widely used [7,20,25,66]. However, this interpretability perspective is not related to the search for simpler classifiers in terms of fewer number of rules and conditions, in which we are really interested.

2.2. Interpretability metrics

There is no well-established definition of the interpretability of a rule-based system. The interpretability of a rule set is very important, due to the fact that very large sets of rules or very complex rules are rather lacking in interest. In fact, studies have focused on reducing the complexity of rule-based classifiers [35,53]. Nevertheless, there are some indicators that allow us to estimate the interpretability and comprehensibility of a rule-based classifier, which are described by García et al. [27]:

these are the number of rules and the number of conditions. A complexity value of a rule-based classifier was provided by Nauck [43], who proposes an interpretability measure related to the number of conditions of the classifier and the number of classes to be covered:

$$complexity = \frac{m}{\sum_{i=1}^{r} n_i} \tag{1}$$

where m is the number of classes, r is the number of rules, and n_i is the number of conditions used in the ith rule. This measure is 1 if the classifier contains only one rule per class using one condition each and it approaches 0 the more rules and conditions are used. However, this measure can be up to 2 for a two class data set where the algorithm learns a classifier containing one rule with one condition and another default rule with no conditions predicting the default class.

2.3. Genetic rule-based systems

Genetic algorithms (GAs) evolve a population of individuals which correspond to candidate solutions to a problem. GAs have been used for learning rules (Genetic rule-based systems) [21], including crisp and fuzzy rules, and they follow two approaches for encoding rules within a population.

The first one represents an individual as a single rule (individual = rule). The rule base is formed by combining several individuals from the population (rule cooperation) or via different evolutionary runs (rule competition). This representation results in three approaches:

- Michigan: they employ reinforcement learning and the GA is used to learn new rules that replace the older ones via competition through the evolutionary process. These systems are usually called learning classifier systems [14,40], such as XCS [15] and UCS [12].
- Iterative Rule Learning (IRL): individuals compete to be chosen in every GA run. The rule base is formed by the best rules obtained when the algorithm is run multiple times. SLAVE [30] and HIDER [1] are examples which follow this model.
- Genetic Cooperative-Competitive Learning (GCCL): the whole population or a subset of individuals encodes the rule base. In this model, the individuals compete and cooperate simultaneously. This approach makes it necessary to introduce a mechanism to maintain the diversity of the population in order to avoid a convergence of all the individuals in the population. GP-COACH [11] or COGIN [31] follow this approach.

The second one represents an individual as a complete set of rules (individual = set of rules), which is also known as the Pittsburgh approach. The main advantage of this approach compared to the first one is that it allows of addressing the cooperation–competition problem, involving the interaction between rules in the evolutionary process [8,45]. Pittsburgh systems are generally slower, since they evolve more complex structures [39]. Therefore, studies have focused on improving the effectiveness and efficiency of the rule structure exploration [63]. Moreover, one of their main problems is controlling the number of rules, which increases the complexity of the individuals, adding computational cost to their evaluation and becoming an unmanageable problem. This problem is known as the bloat effect [9], i.e., a growth without control of the size of the individuals.

One method based on this approach is the Memetic Pittsburgh Learning Classifier System (MPLCS) [10], which evaluates several local search mechanisms that heuristically edit classification rules and rule sets to improve their performance. In order to avoid the bloat effect, they employ a rule deletion operator and a fitness function based on the minimum description length [9,50], which balances the complexity and accuracy of the rule set. Moreover, this system uses a windowing scheme that reduces the run-time of the system by dividing the training set into many non-overlapping subsets over which the fitness is computed at each GA iteration. Another Pittsburgh style method is ILGA [32], which fixes the number of rules a priori, i.e., the complexity regarding to the number of rules is let to the user.

The order of the rules in the rule base is critical when dealing with decision lists [51]. A decision list is an ordered list of conjunctive rules. The classifier predicts the class membership of the highest priority rule which matches the antecedent of the rule and the data instance. Some authors have used evolutionary algorithms to sort the rules a posteriori, such as Tan et al. and their the CO-Evolutionary Rule Extractor (CORE) algorithm [54]. This GCCL model uses a gene to represent the rule index and the chromosome represents the ordering of the rules. Thus, the ordering of the rules is optimized.

There are also hybrid models such as DTGA [19] by Carvalho and Freitas, which is a hybrid decision tree/genetic algorithm method. The idea of this hybrid method involves the concept of small disjuncts. A set of classification rules can be regarded as a logical disjunction of rules, so that each rule can be regarded as a disjunct. A small disjunct is a rule covering a small number of examples. However, although each small disjunct covers just a few examples, the set of all small disjuncts can cover a large number of examples.

3. The ICRM algorithm

This section describes the most relevant features and the execution model of the ICRM algorithm. This paper presents a comprehensive and extended version of the ICRM algorithm, whose initial results were reported in [17]. The algorithm consists of three phases. In the first phase, the algorithm creates a pool of rules that explore the attribute domains. In the second phase, the algorithm iterates to find classification rules and builds the classifier. Finally, the third phase optimizes the accuracy of the classifier. Fig. 1 shows the workflow of the algorithm. The following three sections describe the phases of the algorithm.

3.1. Phase 1: exploring the attribute domains

This phase explores the attribute domains, whose output is a pool of rules composed of a single attribute–value comparison. The idea is to find the best attribute–value comparisons to classify each of the classes. Fig. 2 shows the grammar used to represent single attribute–value comparisons.

The point is to ensure the exploration of the entire range of attribute domains. The domains can be categorical or numerical. On the one hand, categorical domains are constrained to a limited number of labels or nominal values. Therefore, finding a relevant label that best classifies a class is simple and it is a combinatorial problem with a relatively low number of solutions. The algorithm creates a single attribute–label comparison for each label and nominal relational operator.

On the other hand, numerical domains have infinite solutions. Therefore, a method to explore numerical domains is described now. The idea is to initially explore the range of the numerical domain at multiple points uniformly distributed (then, the rules' numerical values are optimized in Phase 2). The algorithm creates a single attribute–value comparison rule for each of these points and numerical relational operators. All these rules are stored in a pool and they will be used in Phase 2. The pseudo-code of the creation procedure of these single attribute–value comparison is shown in Algorithm 1.

Algorithm 1. Pool creation procedure

```
1: for $i$ = 1 to numberAttributes do
2:   if attribute (i) is numerical then
3:     step ← domain (i)/numberPoints
4:     left ← minValueDomain (i)
5:     for $j$ = 1 to numberPoints-1 do
6:       value ← left + j * step
7:       createComparison (≥attribute (i) value)
8:       createComparison (≤attribute (i) value)
9:     end for
10:  else if attribute (i) is categorical then
11:    for all label in domain(i) do
12:      createComparison (=attribute (i) label)
13:      createComparison (≠attribute (i) label)
14:    end for
15:  end if
16: end for
```

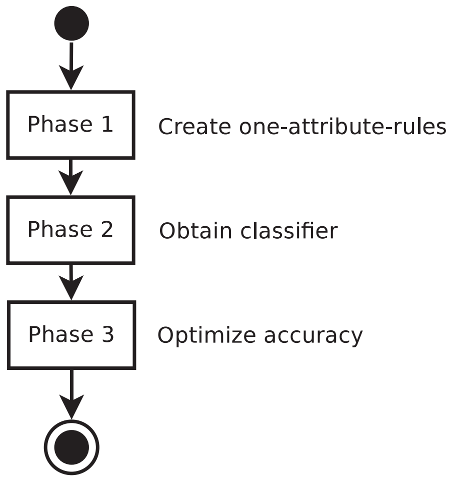


Fig. 1. Algorithm's workflow.

$$\begin{aligned}
&\langle S \rangle \to \langle cmp \rangle \\
&\langle cmp \rangle \to \langle op\_num \rangle \; \langle variable \rangle \; \langle value \rangle \\
&\langle cmp \rangle \to \langle op\_cat \rangle \; \langle variable \rangle \; \langle value \rangle \\
&\langle op\_num \rangle \to \geq \mid \leq \\
&\langle op\_cat \rangle \to = \mid \neq \\
&\langle variable \rangle \to Any\ valid\ attribute\ in\ data\ set \\
&\langle value \rangle \to Any\ valid\ value
\end{aligned}$$

Fig. 2. Grammar used to create single attribute–value comparisons.

3.2. Phase 2: obtaining the classifier

The objective of this phase is to obtain interpretable, accurate and complete classification rules. The input of this phase is the pool of single attribute–value comparisons created in the first phase and the output is a set of classification rules forming a classifier. Fig. 3 shows the workflow of this phase. The following sections detail the procedure to obtain the rules, the encoding of the individuals, the genetic operator, and the fitness function.

3.2.1. Obtaining classification rules

The goal of Phase 2 is to obtain a complete base of rules with as many rules as classes. Each rule predicts a single class and each class is predicted by a single rule. In each iteration of this procedure a classification rule is obtained. The rule predicts a class that has not been covered yet and which is the easiest class to classify among the remaining classes. To do so, it evolves in parallel as many evolutionary processes as classes to be covered. The output of each evolutionary process is a classification rule that predicts a unique class. The best rule among all the outputs from the evolutionary processes is selected and returned to be inserted into the classifier. The class predicted by this rule is set as covered and the algorithm removes the instances of that class from the training set. This process is repeated as many times as the number of classes but one, and the last class is set as the default class of the classifier.

This iterative procedure implies to evolve as many evolutionary algorithms as number of classes, i.e., that for a data set with N classes, the first iteration evolves N evolutionary algorithms, the second, $N-1$, the third, $N-2$, and so on, until there is only one remaining class to be covered. This may seem to require a high computational cost. However, in every iteration the number of classes to discern and the number of instances is lower than in the previous iteration, reducing the dimensionality of the problem (number of instances and classes to cover). Consequently, each iteration runs faster than the previous one. The experimental results and the execution times shown in Section 5.6 demonstrate the great efficiency of the algorithm.

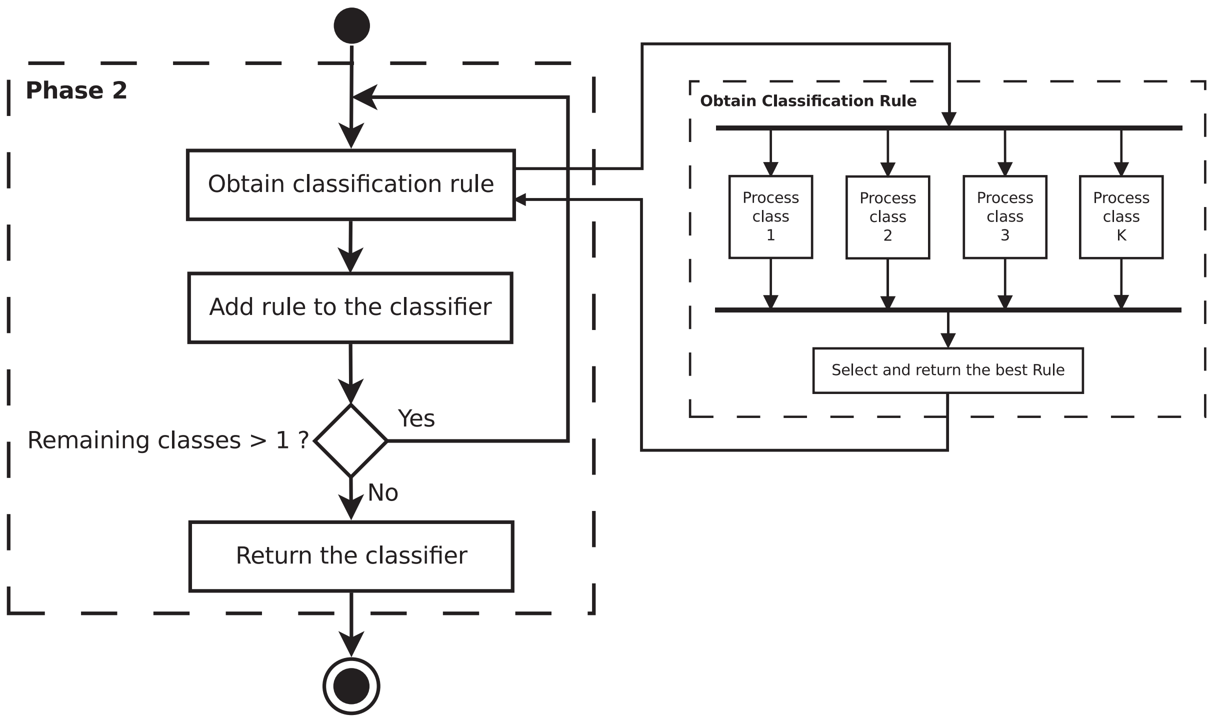


Fig. 3. Phase 2 workflow.

The following sections describe the individual representation, the overview of the algorithm, the genetic operator and the fitness function of the evolutionary algorithm employed to obtain the optimized classification rules which will comprise the classifier.

3.2.2. Individual representation

Phase 2 uses an individual = rule representation to learn interpretable IF-THEN rules that are inserted into a decision list forming a classifier. This representation provides greater efficiency, and addresses the cooperation–competition problem by dealing with the order of the rules in the evolutionary process. The antecedent of the rules represents a conjunction of attribute–value comparisons and the rule consequent contains the predicted class for the concepts satisfied by the antecedent of the rule. Fig. 4 shows the grammar used to represent the rules, i.e., a rule is a conjunction of one or more attribute–value comparisons.

3.2.3. Evolutionary rule generation

The learning process of the algorithm performs as a generational and elitist evolutionary algorithm. The evolutionary process will find the conjunction of attribute–value comparisons over the relevant attributes to discriminate a class from the other classes, by means of the genetic operator. For each rule, the genetic operator employs the not-yet covered attributes of the individual's rule to find the best condition over any other attribute that, appended to the rule, improves its accuracy. Therefore, the maximum number of generations is set as the number of attributes. The survival of the best individuals is guaranteed by copying them to the next generation. Fig. 5 shows the workflow of the evolutionary algorithm.

The genetic operator adds new clauses to the rule. When a new condition is added, the search area and the number of instances covered by the rule are both reduced. Thus, rather than search new conditions on the entire training set, the

$$
\begin{aligned}
&\langle S\rangle \rightarrow \langle cmp\rangle \mid \langle S\rangle\ AND\ \langle cmp\rangle\\
&\langle cmp\rangle \rightarrow \langle op\_num\rangle\ \langle variable\rangle\ \langle value\rangle\\
&\langle cmp\rangle \rightarrow \langle op\_cat\rangle\ \langle variable\rangle\ \langle value\rangle\\
&\langle op\_num\rangle \rightarrow\ \geq \mid \leq\\
&\langle op\_cat\rangle \rightarrow\ = \mid \neq\\
&\langle variable\rangle \rightarrow Any\ valid\ attribute\ in\ data\ set\\
&\langle value\rangle \rightarrow Any\ valid\ value
\end{aligned}
$$

Fig. 4. Rule = conjunction of attribute–value comparisons.

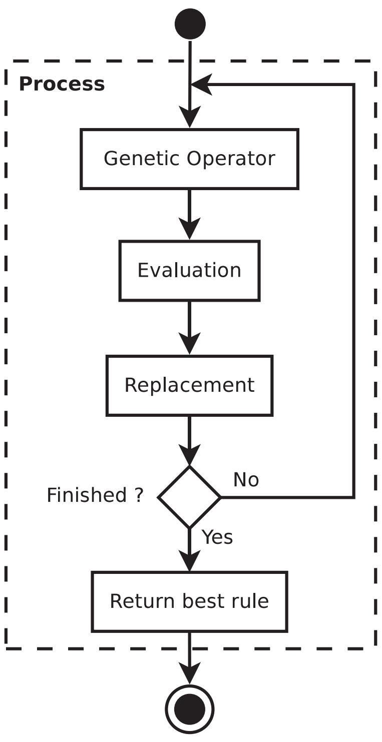


Fig. 5. Rule generation workflow.

genetic operator focuses on finding which attribute is now relevant in the training subset of instances covered by the rule. The genetic operator will return a condition to be appended to the rule, creating a new individual. If the genetic operator did not find a relevant attribute that improves the accuracy of the rule, the individual is marked as unbeatable. Moreover, once an individual is constrained to all the attributes, it is also marked as unbeatable (all the attributes have been covered).

The evaluation method computes the fitness value for each individual created by the genetic operator. The higher the fitness value is, the better the rule classifies.

Finally, the replacement method selects the best individuals from the previous generation and the offspring as the population for the next generation, keeping the population size constant. The algorithm finishes when the maximum number of generations is reached or all the individuals from the population are marked as unbeatable. Finally, the best individual is returned, which maximizes the fitness function and is constrained only to relevant attributes.

3.2.4. Genetic operator

This section describes the genetic operator designed to find the relevant attributes that, appended to a parent rule, improves its fitness. The parameters of this operator are the pool of single attribute–value comparisons, the class to predict, and the training subset of instances covered by the parent rule. The operator executes a hill-climbing microgenetic algorithm (MGA) [37] to optimize the numerical values of the attribute–value comparisons over the instances subset. The operator returns a new attribute–value comparison that best discriminates the predicted class among the other classes. This new attribute–value comparison is appended to the parent rule, creating a new individual.

The optimization of the numeric values of the attribute–value comparisons is treated as an optimization problem of a continuous variable real function. The MGA uses a mutation operator to approximate the numerical values of the rules to the maximum of the function. Each rule mutates the value of the comparison within a close range [− *step*, + *step*] using the *step* size described in Algorithm 1. The selection procedure is responsible for ensuring the survival of the best individuals and after a small number of generations, the individuals converge to the value that best classifies the target class.

3.2.5. Fitness function

The results of the matching of the rules and the instances from the data set are used to build the confusion matrix, which is a table with two rows and two columns that reports the number of true positives (T_P), false positives (F_P), true negatives (T_N), and false negatives (F_N). The algorithm fitness function uses these values and combines two indicators that are commonplace in the domain, namely the sensitivity (Se) and the specificity (Sp). Sensitivity, also called recall in some fields, measures the proportion of actual positives which are correctly identified as such, whereas specificity measures the proportion of negatives which are correctly identified.

$$Se = \frac{T_P}{T_P + F_N} \qquad Sp = \frac{T_N}{T_N + F_P}$$

The fitness value is computed by means of the sensitivity and the specificity in order to maximize both metrics. In addition, to ensure the better interpretability of the rules represented by the individuals, for equally accurate rules, the simplest one with the lowest number of conditions prevails.

$$Fitness = Se * Sp$$

3.3. Phase 3: optimizing the accuracy

This section describes the evolutionary process for optimizing the classifier obtained in Phase 2. The rules of the classifier have been selected and optimized individually to maximize the product of sensitivity and specificity. However, the values from the numerical comparisons of the rules can be slightly tuned to improve the overall accuracy of the classifier. Therefore, Phase 3 runs an evolutionary process which improves the accuracy of the classifier.

Phase 3 uses an individual = classifier representation. The population is initialized using the classifier provided by Phase 2 as the seed. The algorithm iterates mutating the rules of the classifiers and selecting the best offspring until a certain number of generations have been performed. The population size and the maximum number of generations are parameters specified in Section 4.3.

The mutation operator is applied to every individual within the population. Given a parent individual, it creates as many new individuals as there are rules in the parent classifier in order to tune the comparisons of the rules to improve the overall performance of the classifier. Every new individual is a copy of the parent, but one rule has been mutated. The mutation of a rule performs as a local search and tunes the values from the attribute–value comparisons within a close range [− *step*, + *step*] using the *step* size described in Algorithm 1, similar to the MGA described in the genetic operator from Phase 2.

The fitness function computes the fitness value of the individuals from the population. In Phase 3, the fitness of an individual is the accuracy of the classifier, since all the individuals have the same complexity. The accuracy is the ratio of successfully classified instances, i.e., the number of instances where the class prediction was correct compared to the total number of instances.

4. Experimental study

This section describes the details of the experiments performed in various problem domains to evaluate the capabilities of the proposal and compare it to other classification methods.

The experiments carried out compare the results of the ICRM algorithm and nine other classification algorithms over 35 data sets. These data sets were collected from the KEEL repository website [3] and the algorithms are available on the KEEL software tool [4]. The data sets together with their partitions are available to facilitate the replicability of the experiments and future comparisons with previous or new proposals.[1]

First, the application domains and the algorithms used for the comparison are presented. Second, an analysis of the parameters of ICRM algorithm is performed in order to find appropriate parameter settings. Third, a brief description of the experimental methodology and computed metrics is given. Finally, the statistical tests used for validating the results are presented.

4.1. Problem domains

The data sets used in the experiments were collected from the KEEL repository website [3] and are very varied in their degrees of complexity, number of classes, number of attributes and number of instances. The number of classes ranges up to 10, the number of attributes ranges from 3 to 60 and the number of instances ranges from 101 to 10,992. Table 1 summarizes the information about these data sets, which are available through the link 1.

4.2. Classification algorithms

This section describes the algorithms used in our experiments, which are obtained from the KEEL software tool [4]. The most relevant rule-based proposals presented to date are compared to determine whether our evolutionary algorithm is competitive in the different domains with the other approaches. Six of the methods were presented in the background section (MPLCS, ILGA, CORE, SLAVE, GFS-GP, DTGA), but we also consider interesting to compare with three other well-known methods, including one non-evolutionary crisp rule method (RIPPER), the classical decision tree (C4.5) from which extract rules (C45R), and one ant colony method (Ant Miner+).

- MPLCS [10]: a Memetic Pittsburgh Learning Classifier System that combines LS operators and policies.
- ILGA [32]: an Incremental Learning approach to Genetic Algorithms, with different initialization schemes based on different initializations of the GA population.
- CORE [54]: COevolutionary Rule Extractor that coevolves rules and rule sets concurrently in two cooperative populations to confine the search space and to produce good rule sets that are comprehensive.
- SLAVE [30]: Structural Learning Algorithm on Vague Environment is a genetic learning algorithm that uses the iterative approach to learn fuzzy rules.
- GFS-GP [52]: GP algorithm used to learn fuzzy rule-based classifiers.
- DTGA [19]: hybrid decision tree/ genetic algorithm discovering rules on small disjuncts.
- Ant Miner+ [47]: Ant Colony-based data miner to extract classification rules inspired by the research on the behavior of real ant colonies.
- RIPPER [22]: improvement of the efficient incremental reduced error pruning (IREP) algorithm.
- C45R [48]: C45Rules reads the decision tree or trees produced by C4.5 and generates a set of production rules from each tree and from all trees together.

Many different configurations have been established by the authors of each paper for the different techniques. The parameters used for the experimental study in all classification methods are the optimal values provided by their authors.

4.3. ICRM parameters

This section discusses the ICRM parameter settings to determine the influence of each parameter and their combinations in terms of accuracy and computational cost. ICRM has been implemented in the JCLEC software [56] and its main parameters are shown in Table 2. The three different phases of the algorithm have some parameters whose optimal values are shown in Table 2. Phase 1 requires the number of points in which the domains are initially explored. Phase 2 requires a selection pressure value in $(0,1]$, which determines the number of relevant attributes to explore regarding to the number of attributes. The genetic operator from Phase 2 requires two parameters: the population size and the number of generations of the MGA which optimizes the rules. Phase 3 also requires two parameters: the population size and the number of generations of the algorithm that optimizes the full classifier. The effect of these parameters has no relevance in the accuracy results, but is

[1] http://www.uco.es/grupos/kdis/kdiswiki/ICRM.

Table 1
General information about the data sets.

Data set	# Instances	# Attributes	# Classes
Appendicitis	106	7	2
Australian	690	14	2
Balance	625	4	3
Breast	286	9	2
Bupa	345	6	2
Car	1728	6	4
Chess	3196	36	2
Contraceptive	1473	9	3
Dermatology	366	34	6
Ecoli	336	7	8
Flare	1066	11	6
German	1000	20	2
Glass	214	9	7
Haberman	306	3	2
Heart	270	13	2
Ionosphere	351	33	2
Iris	150	4	3
Lymphography	148	18	4
Monk-2	432	6	2
New-thyroid	215	5	3
Page-blocks	5472	10	5
Penbased	10,992	16	10
Pima	768	8	2
Saheart	462	9	2
Satimage	6435	36	7
Segment	2310	19	7
Sonar	208	60	2
Tae	151	5	3
Thyroid	7200	21	3
Tic–tac–toe	958	9	2
Twonorm	7400	20	2
Vehicle	846	18	4
Wine	178	13	3
Yeast	1484	8	10
Zoo	101	16	7

Table 2
Parameter of ICRM algorithm.

	Parameter	Value
Phase 1	Number points	10
	Selection pressure	0.5
Phase 2	MGA population	10
	MGA generations	10
Phase 3	Fitting population	5
	Fitting generations	100

significant as to the computational time, increasing it considerably as long as the population size or the number of generations increases.

4.4. Experimental settings

The experiments consider the following metrics: the predictive accuracy, the number of rules of the classifier, the number of conditions per rule, the number of conditions per classifier, the complexity measure from Eq. (1), and the execution time. All experiments are repeated with 10 different seeds for stochastic methods and the average results are shown in the results tables. All algorithms are tested using 10-fold stratified cross validation [38,58] on all data sets. Experiments were run on a PC equipped with an Intel Core i7 quad-core processor running at 2.66 GHz and 12 GB of DDR3 memory. The host operating system was GNU/Linux Ubuntu 12.10 64 bit.

4.5. Statistical analysis

In order to analyse the results from the experiments, some non-parametric statistical tests are used to validate the results and conclusions [28,29]. To evaluate whether there are significant differences in the results of the different algorithms, the

Iman and Davenport test is performed. This useful non-parametric test, recommended by Demsar [23], is applied to rank the K algorithms over the N data sets (in our case there re 10 algorithms and 35 data sets) according to the F-distribution. When the Iman and Davenport test indicates that the results are significantly different, the Bonferroni–Dunn post hoc test [24] is used to find the significant differences occurring between algorithms in the multiple comparison. It assumes that the performance of two classifiers is significantly different if the corresponding average ranks differ by at least a critical difference value. Finally, the Wilcoxon rank-sum test [59] is used to perform multiple pairwise comparisons among the algorithms. The Wilcoxon rank-sum test statistic is the sum of the ranks for observations from one of the samples.

5. Results

This section discusses the experimental results and compares our method to different algorithms. In order to demonstrate the effectiveness and efficiency of our model, the accuracy, the execution time, and the different interpretability measures are evaluated.

5.1. Predictive accuracy

The predictive accuracy determines the ratio of successfully classified patterns by the model. Obviously, this is one of the most relevant factors when designing a classification algorithm, since it is desired to be as accurate as possible when making the predictions.

Table 3 shows the average accuracy results for the test folds and the ranking of the algorithms. The best values for accuracy, at a ranking value of 2.4286, are obtained by the MPLCS algorithm. The Iman and Davenport statistic for average accuracy distributed according to F-distribution with $K - 1 = 9$ and $(K - 1)$ $(N - 1) = 306$ degrees of freedom is 18.8472. The test establishes an F-distribution value = 2.4656 for a significance level of alpha = 0.01. This value is lower than the

Table 3
Accuracy results from the test folds of the 10-fold cross-validation.

Accuracy	ICRM	C45R	RIPPER	MPLCS	AntMin+	CORE	ILGA	SLAVE	DTGA	GFS-GP
Appendicitis	0.8551	0.8517	0.8300	**0.8623**	0.8565	0.8415	0.8498	0.7063	0.8327	0.8518
Australian	0.8551	0.8517	0.8300	**0.8623**	0.8565	0.8415	0.8498	0.7063	0.8609	0.8449
Balance	0.7985	**0.8128**	0.5168	0.8043	0.7041	0.6460	0.5003	0.7460	0.7520	0.7234
Breast	0.7162	0.6970	0.6589	0.7186	0.7269	0.7234	0.7404	0.7080	0.7546	**0.7691**
Bupa	0.5961	0.6505	0.6232	0.6226	0.4154	0.5912	0.5540	0.5877	**0.6589**	0.5737
Car	0.8078	0.8794	0.8957	**0.9890**	0.8080	0.7982	0.7002	0.7002	0.8310	0.7882
Chess	0.7741	**0.9945**	0.9931	0.9927	0.8673	0.5438	0.9662	0.5222	0.9900	0.7650
Contraceptive	0.5119	0.5114	0.5169	**0.5484**	0.4626	0.4628	0.4417	0.4368	0.5275	0.4847
Dermatology	**0.9635**	0.9436	0.9300	0.9561	0.8710	0.3594	0.5877	0.9165	0.9408	0.6667
Ecoli	0.7774	0.7767	0.7482	0.8089	0.7409	0.6549	0.6352	**0.8444**	0.7709	0.7022
Flare	0.6857	0.6839	0.6720	0.7296	0.7049	0.6592	0.6273	0.0000	**0.7402**	0.5826
German	0.6920	0.7000	0.6610	**0.7250**	0.6840	0.6930	0.7100	0.7050	0.7120	0.7100
Glass	**0.6855**	0.6805	0.6397	0.6541	0.4936	0.5026	0.5013	0.5961	0.6117	0.5515
Haberman	**0.7459**	0.7119	0.4782	0.7349	0.7049	0.7394	0.7200	0.7178	0.7348	0.7156
Heart	0.7481	0.7901	0.7568	**0.8136**	0.8037	0.7198	0.6543	0.7975	0.7778	0.7370
Ionosphere	0.8578	0.9005	0.8606	0.9173	0.8550	0.5810	0.8206	**0.9259**	0.8890	0.8091
Iris	**0.9707**	0.9667	0.9378	0.9578	0.9356	0.9467	0.9289	0.9533	0.9600	0.9333
Lymphography	0.8031	0.7360	0.7475	**0.8074**	0.7420	0.6332	0.7769	0.6875	0.7425	0.7187
Monk-2	0.9727	**1.0000**	**1.0000**	0.9955	0.9727	0.9284	0.5437	0.9727	**1.0000**	0.9471
New-thyroid	0.9275	**0.9353**	0.9320	0.9242	0.8916	0.9136	0.9166	0.9013	0.9208	0.8374
Page-blocks	0.9502	0.9551	0.9646	0.9434	0.9376	0.9038	0.9278	0.9355	**0.9662**	0.9331
Penbased	0.7533	0.9490	**0.9648**	0.9420	0.4728	0.1569	0.5325	0.9322	0.9377	0.5330
Pima	0.7411	0.7183	0.6923	**0.7483**	0.7210	0.7297	0.7336	0.7337	0.7331	0.7293
Saheart	**0.6919**	0.6804	0.6002	0.6870	0.6795	0.6883	0.6730	0.6406	0.6773	0.6861
Satimage	0.7587	0.8406	0.8552	**0.8612**	0.7063	0.3978	0.7293	0.6566	0.8281	0.7346
Segment	0.9177	0.9515	**0.9537**	0.9532	0.8121	0.4294	0.8264	0.8784	0.9494	0.6571
Sonar	0.6939	0.7055	0.7528	**0.7733**	0.7463	0.5338	0.7080	0.7363	0.6576	0.6969
Tae	0.4918	0.4549	0.4929	**0.5706**	0.3312	0.4500	0.4251	0.4747	0.3783	0.4629
Thyroid	0.9879	**0.9960**	0.9947	0.9457	0.9826	0.7589	0.9405	0.9304	0.9950	0.9301
Tic–tac–toe	0.8993	0.8423	0.9760	**1.0000**	0.9833	0.6986	0.7811	0.6535	0.7880	0.7599
Twonorm	0.8607	0.8677	0.9142	0.8797	0.7878	0.6811	**0.9284**	0.8516	0.8315	0.7939
Vehicle	0.6312	0.6666	0.6915	0.7053	0.6332	0.3853	0.5853	0.6369	**0.7174**	0.5224
Wine	0.9356	**0.9490**	0.9377	0.9227	0.8690	0.9439	0.8893	0.9211	**0.9490**	0.8925
Yeast	0.4428	0.5594	0.5054	**0.5755**	0.4324	0.3767	0.4252	0.5013	0.4812	0.4628
Zoo	0.9583	0.9281	0.9269	**0.9656**	0.4990	0.9325	0.8587	0.0000	0.8561	0.8236
Avg. values	0.7845	0.8040	0.7843	**0.8256**	0.7340	0.6528	0.7140	0.7033	0.7930	0.7237
Avg. ranks	4.2857	3.9000	4.7714	**2.4286**	6.7143	7.7429	7.2000	6.6429	4.1286	7.1857

statistic critical value 18.8472. Thus, the test rejects the null hypothesis and therefore it can be said that there are statistically significant differences between the accuracy results of the algorithms.

Fig. 6 shows the application of the Bonferroni–Dunn test to accuracy for alpha = 0.01, whose critical difference is 2.3601. This graph is a bar chart, whose values are proportional to the mean rank obtained from each algorithm. The critical difference value is represented as a thicker horizontal line and those values that exceed this line are algorithms with significantly different results than the control algorithm, which is the one with the lowest rank value. Therefore, the algorithms right beyond the critical difference are significantly worse than the control algorithm. For accuracy, MPLCS is the best ranked algorithm and therefore, it is the control algorithm. Observing this figure, C45R, DTGA, ICRM and RIPPER achieve statistically similar accuracy results than MPLCS. On the other hand, AntMiner+, CORE, ILGA, SLAVE, and GFS-GP perform statistically significantly worse.

Table 4 shows the results of the Wilcoxon rank-sum test for the accuracy to compute multiple pairwise comparisons among the ICRM algorithm and the other methods. The p-values reported indicate that no significant differences can be found when comparing ICRM vs C45R, RIPPER, MPLCS, and DTGA. However, these differences are significant when compared to AntMiner+, CORE, ILGA, SLAVE, and GFS-GP, achieving p-values lower than 0.01, i.e., a statistical confidence higher than 99%.

5.2. Number of rules

The number of rules determines the complexity of the model. The greater the number of rules, the greater probability of conflicts between them and the greater difficulty in understanding the conditions necessary to predict a particular class.

Table 5 shows the average number of rules of the classifiers and the ranking of the algorithms. The best values are the lowest number of rules, at a ranking value of 1.63, ICRM performs better than the others and it is followed by CORE and ANTMiner+. The Iman and Davenport statistic for the number of rules distributed according to an F-distribution with $K - 1 = 9$ and $(K - 1)(N - 1) = 306$ degrees of freedom is 104.4333. The test establishes an F-distribution value = 2.4656 for a significance level of alpha = 0.01. This value is lower than the statistic critical value 104.4333. Thus, the test rejects the null hypothesis and therefore it can be said that there are statistically significant differences between the number of rules of the algorithms.

Fig. 7 shows the application of the Bonferroni–Dunn test to the number of rules for alpha = 0.01, whose critical difference is 2.3601. The algorithms right beyond the critical difference from the ICRM rank are significantly worse than ICRM with a confidence level higher than 99%. Observing this figure, all the algorithms but ANTMiner + and CORE are significantly worse than ICRM, whereas MPLCS is borderline.

Table 6 shows the results of the Wilcoxon rank-sum test for the number of rules to compute multiple pairwise comparisons among the proposal and the other methods. The p-values reported indicate significant differences in the number of rules of the algorithms with a confidence level higher than 99%. All methods are shown to obtain classifiers with significantly higher number of rules.

5.3. Conditions per rule

The number of conditions in a rule determines the length and complexity of a rule. The greater the number of conditions, the lower the interpretability of the rule.

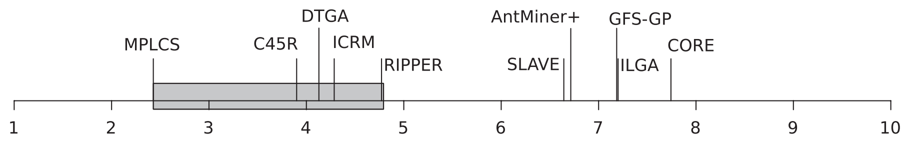


Fig. 6. Bonferroni-Dunn for accuracy.

Table 4
Wilcoxon test for accuracy.

ICRM vs	R^+	R^-	p-value
C45R	224.0	406.0	⩾0.2
RIPPER	296.0	334.0	⩾0.2
MPLCS	103.0	527.0	⩾0.2
AntMiner+	471.5	123.5	0.002256
CORE	618.0	12.0	4.074E−9
ILGA	555.0	75.0	2.612E−5
SLAVE	465.0	130.0	0.003394
DTGA	272.0	358.0	⩾0.2
GFS-GP	588.0	42.0	6.602E−7

Table 5
Number of rules.

# Rules	ICRM	C45R	RIPPER	MPLCS	AntMin+	CORE	ILGA	SLAVE	DTGA	GFS-GP
Appendicitis	**2.00**	11.07	23.53	4.83	5.50	4.03	30.00	6.97	3.00	219.40
Australian	**2.00**	11.07	23.53	4.83	5.50	4.03	30.00	6.97	27.70	197.75
Balance	**3.00**	34.80	37.47	14.83	6.13	4.23	30.00	24.10	53.70	83.10
Breast	2.00	14.60	35.60	17.90	7.50	5.20	30.00	**1.00**	11.00	175.25
Bupa	**2.00**	8.57	23.80	6.60	**2.00**	3.43	30.00	6.30	30.10	174.90
Car	4.00	76.87	68.67	19.73	25.13	3.33	30.00	**1.00**	131.80	82.55
Chess	2.00	30.40	22.50	9.70	16.20	14.00	30.00	**1.00**	34.00	153.85
Contraceptive	**3.00**	31.00	64.60	5.67	3.13	8.93	30.00	44.90	138.80	82.85
Dermatology	**6.00**	9.20	14.27	7.77	6.70	6.47	30.00	10.57	10.30	118.10
Ecoli	8.00	11.40	35.50	8.90	7.63	**6.33**	30.00	12.77	27.70	79.10
Flare	6.00	27.90	101.60	16.67	20.30	4.80	30.00	**0.00**	49.60	110.65
German	**2.00**	27.20	36.30	12.40	14.80	3.30	30.00	8.80	84.10	71.05
Glass	7.00	10.07	22.20	8.27	**4.57**	7.03	30.00	16.27	39.50	176.25
Haberman	**2.00**	4.30	17.67	4.37	**2.00**	4.13	30.00	5.63	2.60	230.40
Heart	**2.00**	10.83	13.97	6.90	4.87	5.47	30.00	7.47	17.70	100.30
Ionosphere	**2.00**	12.30	11.60	4.70	9.80	2.30	30.00	4.30	14.80	153.25
Iris	3.00	5.00	6.30	4.07	**2.97**	3.63	30.00	3.00	4.70	85.95
Lymphography	4.00	11.60	13.10	7.57	**3.70**	6.43	30.00	4.37	17.90	84.75
Monk-2	**2.00**	6.00	4.00	4.00	3.00	3.57	30.00	3.00	5.00	121.05
New-thyroid	**3.00**	6.80	6.93	5.43	3.57	3.30	30.00	5.50	8.70	185.65
Page-blocks	**5.00**	22.50	51.23	7.10	14.70	5.43	30.00	9.90	51.60	65.85
Penbased	10.00	125.50	109.90	47.70	82.70	**6.00**	30.00	40.20	201.10	56.00
Pima	**2.00**	8.40	25.10	5.43	12.30	3.13	30.00	8.57	21.10	156.10
Saheart	**2.00**	6.70	24.57	6.23	3.70	6.37	30.00	11.30	11.90	112.95
Satimage	7.00	75.10	109.00	30.90	69.70	**1.00**	30.00	36.60	334.30	85.45
Segment	7.00	26.30	33.20	16.90	35.00	**4.40**	30.00	18.80	50.60	137.75
Sonar	2.00	8.70	8.13	6.17	4.83	**1.00**	30.00	8.37	20.20	97.25
Tae	3.00	7.57	28.43	5.57	**2.00**	6.47	30.00	11.03	27.30	224.60
Thyroid	**3.00**	10.33	11.43	4.03	24.17	3.97	30.00	6.93	19.60	108.30
Tic–tac–toe	2.00	50.40	16.00	9.00	9.00	3.43	30.00	**1.00**	83.20	211.15
Twonorm	**2.00**	57.70	56.60	23.20	39.10	7.10	30.00	35.80	287.70	96.30
Vehicle	**4.00**	19.80	46.70	15.17	25.97	6.77	30.00	34.77	69.30	156.50
Wine	**3.00**	5.00	5.47	4.47	3.77	3.07	30.00	4.03	5.70	96.35
Yeast	10.00	36.40	140.10	13.70	36.00	**7.20**	30.00	23.20	174.70	121.70
Zoo	7.00	8.70	9.10	7.00	4.20	7.03	30.00	**0.00**	13.00	80.15
Avg. values	**3.89**	23.72	35.95	10.79	14.92	5.04	30.00	12.13	59.54	128.36
Avg. ranks	**1.63**	6.23	7.44	3.99	3.83	2.54	7.54	4.20	7.94	9.66

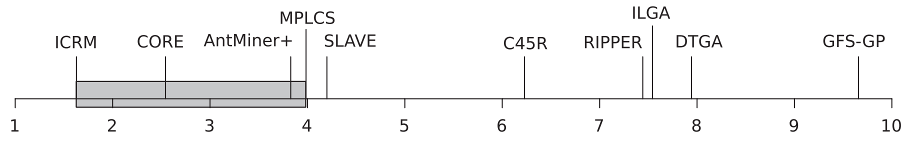


Fig. 7. Bonferroni-Dunn for the number of rules.

Table 6
Wilcoxon test for the number of rules.

ICRM vs	R^+	R^-	p-value
C45R	630.0	0.0	5.82E−11
RIPPER	630.0	0.0	5.82E−11
MPLCS	595.0	0.0	1.16E−10
AntMiner+	577.0	53.0	2.584E−6
CORE	464.0	166.0	0.013682
ILGA	630.0	0.0	5.82E−11
SLAVE	533.5	61.5	1.325E−5
DTGA	630.0	0.0	5.82E−11
GFS-GP	630.0	0.0	5.82E−11

Table 7 shows the average number of conditions per rule of the classifiers and the ranking of the algorithms. The best values are the lowest number of conditions, at a ranking value of 1.31, our method performs better than the others and it is followed distantly by AntMiner+. The Iman and Davenport statistic for the number of conditions per rule distributed

Table 7
Conditions per rule.

# Cond/ Rule	ICRM	C45R	RIPPER	MPLCS	AntMin+	CORE	ILGA	SLAVE	DTGA	GFS-GP
Appendicitis	**0.50**	3.15	3.22	3.49	4.00	2.21	8.36	3.52	1.43	3.45
Australian	**0.50**	3.15	3.22	3.49	4.00	2.21	8.36	3.52	3.66	2.86
Balance	1.27	3.12	3.33	2.39	2.49	**0.99**	1.95	2.75	3.44	3.73
Breast	0.55	3.40	3.28	5.63	1.01	1.86	7.51	**0.00**	1.59	2.54
Bupa	**0.90**	2.48	3.28	3.25	1.50	1.83	2.81	3.40	3.23	3.97
Car	1.95	3.98	4.40	3.60	3.59	2.42	5.20	**0.00**	2.99	4.48
Chess	1.00	3.74	3.37	5.82	1.07	10.64	29.18	**0.00**	9.05	3.24
Contraceptive	**1.00**	5.04	5.93	2.92	3.14	1.76	4.09	4.19	6.17	5.99
Dermatology	**1.43**	2.56	2.44	3.38	8.30	9.15	14.12	1.98	4.73	3.25
Ecoli	**1.89**	2.98	2.43	2.47	2.82	2.69	3.18	3.30	5.13	3.75
Flare	1.23	3.02	5.20	6.20	4.71	3.42	9.14	**0.00**	3.00	3.58
German	**0.50**	3.81	3.79	8.73	1.36	3.41	12.75	3.25	3.18	7.84
Glass	**2.06**	3.17	2.46	2.67	2.65	3.21	4.16	3.46	5.68	3.46
Haberman	**0.50**	1.31	2.63	1.54	0.65	0.94	1.47	2.19	1.80	1.80
Heart	**1.00**	2.68	2.52	3.97	4.07	2.03	5.48	3.59	2.73	4.09
Ionosphere	**1.00**	2.28	1.70	4.60	1.11	6.20	13.69	3.35	3.03	3.31
Iris	1.00	1.14	1.53	**0.87**	1.29	1.28	1.87	1.36	2.17	66.75
Lymphography	**1.25**	2.06	2.14	4.61	3.61	3.96	13.13	1.97	2.91	3.66
Monk-2	1.00	1.67	1.25	1.31	2.31	0.98	2.87	**0.67**	2.10	2.55
New-thyroid	**1.29**	1.87	1.55	1.69	1.51	2.38	2.34	1.67	2.35	21.51
Page-blocks	**1.43**	3.46	3.31	2.56	3.36	2.72	4.40	3.61	5.03	3.01
Penbased	1.70	5.67	4.05	4.55	**1.05**	3.61	6.61	7.21	5.11	2.99
Pima	**0.50**	2.34	4.06	3.35	4.72	1.50	3.60	3.42	3.78	3.76
Saheart	**0.50**	1.97	3.48	3.38	3.94	1.70	4.40	3.83	2.78	3.85
Satimage	1.39	5.76	4.75	4.79	**1.00**	2.00	16.55	6.54	10.46	6.55
Segment	1.41	3.78	2.96	3.42	**1.03**	4.26	8.28	4.22	4.65	3.75
Sonar	**0.89**	2.66	1.93	6.43	14.82	15.00	25.09	6.07	39.26	3.97
Tae	**1.17**	2.17	2.91	2.50	**1.17**	1.37	2.27	2.94	3.84	3.15
Thyroid	**1.38**	2.70	3.97	1.94	10.34	2.98	9.43	3.12	5.53	4.09
Tic–tac–toe	0.50	3.31	3.68	2.67	2.67	1.69	7.60	**0.00**	3.72	3.46
Twonorm	**0.55**	5.14	4.92	5.47	1.00	4.32	9.42	5.74	5.10	5.68
Vehicle	**1.25**	3.60	3.06	3.72	6.60	3.46	8.03	5.96	4.24	3.50
Wine	**1.47**	1.62	1.72	2.18	2.26	4.34	5.71	3.14	2.60	4.11
Yeast	**1.18**	4.74	4.21	2.49	1.53	3.31	3.77	3.56	6.08	3.55
Zoo	1.14	2.11	1.94	1.39	3.26	4.85	13.53	**0.00**	5.31	14.30
Avg. values	**1.09**	3.08	3.16	3.53	3.25	3.45	8.01	2.96	5.08	6.44
Avg. ranks	**1.31**	5.11	5.29	5.39	4.94	4.57	8.63	5.50	7.16	7.10

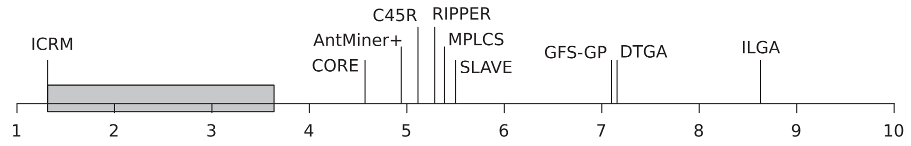


Fig. 8. Bonferroni–Dunn for the number of conditions per rule.

according to an F-distribution with $K-1=9$ and $(K-1)(N-1)=306$ degrees of freedom is 23.8287. The test establishes an F-distribution value = 2.4656 for a significance level of alpha = 0.01. Thus, the test rejects the null hypothesis and therefore it can be said that there are statistically significant differences between the number of conditions per rule of the algorithms.

Fig. 8 shows the application of the Bonferroni-Dunn test to the number of conditions per rule for alpha = 0.01, whose critical difference is 2.3601. The algorithms right beyond the critical difference from the ICRM value are significantly worse than ICRM. All the algorithms show significant differences compared to our proposal, which is the one that obtains the best results. Moreover, our proposal employs an attribute only once at most, performing a selection of relevant features. Looking at the results of Table 7, there are significant differences, e.g., between our proposal and ILGA over the two classes and 60 attributes Sonar data set. While our proposal uses on average 0.89 conditions per rule, i.e., almost one rule with one condition and another default class rule, ILGA requires 25.09 conditions per rule. Therefore, ICRM considers only one of the 60 attributes as relevant to discriminate between the two classes, while ILGA considers 25. In this case the predictive accuracy difference between ICRM and ILGA is only 1.4%.

Table 8 shows the results of the Wilcoxon rank-sum test for the number of conditions per rule to compute multiple pairwise comparisons among the proposal and the other methods. The p-values reported indicate significant differences with a confidence level higher than 99% from ICRM versus all.

Table 8
Wilcoxon test for the number of conditions per rule.

ICRM vs	R^+	R^-	p-value
C45R	630.0	0.0	5.82E−11
RIPPER	630.0	0.0	5.82E−11
MPLCS	629.0	1.0	1.16E−10
AntMiner+	567.0	28.0	1.726E−7
CORE	625.5	4.5	4.94E−10
ILGA	630.0	0.0	5.82E−11
SLAVE	574.5	55.5	3.451E−6
DTGA	630.0	0.0	5.82E−11
GFS-GP	630.0	0.0	5.82E−11

5.4. Conditions per classifier

The total number of conditions of the classifier determines its overall complexity, regardless of the number of rules, since it measures the number of conditions that must be evaluated to classify instances. The greater the number of conditions, the lower the interpretability of the classifier and the more time required to evaluate the conditions.

Table 9 shows the average number of conditions of the classifiers and the ranking of the algorithms. The best ranking values are the lowest number of conditions, at an average ranking value of 1.27, our method performs better than the others and it is followed distantly by CORE. The Iman and Davenport statistic for the number of conditions per classifier distributed according to an F-distribution with $K - 1 = 9$ and $(K - 1)(N - 1) = 306$ degrees of freedom is 86.5088. The test establishes an F-distribution value = 2.4656 for a significance level of alpha = 0.01. Thus, the test rejects the null hypothesis and therefore it can be said that there are statistically significant differences between the number of conditions per classifier of the algorithms.

Table 9
Conditions per classifier.

# Cond/ Clrf	ICRM	C45R	RIPPER	MPLCS	AntMin+	CORE	ILGA	SLAVE	DTGA	GFS-GP
Appendicitis	**1.00**	35.17	75.93	17.20	23.20	10.50	250.83	24.70	4.70	761.60
Australian	**1.00**	35.17	75.93	17.20	23.20	10.50	250.83	24.70	102.10	544.70
Balance	**3.80**	108.60	124.83	35.27	15.80	4.37	58.40	66.53	182.50	316.80
Breast	1.10	49.70	117.10	101.40	7.60	10.20	225.30	**0.00**	19.20	438.70
Bupa	**1.80**	21.37	78.47	21.60	3.00	6.03	84.17	21.60	97.80	619.50
Car	7.80	306.20	302.70	71.30	90.23	8.00	155.90	**0.67**	393.60	371.20
Chess	2.00	113.80	76.20	56.90	17.40	149.00	875.50	**0.00**	301.90	422.50
Contraceptive	**3.00**	156.43	382.50	17.27	11.73	16.77	122.70	188.00	846.70	442.10
Dermatology	**8.60**	23.60	34.90	26.23	55.17	62.27	423.73	21.20	47.20	381.10
Ecoli	**15.14**	34.10	86.40	21.80	21.60	17.07	95.43	42.03	129.90	322.70
Flare	7.40	84.57	528.40	103.80	95.80	16.23	274.10	**0.00**	147.50	381.50
German	**1.00**	103.80	137.30	108.30	20.10	12.20	382.40	29.50	270.40	301.30
Glass	14.42	31.87	54.67	22.07	**12.23**	22.70	124.67	56.47	204.90	601.10
Haberman	**1.00**	5.80	46.60	6.70	1.30	4.03	44.10	12.30	4.70	728.30
Heart	**2.00**	29.40	35.30	27.47	19.93	12.97	164.50	27.07	48.60	446.20
Ionosphere	**2.00**	28.30	19.70	20.70	10.90	13.70	410.80	13.40	45.10	488.10
Iris	**3.00**	5.70	9.70	3.57	3.83	4.70	55.97	4.07	10.30	539.80
Lymphography	**5.00**	23.97	28.10	34.97	13.47	26.80	393.87	8.40	51.90	309.90
Monk-2	**2.00**	10.00	5.00	5.23	6.93	4.20	86.07	**2.00**	10.50	302.20
New-thyroid	**3.86**	12.70	10.73	9.27	5.33	7.80	70.17	9.10	20.60	770.90
Page-blocks	**7.17**	78.20	169.47	18.50	49.40	14.67	131.97	35.77	253.20	196.10
Penbased	**17.00**	710.90	444.90	220.50	86.70	21.70	198.40	291.20	1,025.9	170.60
Pima	**1.00**	20.10	101.90	18.30	58.03	5.00	108.13	29.83	83.20	563.50
Saheart	**1.00**	13.37	85.53	21.10	14.90	12.03	132.10	43.30	34.20	416.40
Satimage	9.70	433.20	517.70	147.90	69.70	**2.00**	496.40	239.90	3,433.7	439.30
Segment	**9.90**	99.70	98.10	58.30	36.00	18.40	248.40	79.30	232.40	494.30
Sonar	**1.78**	23.10	15.80	39.73	73.73	15.00	752.67	51.13	651.90	443.80
Tae	3.50	16.53	83.13	13.83	**2.33**	9.20	68.23	32.40	106.40	704.20
Thyroid	**4.14**	27.80	45.60	7.77	250.17	14.70	282.83	23.07	106.80	371.20
Tic–tac–toe	1.00	167.50	58.93	24.00	24.00	6.63	227.87	**0.67**	310.40	661.90
Twonorm	**1.10**	296.20	278.30	126.70	39.10	30.20	282.70	203.60	1,467.1	387.40
Vehicle	**5.00**	71.33	143.50	56.70	171.40	28.10	240.97	208.17	297.00	570.50
Wine	**4.42**	8.10	9.37	9.73	8.50	13.27	171.23	12.60	16.10	440.80
Yeast	**11.80**	173.40	590.00	33.90	55.00	23.30	113.20	82.20	1,059.5	456.00
Zoo	8.00	18.40	17.73	9.73	13.93	34.17	405.77	**0.00**	67.30	442.30
Avg. values	**4.96**	96.52	139.73	43.86	40.33	19.10	240.29	53.85	345.29	464.24
Avg. ranks	**1.27**	5.63	6.74	4.37	3.76	3.23	8.23	4.34	8.06	9.37

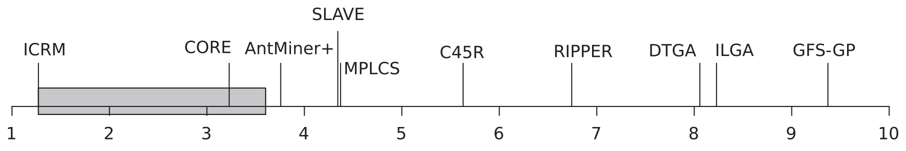


Fig. 9. Bonferroni-Dunn for the number of conditions per classifier.

Table 10
Wilcoxon test for the total number of conditions.

ICRM vs	R^+	R^-	p-value
C45R	630.0	0.0	5.82E−11
RIPPER	630.0	0.0	5.82E−11
MPLCS	630.0	0.0	5.82E−11
AntMiner+	621.0	9.0	1.921E−9
CORE	616.0	14.0	6.402E−9
ILGA	630.0	0.0	5.82E−11
SLAVE	563.0	32.0	3.218E−7
DTGA	630.0	0.0	5.82E−11
GFS-GP	630.0	0.0	5.82E−11

Fig. 9 shows the application of the Bonferroni–Dunn test to the number of conditions of the classifier for alpha = 0.01, whose critical difference is 2.3601. The algorithms right beyond the critical difference from the ICRM value are significantly worse than ICRM, which is again the best algorithm and the most interpretable regarding the total number of conditions of the rule-based classifier. Looking at the values from Table 9, there are significant differences with all the algorithms but CORE regarding the average number of conditions of the classifiers.

Table 10 shows the results of the Wilcoxon rank-sum test for the total number of conditions to compute multiple pairwise comparisons among the proposal and the other methods. The p-values reported indicate significant differences with a confidence level higher than 99% from ICRM with all the other methods.

5.5. Complexity

The complexity metric provided by Nauck [43] permits to compute an interpretability value that combines all the previous results.

Table 11 shows the average complexity values and the ranking of the algorithms. The best values are the higher complexity values, at a ranking value of 1.16, our method performs best. The Iman and Davenport statistic for the complexity distributed according to an F-distribution with $K - 1 = 9$ and $(K - 1)(N - 1) = 306$ degrees of freedom is 74.2908. The test establishes an F-distribution value = 2.4656 for a significance level of alpha = 0.01. Thus, the null hypothesis is rejected and there are significant complexity differences.

Fig. 10 shows the application of the Bonferroni–Dunn test to the complexity for alpha = 0.01, whose critical difference is 2.3601. The algorithms right beyond the critical difference from the ICRM value have significantly worse complexity values than ICRM. Observing this figure, all the algorithms but CORE have significantly worse results than ICRM regarding to the complexity value, i.e., worse interpretability. Interestingly, the results of CORE show it to be the second most interpretable algorithm in the comparison, but its accuracy results are among the worst.

Table 12 shows the results of the Wilcoxon rank-sum test for the complexity metric to compute multiple pairwise comparisons among the proposal and the other methods. The p-values reported indicate significant differences with a confidence level higher than 99% from ICRM versus all. These are the best results achieved by the ICRM method among all the metrics considered.

5.6. Execution time

The execution times for the different algorithms and data sets are shown in Table 13. These values represent the time (in s) taken by an algorithm to learn from the training data, build a classifier and perform the classification over both the training and test data. The best values are the lowest running times, at a ranking value of 2.21, C45R is the fastest algorithm followed by RIPPER. The Iman and Davenport statistic for the execution time distributed according to an F-distribution with $K - 1 = 9$ and $(K - 1)(N - 1) = 306$ degrees of freedom is 86.6229. The test establishes an F-distribution value = 2.4656 for a significance level of alpha = 0.01. Thus, the test rejects the null hypothesis and therefore it can be said that there are significant differences between the execution times of the algorithms.

Fig. 11 shows the application of the Bonferroni–Dunn test to the execution time for alpha = 0.01, whose critical difference is 2.3601. The algorithms right beyond the critical difference from the C45R value are significantly slower. Observing this

Table 11
Complexity.

Complexity	ICRM	C45R	RIPPER	MPLCS	AntMin+	CORE	ILGA	SLAVE	DTGA	GFS-GP
Appendicitis	**2.00**	0.06	0.03	0.12	0.09	0.19	0.01	0.08	0.43	0.00
Australian	**2.00**	0.06	0.03	0.12	0.09	0.19	0.01	0.08	0.02	0.00
Balance	**0.79**	0.03	0.02	0.09	0.19	0.69	0.05	0.05	0.02	0.01
Breast	**1.82**	0.04	0.02	0.02	0.26	0.20	0.01	0.00	0.10	0.00
Bupa	**1.11**	0.09	0.03	0.09	0.67	0.33	0.02	0.09	0.02	0.00
Car	0.51	0.01	0.01	0.06	0.04	0.50	0.03	**6.00**	0.01	0.01
Chess	**1.00**	0.02	0.03	0.04	0.11	0.01	0.00	0.00	0.01	0.00
Contraceptive	**1.00**	0.02	0.01	0.17	0.26	0.18	0.02	0.02	0.00	0.01
Dermatology	**0.70**	0.25	0.17	0.23	0.11	0.10	0.01	0.28	0.13	0.02
Ecoli	**0.53**	0.23	0.09	0.37	0.37	0.47	0.08	0.19	0.06	0.02
Flare	**0.81**	0.07	0.01	0.06	0.06	0.37	0.02	0.00	0.04	0.02
German	**2.00**	0.02	0.01	0.02	0.10	0.16	0.01	0.07	0.01	0.01
Glass	0.49	0.22	0.13	0.32	**0.57**	0.31	0.06	0.12	0.03	0.01
Haberman	**2.00**	0.34	0.04	0.30	1.54	0.50	0.05	0.16	0.43	0.00
Heart	**1.00**	0.07	0.06	0.07	0.10	0.15	0.01	0.07	0.04	0.00
Ionosphere	**1.00**	0.07	0.10	0.10	0.18	0.15	0.00	0.15	0.04	0.00
Iris	**1.00**	0.53	0.31	0.84	0.78	0.64	0.05	0.74	0.29	0.01
Lymphography	**0.80**	0.17	0.14	0.11	0.30	0.15	0.01	0.48	0.08	0.01
Monk-2	**1.00**	0.20	0.40	0.38	0.29	0.48	0.02	**1.00**	0.19	0.01
New-thyroid	**0.78**	0.24	0.28	0.32	0.56	0.38	0.04	0.33	0.15	0.00
Page-blocks	**0.70**	0.06	0.03	0.27	0.10	0.34	0.04	0.14	0.02	0.03
Penbased	**0.59**	0.01	0.02	0.05	0.12	0.46	0.05	0.03	0.01	0.06
Pima	**2.00**	0.10	0.02	0.11	0.03	0.40	0.02	0.07	0.02	0.00
Saheart	**2.00**	0.15	0.02	0.09	0.13	0.17	0.02	0.05	0.06	0.00
Satimage	0.72	0.02	0.01	0.05	0.10	**3.50**	0.01	0.03	0.00	0.02
Segment	**0.71**	0.07	0.07	0.12	0.19	0.38	0.03	0.09	0.03	0.01
Sonar	**1.12**	0.09	0.13	0.05	0.03	0.13	0.00	0.04	0.00	0.00
Tae	0.86	0.18	0.04	0.22	**1.29**	0.33	0.04	0.09	0.03	0.00
Thyroid	**0.72**	0.11	0.07	0.39	0.01	0.20	0.01	0.13	0.03	0.01
Tic–tac–toe	2.00	0.01	0.03	0.08	0.08	0.30	0.01	**3.00**	0.01	0.00
Twonorm	**1.82**	0.01	0.01	0.02	0.05	0.07	0.01	0.01	0.00	0.01
Vehicle	**0.80**	0.06	0.03	0.07	0.02	0.14	0.02	0.02	0.01	0.01
Wine	**0.68**	0.37	0.32	0.31	0.35	0.23	0.02	0.24	0.19	0.01
Yeast	**0.85**	0.06	0.02	0.29	0.18	0.43	0.09	0.12	0.01	0.02
Zoo	**0.88**	0.38	0.39	0.72	0.50	0.20	0.02	0.00	0.10	0.02
Avg. values	**1.11**	0.13	0.09	0.19	0.28	0.38	0.03	0.40	0.07	0.01
Avg. ranks	**1.16**	5.56	6.76	4.19	3.73	3.13	8.09	5.37	7.89	9.14

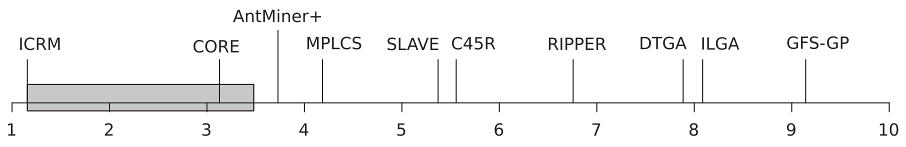


Fig. 10. Bonferroni–Dunn for complexity.

Table 12
Wilcoxon test for complexity.

ICRM vs	R^+	R^-	p-value
C45R	630.0	0.0	5.82E−11
RIPPER	630.0	0.0	5.82E−11
MPLCS	630.0	0.0	5.82E−11
AntMiner+	622.0	8.0	1.455E−9
CORE	595.0	35.0	2.508E−7
ILGA	630.0	0.0	5.82E−11
SLAVE	538.5	56.5	7.672E−6
DTGA	630.0	0.0	5.82E−11
GFS-GP	630.0	0.0	5.82E−11

figure, C45R, RIPPER, DTGA and ICRM have no statistically significant differentes in the execution time of the algorithsm. On the other hand, AntMiner+, SLAVE, MPLCS, CORE, ILGA and GFS-GP do perform significantly slower.

Table 14 shows the results of the Wilcoxon rank-sum test for the execution time to compute multiple pairwise comparisons among the proposal and the other methods. The p-values reported indicate significant differences with a confidence level higher than 99% from ICRM with MPLCS, AntMiner+, CORE, ILGA, SLAVE, and GFS-GP.

Table 13
Execution time.

Time	ICRM	C45R	RIPPER	MPLCS	AntMin+	CORE	ILGA	SLAVE	DTGA	GFS-GP
Appendicitis	2.26	1.00	1.19	18.56	55.93	35.41	56.89	32.26	**0.11**	129.11
Australian	2.26	**1.00**	1.19	18.56	55.93	35.41	56.89	32.26	**1.00**	683.78
Balance	1.15	1.07	**0.67**	19.78	26.11	12.89	9.07	26.56	2.56	412.22
Breast	**0.09**	0.44	0.44	32.33	0.33	8.11	10.44	2.33	0.44	233.44
Bupa	**0.54**	0.59	0.56	8.22	5.56	10.63	8.63	8.00	0.78	482.11
Car	**0.33**	1.04	5.07	69.74	458.85	109.96	29.26	10.81	5.44	1,222.6
Chess	**1.60**	2.33	2.89	470.00	2.00	551.56	1,402.4	67.56	6.22	2,643.9
Contraceptive	**2.94**	11.52	3.59	45.22	66.74	42.89	55.15	194.19	9.00	1,506.0
Dermatology	134.59	0.59	**0.48**	48.70	118.44	60.96	144.59	51.81	0.78	304.11
Ecoli	26.49	0.59	**0.56**	13.30	32.07	22.70	15.41	29.63	1.67	381.89
Flare	1.44	**0.81**	3.52	38.37	178.56	59.67	73.11	17.67	2.78	992.78
German	0.78	1.89	2.33	109.11	**0.67**	134.78	167.44	66.22	1.67	689.89
Glass	24.45	**0.52**	**0.52**	9.30	11.74	13.96	15.52	26.52	2.22	206.44
Haberman	**0.12**	0.30	0.33	4.89	5.48	8.00	3.22	3.70	0.22	441.44
Heart	1.64	0.63	**0.48**	13.70	33.04	13.04	27.00	12.19	0.67	358.44
Ionosphere	2.28	**0.56**	1.00	102.44	1.00	55.11	89.11	16.00	0.67	415.89
Iris	0.71	0.15	**0.11**	4.11	4.07	9.33	2.93	2.41	**0.11**	132.00
Lymphography	3.54	0.30	**0.26**	6.30	12.15	21.15	19.96	10.67	0.56	142.44
Monk-2	0.27	0.33	**0.19**	10.74	12.22	14.44	7.41	4.37	0.22	305.22
New-thyroid	1.49	0.22	**0.19**	5.41	7.52	10.81	5.70	5.11	0.33	265.67
Page-blocks	46.22	**10.26**	14.67	191.30	616.30	225.63	469.74	352.11	22.33	3,044.2
Penbased	62.62	40.78	21.11	9,343.0	**19.11**	1,011.3	4,261.0	4,976.7	62.67	4,969.7
Pima	1.36	**0.96**	1.48	19.11	115.33	24.15	35.63	22.59	3.22	916.78
Saheart	1.08	**0.67**	1.04	11.48	28.78	13.63	33.96	20.74	**0.67**	533.89
Satimage	123.52	39.67	**27.78**	7,040.4	95.67	924.44	2811.1	2,886.4	142.78	6,984.8
Segment	14.70	**3.11**	7.22	794.00	5.33	301.56	508.00	501.11	8.78	1,988.4
Sonar	98.20	**0.70**	0.96	34.63	78.44	34.11	152.74	21.96	3.44	197.00
Tae	0.65	0.33	**0.26**	4.52	2.78	5.85	4.19	6.26	0.56	172.11
Thyroid	71.61	4.37	**4.15**	629.81	9,671.8	893.44	813.78	331.07	9.78	4,988.6
Tic–tac–toe	**0.08**	0.81	0.70	20.37	151.52	27.37	56.07	6.04	5.89	1,096.9
Twonorm	**4.74**	36.33	270.67	2,757.8	7.67	913.33	3761.78	1,454.2	9.67	7,898.0
Vehicle	32.74	**2.00**	2.63	85.85	274.22	73.67	158.52	204.15	2.78	880.22
Wine	13.86	**0.26**	**0.26**	9.41	12.78	16.48	14.59	7.19	0.33	167.56
Yeast	6.08	11.67	**4.56**	232.11	18.17	83.89	97.22	234.22	16.33	1,221.1
Zoo	2.98	**0.19**	**0.19**	3.07	9.59	14.93	16.70	5.19	0.56	95.67
Avg. values	19.70	**5.09**	10.95	635.02	348.45	165.56	439.86	332.86	9.35	1,345.8
Avg. ranks	3.60	**2.21**	2.24	6.51	6.39	7.06	7.54	6.41	3.13	9.90

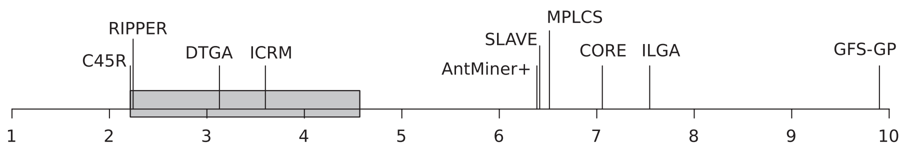


Fig. 11. Bonferroni–Dunn for the execution time.

Table 14
Wilcoxon test for the execution time.

ICRM vs	R^{+}	R^{-}	p-value
C45R	139.0	491.0	⩾0.2
RIPPER	150.0	480.0	⩾0.2
MPLCS	551.0	79.0	3.796E−5
AntMiner+	500.0	130.0	0.001843
CORE	570.0	60.0	5.686E−6
ILGA	609.0	21.0	2.602E−8
SLAVE	568.0	62.0	7.052E−6
DTGA	234.0	396.0	⩾0.2
GFS-GP	630.0	0.0	5.82E−11

5.7. Overall comparison

Table 15 summarizes the ranks obtained by the algorithms using the different metrics. The bottom rows show the average rank values and the meta rank (rank of the ranks). This table shows the great overall performance of the ICRM algorithm and

Table 15
Ranks and Meta Rank (Rank of the Ranks).

Meta Rank	ICRM	C45R	RIPPER	MPLCS	AntMin+	CORE	ILGA	SLAVE	DTGA	GFS-GP
Accuracy	4.29	3.90	4.77	**2.43**	6.71	7.74	7.20	6.64	4.13	7.19
Rules	**1.63**	6.23	7.44	3.99	3.83	2.54	7.54	4.20	7.94	9.66
C/R	**1.31**	5.11	5.29	5.39	4.94	4.57	8.63	5.50	7.16	7.10
Conditions	**1.27**	5.63	6.74	4.37	3.76	3.23	8.23	4.34	8.06	9.37
Complexity	**1.16**	5.56	6.76	4.19	3.73	3.13	8.09	5.37	7.89	9.14
Time	3.60	**2.21**	2.24	6.51	6.39	7.06	7.54	6.41	3.13	9.90
Avg. ranks	**2.21**	4.77	5.54	4.48	4.89	4.71	7.87	5.41	6.38	8.73
Meta rank	**2.00**	4.17	5.50	4.50	4.00	4.33	9.00	5.50	6.67	9.33

the bad performance of GFS-GP and ILGA. Ripper and C45R are the fastest algorithms since they are not evolutionary, but they provide classifiers with many rules and significantly more complex. MPLCS provides the most accurate classifier but its execution time is significantly slower. AntMiner + and CORE provide a relatively low number of rules but they run slower. ILGA and GFS-GP are definitely the worst algorithms from the comparison. Specifically, GFS-GP achieves the worst ranks over four of the metrics considered. Our proposal, ICRM, obtains the best rankings regarding the number of rules, the number of conditions per rule, the number of conditions per classifier, and complexity. The accuracy, the complexity of the classifier and the execution time are conflicting objectives on which ICRM has shown best overall performance.

6. Conclusion

In this paper we have proposed an interpretable classification rule mining (ICRM) algorithm, which is an interpretable and efficient rule-based evolutionary programming classification algorithm. The algorithm solves the cooperation–competition problem by dealing with the interaction among the rules during the evolutionary process. The proposal minimizes the number of rules, the number of conditions per rule, and the number of conditions of the classifier, increasing the interpretability of the solutions. The algorithm does not explore already explored search spaces. Once one rule has no better conditions to be appended, the rule is marked as unbeatable and no more searches are performed over that rule, saving computational resources. Moreover, the algorithm stops when all the individuals are marked as unbeatable, i.e., there is no possibility of improving any individual, saving further generations. The population size and the maximum number of generations are equal to the number of features. Thus, it is not necessary to configure or optimize these parameters; it is self-adapting to the problem complexity.

The experiments performed compared our algorithm with other machine learning classification methods, including crisp and fuzzy rules, rules extracted from decision trees, and an ant colony algorithm. The results show the competitive performance of our proposal in terms of predictive accuracy, obtaining significantly better results than AntMiner+, CORE, ILGA, GFS-GP, and SLAVE. ICRM obtains the best results in terms of interpretability, i.e, it minimizes the number of rules, the number of conditions per rule, and the number of conditions of the classifier. These interpretability measures can be summarized using the Nauck complexity metric. Using this measure, ICRM obtains significantly better interpretability results than all the other techniques. The algorithms whose overall performances are closest to ICRM, in accuracy vs interpretability, are C45R and MPLCS. They obtain significantly worse interpretability results but a little improvement in accuracy.

Experimental results have shown the good performance of the algorithm, but it would be honest to note its limitations. The algorithm is capable of finding comprehensible classifiers with low number of rules and conditions, while achieving competitive accuracy. However, the comprehensibility is prioritized between these conflicting objectives. The one rule per class design allows to obtain very interpretable solutions and it is useful to extract fast "big pictures" of the data. Nevertheless, the accuracy on very complex data might be lower than the obtained by other algorithms but with much more complex classifiers. There is no classifier which achieves best accuracy and comprehensibility for all data. Thus, this algorithm focuses on the interpretability of the classifier, which is very useful for knowledge discovery and decision support systems.

The algorithm runs quite fast, significantly faster than other evolutionary-based algorithms, according to the experiments. However, there exits linear performance complexity regarding to the number of attributes. Therefore, mining extremely high-dimensional data with thousands of attributes would impact the time performance. The solution to this problem would be the execution of a feature selection algorithm prior to the classification algorithm.

Acknowledgments

This work has been supported by the Regional Government of Andalusia and the Ministry of Science and Technology, projects P08-TIC-3720 and TIN-2011–22408, FEDER funds, and Ministry of Education FPU Grant AP2010–0042.

References

[1] J.S. Aguilar-Ruiz, R. Giraldez, J.C. Riquelme, Natural encoding for evolutionary supervised learning, IEEE Transactions on Evolutionary Computation 11 (2007) 466–479.
[2] D.W. Aha, D. Kibler, M.K. Albert, Instance-based learning algorithms, Machine Learning 6 (1991) 37–66.

[3] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, KEEL data-mining software tool: data set repository, integration of algorithms and experimental analysis framework, analysis framework, Journal of Multiple-Valued Logic and Soft Computing 17 (2011) 255–287.
[4] J. Alcalá-Fdez, L. Sánchez, S. García, M. del Jesus, S. Ventura, J. Garrell, J. Otero, C. Romero, J. Bacardit, V. Rivas, J. Fernández, F. Herrera, KEEL: a software tool to assess evolutionary algorithms for data mining problems, Soft Computing 13 (2009) 307–318.
[5] J. Alonso, L. Magdalena, Hilk++: an interpretability-guided fuzzy modeling methodology for learning readable and comprehensible fuzzy rule-based classifiers, Soft Computing 15 (2011) 1959–1980.
[6] S. Alonso, E. Herrera-Viedma, F. Chiclana, F. Herrera, A web based consensus support system for group decision making problems and incomplete preferences, Information Sciences 180 (2010) 4477–4495.
[7] T. Amin, I. Chikalov, M. Moshkov, B. Zielosko, Dynamic programming approach to optimization of approximate decision rules, Information Sciences 221 (2013) 403–418.
[8] R. Axelrod, The Complexity of Cooperation: Agent-based Models of Competition and Collaboration, Princeton University Press, 1997.
[9] J. Bacardit, J. Garrell, Bloat control and generalization pressure using the minimum description length principle for a Pittsburgh approach learning classifier system, in: T. Kovacs, X. Llor, K. Takadama, P. Lanzi, W. Stolzmann, S. Wilson (Eds.), Learning Classifier Systems, Lecture Notes in Computer Science, vol. 4399, Springer, 2007, pp. 59–79.
[10] J. Bacardit, N. Krasnogor, Performance and efficiency of memetic Pittsburgh learning classifier systems, Evolutionary Computation 17 (2009) 307–342.
[11] F.J. Berlanga, A.J.R. Rivas, M.J. del Jesús, F. Herrera, GP-COACH: genetic programming-based learning of compact and accurate fuzzy rule-based classification systems for high-dimensional problems, Information Sciences 180 (2010) 1183–1200.
[12] E. Bernadó-Mansilla, J.M. Garrell-Guiu, Accuracy-based learning classifier systems: models, analysis and applications to classification tasks, Evolutionary Computation 11 (2003) 209–238.
[13] U. Bhanja, S. Mahapatra, R. Roy, An evolutionary programming algorithm for survivable routing and wavelength assignment in transparent optical networks, Information Sciences 222 (2013) 634–647.
[14] L. Bull, E. Bernadó-Mansilla, J.H. Holmes (Eds.), Learning Classifier Systems in Data Mining, Studies in Computational Intelligence, vol. 125, Springer, 2008.
[15] M.V. Butz, T. Kovacs, P.L. Lanzi, S.W. Wilson, Toward a theory of generalization and learning in XCS, IEEE Transactions on Evolutionary Computation 8 (2004) 28–46.
[16] C. Campbell, Kernel methods: a survey of current techniques, Neurocomputing 48 (2002) 63–84.
[17] A. Cano, A. Zafra, S. Ventura, An ep algorithm for learning highly interpretable classifiers, in: Proceedings of the 11th International Conference on Intelligent Systems Design and Applications (ISDA), 2011, pp. 325–330.
[18] J. Cano, F. Herrera, M. Lozano, Evolutionary stratified training set selection for extracting classification rules with trade off precision-interpretability, Data and Knowledge Engineering 60 (2007) 90–108.
[19] D. Carvalho, A. Freitas, A hybrid decision tree/genetic algorithm method for data mining, Information Sciences 163 (2004) 13–35.
[20] W.J. Choi, T.S. Choi, Genetic programming-based feature transform and classification for the automatic detection of pulmonary nodules on computed tomography images, Information Sciences 212 (2012) 57–78.
[21] M. Cintra, M. Monard, H. Camargo, On rule learning methods: a comparative analysis of classic and fuzzy approaches, Studies in Fuzziness and Soft Computing 291 (2013) 89–104.
[22] W. Cohen, Fast effective rule induction, in: Proceedings of the 12th International Conference on Machine Learning, 1995, pp. 1–10.
[23] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Machine Learning Research 7 (2006) 1–30.
[24] O. Dunn, Multiple comparisons among means, Journal of the American Statistical Association 56 (1961) 52–64.
[25] P. Espejo, S. Ventura, F. Herrera, A survey on the application of genetic programming to classification, IEEE Transactions on Systems, Man, and Cybernetics, Part C 40 (2010) 121–144.
[26] D. Fisch, B. Kühbeck, B. Sick, S.J. Ovaska, So near and yet so far: new insight into properties of some well-known classifier paradigms, Information Sciences 180 (2010) 3381–3401.
[27] S. García, A. Fernández, J. Luengo, F. Herrera, A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability, Soft Computing 13 (2009) 959–977.
[28] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power, Information Sciences 180 (2010) 2044–2064.
[29] S. García, D. Molina, M. Lozano, F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study, Journal of Heuristics 15 (2009) 617–644.
[30] A. González, R. Perez, Selection of relevant features in a fuzzy genetic learning algorithm, IEEE Transactions on Systems and Man and and Cybernetics and Part B: Cybernetics 31 (2001) 417–425.
[31] D.P. Greene, S.F. Smith, Competition-based induction of decision models from examples, Machine Learning 13 (1994) 229–257.
[32] S. Guan, F. Zhu, An incremental approach to genetic-algorithms-based classification, IEEE Transactions on Systems and Man and and Cybernetics and Part B 35 (2005) 227–239.
[33] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Chapter 4: Context-Free Grammars, Addison-Wesley, 2006.
[34] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, B. Baesens, An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models, Decision Support Systems 51 (2011) 141–154.
[35] H. Ishibuchi, Y. Kaisho, Y. Nojima, Complexity, interpretability and explanation capability of fuzzy rule-based classifiers, in: Proceedings of the IEEE International Conference on Fuzzy Systems, 2009, pp. 1730–1735.
[36] U. Johansson, C. Sönstrd, T. Löfström, H. Boström, Obtaining accurate and comprehensible classifiers using oracle coaching, Intelligent Data Analysis 16 (2012) 247–263.
[37] S.A. Kazarlis, S.E. Papadakis, J.B. Theocharis, V. Petridis, Microgenetic algorithms as generalized hill-climbing operators for GA optimization, IEEE Transactions on Evolutionary Computation 5 (2001) 204–217.
[38] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: Proceedings of the 14th International Joint Conference on Artificial Intelligence, vol. 2, 1995, pp. 1137–1143.
[39] T. Kovacs, Genetics-based machine learning, in: G. Rozenberg, T. Bäck, J. Kok (Eds.), Handbook of Natural Computing: Theory, Experiments, and Applications, Springer-Verlag, 2011.
[40] P.L. Lanzi, Learning classifier systems: then and now, Evolutionary Intelligence 1 (2008) 63–82.
[41] M. Li, Z.W. Z, A hybrid coevolutionary algorithm for designing fuzzy classifiers, Information Sciences 179 (2009) 1970–1983.
[42] D. Martens, B. Baesens, T.V. Gestel, J. Vanthienen, Comprehensible credit scoring models using rule extraction from support vector machines, European Journal of Operational Research 183 (2007) 1466–1476.
[43] D.D. Nauck, Measuring interpretability in rule-based classification systems, in: Proceedings of IEEE International Conference on Fuzzy Systems, 2002, pp. 196–201.
[44] C. Nguyen, W. Pedrycz, T. Duong, T. Tran, A genetic design of linguistic terms for fuzzy rule based classifiers, International Journal of Approximate Reasoning 54 (2013) 1–21.
[45] A. Palacios, L. Sánchez, I. Couso, Extending a simple genetic cooperative-competitive learning fuzzy classifier to low quality datasets, Evolutionary Intelligence 2 (2009) 73–84.
[46] M. Paliwal, U. Kumar, Neural networks and statistical techniques: a review of applications, Expert Systems with Applications 36 (2009) 2–17.

[47] R. Parpinelli, H. Lopes, A. Freitas, Data mining with an ant colony optimization algorithm, IEEE Transactions on Evolutionary Computation 6 (2002) 321–332.
[48] J. Quinlan, C4.5: Programs for Machine Learning, 1993.
[49] D. Richards, Two decades of ripple down rules research, Knowledge Engineering Review 24 (2009) 159–184.
[50] J. Rissanen, Minimum description length principle, in: Encyclopedia of Machine Learning, 2010, pp. 666–668.
[51] R.L. Rivest, Learning decision lists, Machine Learning 2 (1987) 229–246.
[52] L. Sánchez, I. Couso, J. Corrales, Combining gp operators with sa search to evolve fuzzy rule based classifiers, Information Sciences 136 (2001) 175–192.
[53] D. Stavrakoudis, G. Galidaki, I. Gitas, J. Theocharis, Reducing the complexity of genetic fuzzy classifiers in highly-dimensional classification problems, International Journal of Computational Intelligence Systems 5 (2012) 254–275.
[54] K. Tan, Q. Yu, J. Ang, A coevolutionary algorithm for rules discovery in data mining, International Journal of Systems Science 37 (2006) 835–864.
[55] S. Tsumoto, Mining diagnostic rules from clinical databases using rough sets and medical diagnostic model, Information Sciences 162 (2004) 65–80.
[56] S. Ventura, C. Romero, A. Zafra, J. Delgado, C. Hervás, JCLEC: a Java framework for evolutionary computation, Soft Computing 12 (2007) 381–392.
[57] W. Verbeke, D. Martens, C. Mues, B. Baesens, Building comprehensible customer churn prediction models with advanced rule induction techniques, Expert Systems with Applications 38 (2011) 2354–2364.
[58] T. Wiens, B. Dale, M. Boyce, G. Kershaw, Three way k-fold cross-validation of resource selection functions, Ecological Modelling 212 (2008) 244–255.
[59] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics Bulletin 1 (1945) 80–83.
[60] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2005.
[61] M.L. Wong, K.S. Leung, Data Mining Using Grammar Based Genetic Programming and Applications, Kluwer Academic Publisher, 2000.
[62] N. Xie, Y. Liu, Review of decision trees, in: Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), vol. 5, 2010, pp. 105–109.
[63] J. Yang, H. Xu, P. Jia, Effective search for pittsburgh learning classifier systems via estimation of distribution algorithms, Information Sciences 198 (2012) 100–117.
[64] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Transactions on Evolutionary Computation 3 (1999) 82–102.
[65] X. Yu, M. Gen, Introduction to Evolutionary Algorithms, Springer, 2010.
[66] A. Zafra, S. Ventura, G3P-MI: a genetic programming algorithm for multiple instance learning, Information Sciences 180 (2010) 4496–4513.

A Parallel Genetic Programming Algorithm for Classification

Alberto Cano, Amelia Zafra, and Sebastián Ventura

Department of Computing and Numerical Analysis, University of Córdoba
14071 Córdoba, Spain
{i52caroa,azafra,sventura}@uco.es

Abstract. In this paper a Grammar Guided Genetic Programming-based method for the learning of rule-based classification systems is proposed. The method learns disjunctive normal form rules generated by means of a context-free grammar. The individual constitutes a rule based decision list that represents the full classifier. To overcome the problem of computational time of this system, it parallelizes the evaluation phase reducing significantly the computation time. Moreover, different operator genetics are designed to maintain the diversity of the population and get a compact set of rules. The results obtained have been validated by the use of non-parametric statistical tests, showing a good performance in terms of accuracy and interpretability.

Keywords: Genetic Programming, Classification.

1 Introduction

The general idea of discovering knowledge in large amounts of data is both appealing and intuitive, but technically it is significantly challenging and difficult, especially in the fields where really huge amounts of relational data have been collected over last decades. In this paper we focus on classification which is a well-known task in data mining.

The classification task has been overcomed with numerous computer techniques (rule learning, instance based learning, neural networks, support vector machines, statistical classifiers and so on). These include crisp rule learning [10,6], decision trees [15], evolutionary algorithms [3,4,13,20] and specifically Genetic Programming (GP) algorithms [7,11,18,19]. One of the best advantage of rule-based system is that they provide interpretable solutions to the user. When considering a rule-based learning system, the different genetic learning methods follow two approaches in order to encode rules within a population of individuals. The first one represents an individual as a rule set, this proposal is known as Pittsburgh approach [17]. The second one represents an individual as a single rule, and the whole rule set is provided by combining several individuals in a population (rule cooperation) or via different evolutionary runs (rule competition). In turn, within the individual as a single rule approach, there are three generic proposals: Michigan, Iterative Rule Learning and Genetic Cooperative-Competitive Learning).

The main advantage of the Pittsburgh model compared to other approaches is that it allows to address the cooperation-competition problem, dealing the interaction among rules in the evolutionary process. However, its main problem is controlling the number of rules of the individuals, as the total number of rules in the population can grow quite, increasing the computational cost and becoming unmanageable problems. On the other hand, the other approches provide good results but are inefficient methods and have addressed the cooperation-competition problem by not dealing the interaction between rules in the evolutionary process.

In this paper we propose a Grammar Guided GP (G3P) based algorithm that learns disjunctive normal form (DNF) rules generated by means of a context-free grammar, coded as one rule base decision list [16] per individual (Pittsburgh approach). This provides easily interpretable and understandable rules, and it also considers the interaction among the rules in the evolutionary process. The genetic operators are designed to work on two levels. On the one hand, it allows the optimization of particular rules by combining the rules to obtain the bests disjunction and conjunction of attribute-value comparisons. On the other hand, it address the problem of cooperation-competition, considering the interaction that occurs among the rules when these are combined to form the final classifiers.

Evolving full classifiers allows us to evaluate the relationship among rules but introduces greater complexity and computation time. To solve this problem and reduce the computation time, the evaluation phase for fitness computation is parallelized using the GPU or multiple CPU threads. Furthermore, the problem of controlling the number of rules is solved by setting a parameter with the maximum number of rules per class. This parameter allows the user to decide to obtain simpler or more complex classificators by limiting the number of rules.

An experimental study involving 18 datasets and 11 well-known classification algorithms shows that the algorithm obtains accurate and comprehensible rules. Non-parametric statistical methods have been used to compare and analyze the accuracy of the experimental results. They show the good performance in terms of accuracy of our approach compared to other traditional methods analyzed. Moreover, the suitability of some components such as the use of specific genetic operators and the use of full classifier take advantages with respect to the rest of GP and G3P-based methods considered in the study.

The remainder of this paper is organized as follows. Section 2 presents the proposed GP classification algorithm and discusses the genetic operators, the fitness function and the generational model. Section 3 describes the experimental study. In Section 4, the results will be announced and finally the last section presents the final remarks of our investigation and outlines future research work.

2 Algorithm

This section details the algorithm proposed. It describes the encoding of the individual's genotype, the genetic operators, the fitness function, the initialization criterion and the generational model.

2.1 Individual Representation

One of the main advantages of GP is the flexibility to represent the solutions. Thus, GP can be employed to construct classifiers using different kinds of representations, e.g. decision trees, classification rules, discriminant functions, and many more [9]. In our case, the individuals in the Pittsburgh GP algorithm are classifiers where each individual is a set of classification rules generated by means of a context-free grammar and whose expression tree is composed by terminal and non-terminal nodes. A classifier can be expressed as a set of IF-antecedent-THEN-consequent rules in a decision rules list. The antecedent of the rule represents a disjunction and conjunction of attribute-value comparisons and the rule consequent specifies the class to be predicted for an instance that satisfies all the conditions of the rule antecedent. The terminals set consists of the attribute names and attribute values of the dataset being mined, logical operators (AND, OR, NOT), relational operators ($<$, $=$, $<>$, $>$) and the interval range operator (IN). We must ensure that the classifier contains at least one rule for each class.

2.2 Genetic Operators

This subsection describes the genetic crossover and mutation operators which modify the genotype of the individuals throughout the evolutionary process.

Crossover Operator

As mentioned, an individual represents a classifier as a set of rules. Taking advantage of this representation, the crossover operator is designed to optimize both the rules and the interaction among them in the classifier. So on the one hand, the crossover applied on specific rules operates on two rules from the individual and produces two new rules. Two random compatible nodes are selected from within each rule and then the resultant sub-trees are swapped, generating two child rules. These crossed rules become the decision list of a new individual. On the other hand, the crossover applied on classifers acts over the individuals swapping their rules. Given two parent individuals, two crossing points are chosen (one by parent) so the rules are swapped from those points, building two new individuals different from their parents. Therefore, selected crossing points will ensure that at least one rule of a classifier will cross with the classification rules from the other, i.e., it is not allowed to swap all the rules of a classifier.

Mutation Operator

The mutation operator can also be applied to the rules and the individuals. The rules mutation operates either on a function node or a terminal node. It randomly selects a node in a sub-tree and replaces it with a new randomly created sub-tree. The mutation of an individual is determined by the random elimination of a rule of the rule set with a probability degree.

2.3 Fitness Function

The algorithm has two fitness functions. The first one evaluates the rules of each individual independently. The second one evaluates the individuals checking the success rates of the classifiers over the training set. It is necessary to evaluate the rules of each individual first and then evaluate the classifiers success rates.

Fitness Function Applied on Particular Rules

The fitness function we use on particular rules is the proposed by Bojarczuk et al. [11]. Specifically, each rule is evaluated over each instance and each class. This obtains the results of the quality of the predictions of the rules for each class. Thus, the consequent of a rule is reassigned to the class that has produced better results. The rules fitness function combines two indicators that are commonplace in the domain, namely the sensitivity (Se) and the specifity (Sp), which employ the true positive (t_p), false positive (f_p), true negative (t_n) and false negative (f_n) values from the match of the class of the instance and the predicted class.

$$Se = \frac{t_p}{t_p + f_n} \qquad Sp = \frac{t_n}{t_n + f_p} \qquad ruleFitness = Se * Sp \tag{1}$$

Fitness Function Applied on Rule Set (The Classifier)

The classifiers fitness function is performed once the best consequent for each rule is calculated. The fitness of a classifier is simply the success rate of the classifier over the training set. Each instance is submitted to the decision list to find the first rule that covers the instance. If the consequent of the rule matches the class of the instance it is a hit, otherwise it is a fail.

The activation process of the rules defines in which order the rules are evaluated to determine which rule classifies each instance. The activation method employed is a decision list. Thus, an instance is classified by the first rule of the classifier that covers the instance and whose order is defined in the individual's genotype. If no rule covers the instance, it is classified using the default class (most frequent class or defined by the user).

Parallel Evaluation Phase

Many studies have proved the evaluation phase is by far the most expensive [5] since it requires evaluating each rule over each instance. As mentioned, a disadvantage of Pittsburgh approaches is their high computational complexity. In fact, if you analyze our proposal, the total number of rules is the addition of the number of rules for each individual. This number can be large and at least the total number of rules is the product of the number of classes and the population size. An evolutionary system that preforms crossover, mutation, evaluation and selection on so many rules is really expensive.

To solve this problem, numerous studies have parallelized the evaluation phase of these algorithms. Among the references there are two main ways to perform the parallelization. One is the use of different threads and the second one in more recent works is the use of GPUs [14]. The most efficient approaches conclude they can speed up the evaluation phase more than 800 times [5]. Therefore, we will take advantage of its parallelization capability to solve the high computational cost problem of the evaluation phase, specifically in Pittsburgh approaches, by using the GPU.

Our system divides the evaluation phase into two functions: rules fitness function and individuals fitness function. The rules evaluation can be performed completely parallel using GPUs, but also the evaluation of individuals can be parallelized. Each individual can be tested as a rule over the training set independently. Therefore, our model will take advantage of both parallel fitness functions to reduce the computational cost. The full description of the parallel model would exceed the scope and the number of pages of this paper but it is detailed in [5].

2.4 Initialization

The initialization of the algorithm with a genetically diverse population is crucial for a successful outcome. The problem of the fitness function we use to evaluate the rules is that a minority class can ignored. Therefore, we must ensure that individuals contains at least one rule for each class. One way to do this, once the individuals are created, is to complete with new rules of the classes that have not yet been covered by at least one rule. As it might be very expensive to get at least one rule for each and every one of the classes, the process of completing the individual is performed up to a number of times depending on the number of classes. This way, we try most of the classes to be represented in the classifier. Finally, among all the rules of all individuals we decide to keep the best rule for each class that will help us later to fill the classifiers with the rules of the classes that could be missing.

2.5 Generational Model

In this section we describe the generational model represented in Fig. 1. The left box represents the initialization of individuals described before. Once the population is initialized, the algorithm iteratively proceeds to apply genetic operators crossover and mutation described in the section 2.2. Specifically, the algorithm performs the individuals crossover swapping subsets of rules. The individuals can mutate eliminating some of its rules. The rules within each individual are crossed together and mutated, obtaining new classification rules. These new rules must be evaluated to get their consequents and fitness values.

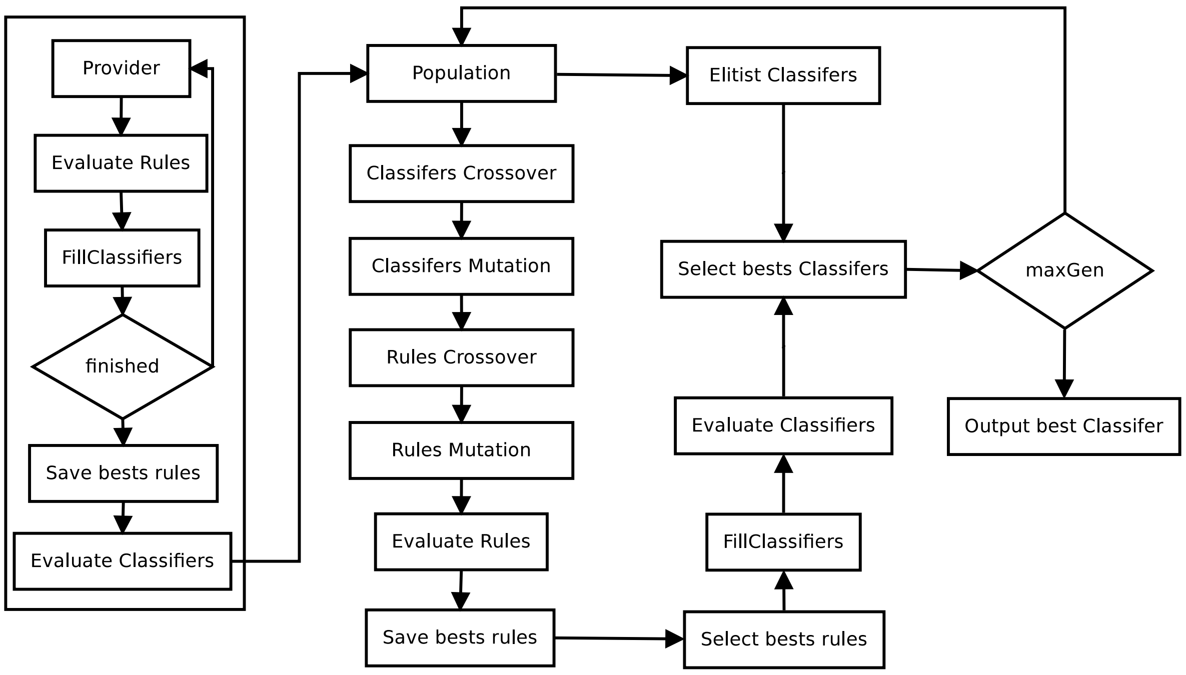


Fig. 1. Algorithm flowchart

To ensure the survival of the best rules, the algorithm checks in each generation and for each class if any new rule is better than the one stored for that class, if so the rule is replaced by the new one. As the crossover operator may have created individuals that exceed the maximum number of rules allowed, we can simplify by selecting the best rules subset.

The individual must be completed by rules from uncovered classes, taking the best rules from the pool to cover them. Once completed, each individual must be evaluated to get its fitness using the training data. We employ elitism to keep a subset of the best individuals in each generation to ensure the survival of the fittest individuals. The algorithm finishes when it has found an individual that correctly classifies all the training instances or when the algorithm has iterated a certain number of generations.

3 Experimental Study

This section describes the details of the experiments, discuss the application domains, the algorithms used and the settings of the tests.

The experiments performed compare the results of 11 different classification algorithms using 18 datasets. These algorithms are available on the JCLEC [21] website and KEEL [2] website. The datasets employed have been selected from the KEEL repository website[1]. These datasets are very varied considering different degrees of complexity, number of classes, number of features and number of instances. Thus, the number of classes ranges up to 10, the number of features ranges from 4 to 60 and the number of instances ranges from 101 to 58000.

To properly evaluate the performance of the algorithm proposed it is considered to make a comparative study with some evolutionary crisp rule learning algorithms for classification (De Falco et al. [7], Bojarczuk et al. [11], Tan et al. [18,19], MPLCS [3], ILGA [13], CORE [20] and UCS [4]), two rule learning algorithms widely extended (PART [10] y RIPPER [6]) and a classic decision tree algorithm (C4.5 [15]). For each algorithm, the values of the parameters to be configured by the user were set to the default values provided by the authors. The population size and the number of generations for our proposal are both set to 200, i.e. the algorithm deals with 200 candidate full classifiers, and the maximum number of rules per class is set to 3. The results are validated using 10-fold cross validation and 10 runs with different seeds for stochastic methods. The results provided are the average of the 100 executions. A CPU time comparison analysis would exceed the number of pages of this paper and the execution time of the proposal is definitely higher than the other algorithms.

The experiments were executed in an Intel i7 quadcore machine with 12 GB DDR3-1600 and two NVIDIA GTX 480 GPUs, which are 3 billion transistors GPUs with 480 cores and 1.5 GB GDDR5. The operating system was Ubuntu Linux 10.10 64 bits, the GPU computing software was NVIDIA CUDA 3.1 and the compiler GCC 4.4.4.

4 Results

In this section we provide the results of the experiments and discuss the performance of the algorithms over the datasets. Table 1 shows the average results of the predictive accuracy obtained running each algorithm ten times in each of the datasets tests folds. The last but one row shows the average accuracy over all the datasets.

Table 1. Experiments results: predictive accuracy (%)

Dataset	Proposal	Falco	Bojarczuk	Tan [18]	Tan [19]	MPLCS	ILGA	CORE	UCS	C4.5	PART	RIPPER
Zoo	95.90%	61.73%	86.31%	93.43%	92.75%	95.50%	84.67%	94.58%	**96.50%**	92.80%	86.13%	93.41%
Iris	95.82%	76.20%	84.40%	89.67%	75.34%	96.00%	93.33%	94.67%	92.00%	**96.67**%	33.33%	94.00%
Hepatitis	**88.51%**	68.85%	74.44%	73.50%	83.43%	85.15%	77.92%	83.43%	80.74%	85.66%	84.17%	79.09%
Wine	94.58%	63.72%	79.33%	75.81%	66.24%	91.54%	89.28%	**95.49%**	91.57%	94.90%	63.98%	93.79%
Sonar	75.09%	61.90%	66.31%	60.57%	59.05%	76.81%	71.57%	53.38%	**77.83%**	70.54%	60.53%	72.45%
Glass	**69.39%**	31.79%	39.98%	36.88%	48.85%	65.88%	52.40%	50.97%	66.07%	68.86%	46.50%	63.47%
New-thyroid	**94.40%**	69.18%	75.54%	73.65%	78.66%	91.65%	90.71%	91.21%	93.55%	93.52%	77.22%	93.05%
Heart	81.85%	66.78%	74.37%	72.55%	76.30%	**83.70%**	64.44%	71.11%	80.37%	78.14%	57.77%	76.29%
Dermatology	94.96%	45.61%	73.59%	77.29%	76.56%	95.53%	60.81%	43.86%	**96.36%**	94.35%	73.12%	94.42%
Haberman	**73.82%**	65.92%	68.43%	52.70%	70.18%	72.17%	71.89%	70.88%	72.18%	71.19%	73.53%	46.72%
Ecoli	78.92%	51.72%	57.63%	50.48%	65.19%	**80.67%**	63.10%	65.78%	79.49%	77.37%	44.67%	72.63%
Australian	85.41%	72.42%	84.29%	75.77%	78.41%	**86.96%**	85.07%	83.33%	86.09%	85.21%	60.72%	81.44%
Pima	75.01%	69.82%	67.59%	62.98%	66.68%	74.60%	73.19%	72.28%	**76.04%**	72.53%	65.11%	69.53%
Vehicle	70.59%	31.11%	41.67%	36.04%	41.52%	71.15%	57.24%	40.05%	**72.45%**	66.66%	37.85%	70.44%
Contraceptive	**55.61%**	40.75%	42.68%	40.37%	44.00%	55.47%	43.59%	45.01%	49.49%	51.19%	42.90%	50.78%
Thyroid	99.22%	68.05%	51.39%	52.92%	92.43%	94.72%	94.10%	68.00%	96.99%	**99.56%**	92.58%	99.37%
Penbased	83.32%	25.54%	40.29%	35.76%	44.90%	91.80%	53.25%	15.69%	14.28%	94.89%	15.86%	**96.15%**
Shuttle	**99.97%**	61.16%	75.51%	63.55%	89.51%	99.60%	93.67%	91.60%	99.62%	99.96%	99.60%	99.96%
Average (%)	**84.02%**	57.34%	65.76%	62.44%	69.44%	83.82%	73.34%	68.40%	78.98%	83.00%	61.97%	80.38%
Ranking	**2.22**	10.56	8.72	9.72	8.08	3.03	7.00	7.25	3.56	3.58	9.19	5.14

Analyzing the results of the table notice that the algorithm proposed obtains better average accuracy results and ranking. Its accuracy is higher than most algorithms and is very close and slightly higher than the algorithms MPLCS, UCS and C4.5. In those datasets in which our algorithm does not achieve the best results, its accuracy is usually quite competitive. Moreover, the algorithm does not stand out negatively in any dataset.

In order to analyze the results and discover the existence of significant differences between the algorithms, a series of statistical tests [8,12] were carried out. We use the Iman and Davenport test to rank the k algorithms over the N datasets. The average rank according to the F-distribution throughout all the datasets is shown in the last row of the table. Notice that the best global position, the lowest ranking value, corresponds to the obtained by our proposal.

The Iman and Davenport statistic for average accuracy distributed according to F-distribution with $k-1 = 11$ and $(k-1)(N-1) = 187$ degrees of fredom is 30.1460. This value does not belong to the critical interval $[0, F_{0.01,11,187} = 2.3439]$ for $p = 0.01$. Thus, we reject the null-hypothesis that all the algorithms perform equally well. In order to analyse if there are significant differences among the algorithms, we use the Bonferroni-Dunn test to reveal the difference in performance using a critical difference (CD) value for $p = 0.01$ equal to 3.9865.

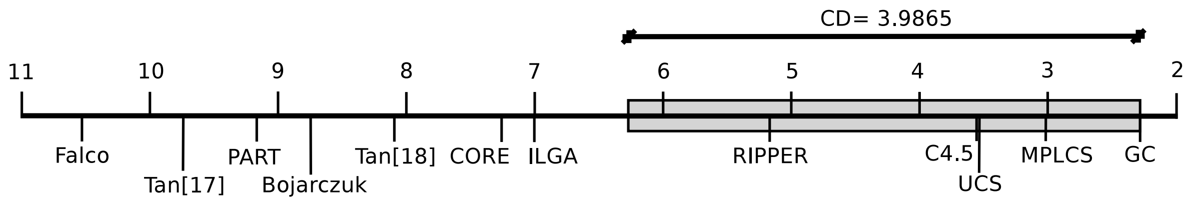


Fig. 2. Bonferroni-Dunn test. The noncritical range is shown shaded.

The results indicate that a significance level of $p = 0.01$ (i.e., with a probability of 99%), there are significant differences between our algorithm and De Falco et al., Bojarczuk et al., Tan et al. [18,19], PART, IGLA and CORE algorithms being our algorithm statistically better. These differences are shown in Fig. 2. The figure represents the algorithm's ranking and the critical distance interval. Regarding to the other algorithms, the test does not indicate significant differences. However, our proposal obtains the lowest ranking value, indicating that considering all datasets, it obtains better results in a greater number of them than other proposals.

5 Conclusions

In this paper we have proposed a G3P algorithm that optimizes the class prediction accuracy of a decision list as a set of interpretable DNF rules. The encoding of the individuals have allowed us to design specific crossover and mutation operators that consider the interaction among the rules in the evolutionary process. The algorithm overcomes the high computational complexity of Pittsburgh approaches by the parallelization of the evaluation phase. Nevertheless, in spite of this improvement, the complexity of the algorithm is still high. The experiments carried out compare the results of our proposal with other classic algorithms for solving classification problems considering 11 algorithms and 18 datasets. A statistical study on results validates the model showing that it provides the most accurate classifiers since obtaining the lowest ranking in the general comparative.

The algorithm design is complex and the execution time is high even using GPUs for the evaluation of the individuals. The individual representation is as a set of classification rules and it is necessary to evaluate both rules and classifiers every generation. Therefore, the algorithm performs slow regarding to the other algorithms from the experimental. Nevertheless, the idea of this work was to experiment a highly complex G3P Pittsburgh algorithm for classification accelerated using GPUs, since GPUs have been successfully used in Michigan approaches to speed up the evaluation phase [5]. Therefore, there is no a sequential implementation of the algorithm to compare with, since it would require excessive CPU time.

Currently, databases are moving toward sparse data, unbalanced, multiple labels, instances. It would be interesting to check the performance of the algorithm in these contexts and to improve the efficiency of the algorithm to perform faster.

Additional information of the paper such as the algorithm's code, the experimental setup and the datasets are published and available on the website: `http://www.uco.es/grupos/kdis/kdiswiki/HAIS11`

Acknowledgments. This work has been financed in part by the TIN2008-06681-C06-03 project of the Spanish Inter-Ministerial Commission of Science and Technology (CICYT), the P08-TIC-3720 project of the Andalusian Science and Technology Department and FEDER funds.

References

1. Alcalá-Fdez, J., Fernandez, A., Luengo, J., Derrac, J., García, S., Sánchez, L., Herrera, F.: KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework. Analysis Framework. Journal of Multiple-Valued Logic and Soft Computing 17, 255–287 (2011)
2. Alcalá-Fdez, J., Sánchez, L., García, S., del Jesus, M., Ventura, S., Garrell, J., Otero, J., Romero, C., Bacardit, J., Rivas, V., Fernández, J., Herrera, F.: KEEL: A Software Tool to Assess Evolutionary Algorithms for Data Mining Problems. Soft Computing - A Fusion of Foundations, Methodologies and Applications 13, 307–318 (2009)
3. Bacardit, J., Krasnogor, N.: Performance and efficiency of memetic pittsburgh learning classifier systems. Evolutionary Computation 17(3), 307–342 (2009)
4. Bernadó-Mansilla, E., Garrell, J.M.: Accuracy-based learning classifier systems: Models and analysis and applications to classification tasks. Evolutionary Computation 11(3), 209–238 (2003)
5. Cano, A., Zafra, A., Ventura, S.: Solving classification problems using genetic programming algorithms on gPUs. In: Corchado, E., Graña Romay, M., Manhaes Savio, A. (eds.) HAIS 2010. LNCS, vol. 6077, pp. 17–26. Springer, Heidelberg (2010)
6. Cohen, W.W.: Fast effective rule induction. In: Proceedings of the 12th International Conference on Machine Learning, pp. 115–123. Morgan Kaufmann, San Francisco (1995)
7. De Falco, I., Della Cioppa, A., Tarantino, E.: Discovering interesting classification rules with genetic programming. Applied Soft Comput. 1(4), 257–269 (2001)

8. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. 7, 1–30 (2006)
9. Espejo, P.G., Ventura, S., Herrera, F.: A Survey on the Application of Genetic Programming to Classification. IEEE Transactions on Systems, Man, and Cybernetics, Part C 40(2), 121–144 (2010)
10. Frank, E., Witten, I.H.: Generating accurate rule sets without global optimization. In: Proceedings of the 15th International Conference on Machine Learning, pp. 144–151 (1998)
11. Freitas, A.A.: Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer-Verlag New York, Inc., Secaucus (2002)
12. García, S., Molina, D., Lozano, M., Herrera, F.: A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the cec'2005 special session on real parameter optimization. Journal of Heuristics 15, 617–644 (2009)
13. Guan, S.U., Zhu, F.: An incremental approach to genetic-algorithms-based classification. IEEE Transactions on Systems and Man and Cybernetics and Part B 35(2), 227–239 (2005)
14. Harding, S.: Genetic programming on graphics processing units bibliography, `http://www.gpgpgpu.com/`
15. Quinlan, J.: C4.5: Programs for Machine Learning (1993)
16. Rivest, R.L.: Learning decision lists. Mach. Learn. 2, 229–246 (1987)
17. Smith, S.F.: A Learning System Based on Genetic Adaptive Algorithms. Phd thesis, University of Pittsburgh (1980)
18. Tan, K.C., Tay, A., Lee, T.H., Heng, C.M.: Mining multiple comprehensible classification rules using genetic programming. In: Proceedings of the Evolutionary Computation CEC 2002, pp. 1302–1307. IEEE Computer Society, Washington, DC, USA (2002)
19. Tan, K.C., Yu, Q., Heng, C.M., Lee, T.H.: Evolutionary computing for knowledge discovery in medical diagnosis. Artificial Intelligence in Medicine 27(2), 129–154 (2003)
20. Tan, K.C., Yu, Q., Ang, J.H.: A coevolutionary algorithm for rules discovery in data mining. International Journal of Systems Science 37(12), 835–864 (2006)
21. Ventura, S., Romero, C., Zafra, A., Delgado, J.A., Hervás, C.: JCLEC: a Java framework for evolutionary computation. Soft. Comput. 12, 381–392 (2007)

Weighted Data Gravitation Classification for Standard and Imbalanced Data

Alberto Cano, *Member, IEEE*, Amelia Zafra, *Member, IEEE*, and Sebastián Ventura, *Senior Member, IEEE*

***Abstract*—Gravitation is a fundamental interaction whose concept and effects applied to data classification become a novel data classification technique. The simple principle of data gravitation classification (DGC) is to classify data samples by comparing the gravitation between different classes. However, the calculation of gravitation is not a trivial problem due to the different relevance of data attributes for distance computation, the presence of noisy or irrelevant attributes, and the class imbalance problem. This paper presents a gravitation-based classification algorithm which improves previous gravitation models and overcomes some of their issues. The proposed algorithm, called DGC+, employs a matrix of weights to describe the importance of each attribute in the classification of each class, which is used to weight the distance between data samples. It improves the classification performance by considering both global and local data information, especially in decision boundaries. The proposal is evaluated and compared to other well-known instance-based classification techniques, on 35 standard and 44 imbalanced data sets. The results obtained from these experiments show the great performance of the proposed gravitation model, and they are validated using several nonparametric statistical tests.**

***Index Terms*—Classification, covariance matrix adaptation evolution strategy (CMA-ES), data gravitation, evolutionary strategies, imbalanced data.**

I. Introduction

Supervised learning is one of the most fundamental tasks in machine learning. A supervised learning algorithm analyzes a set of training examples and produces an inferred function to predict the correct output for any other examples. Classification is a common task in supervised machine learning which aims at predicting the correct class for a given example. Classification has been successfully implemented using many different paradigms and techniques, such as artificial neural networks [1], support vector machines (SVMs) [2], instance-based learning methods [3], or nature-inspired techniques such as genetic programming [4].

The nearest neighbor (NN) algorithm [5] is an instance-based method which might be the simplest classification algorithm. Its classification principle is to classify a new sample with the class of the closest training sample. The extended version of NN to k neighbors (KNN) and their derivatives are indeed one of the most influential data mining techniques, and they have been shown to perform well in many domains [6]. However, the main problem with these methods is that they severely deteriorate with noisy data or high dimensionality: their performance becomes very slow, and their accuracy tends to deteriorate as the dimensionality increases, especially when classes are nonseparable or they overlap [7].

In recent years, new instance-based methods based on data gravitation classification (DGC) have been proposed to solve the aforementioned problems of the NN classifiers [8]–[10]. DGC models are inspired by Newton's law of universal gravitation and simulate the accumulative attractive force between data samples to perform the classification. These gravitation-based classification methods extend the NN concept to the law of gravitation among the objects in the physical world. The basic principle of DGC is to classify data samples by comparing the data gravitation among the training samples for the different data classes, whereas KNNs vote for the k training samples that are the closest in the feature space.

This paper presents a DGC algorithm (DGC+) that compares the gravitational field for the different data classes to predict the class with the highest magnitude. The proposal improves previous data gravitation algorithms by learning the optimal weights of the attributes for each class and solves some of their issues such as nominal attributes handling, imbalanced data performance, and noisy data filtering. The weights of the attributes in the classification of each class are learned by means of the covariance matrix adaptation evolution strategy (CMA-ES) [11] algorithm, which is a well-known, robust, and scalable global stochastic optimizer for difficult nonlinear and nonconvex continuous domain objective functions [12]. The proposal improves accuracy results by considering both global and local data information, especially in decision boundaries.

The experiments have been carried out on 35 standard and 44 imbalanced data sets collected from the KEEL [13] and UCI [14] repositories. The algorithms compared for both standard and imbalanced classifications have been selected from the KEEL [15] and WEKA [16] software tools. The experiments consider different problem domains, number of instances, attributes, and classes. The algorithms from the experiments include some of the most relevant instance-based and imbalanced classification techniques presented up to now. The results reported show the competitive performance of the proposal, obtaining significantly better results in terms of predictive accuracy, Cohen's kappa rate [17], [18], and area under the curve (AUC) [19], [20]. The experimental study includes a statistical

Manuscript received May 24, 2012; revised September 5, 2012; accepted October 25, 2012. Date of publication January 4, 2013; date of current version November 18, 2013. This work was supported by the Regional Government of Andalusia and the Ministry of Science and Technology, projects P08-TIC-3720 and TIN-2011-22408, FEDER funds, and Ministry of Education FPU grant AP2010-0042. This paper was recommended by Editor J. Basak.

The authors are with the Department of Computer Science and Numerical Analysis, University of Cordoba, 14071 Cordoba, Spain (e-mail: acano@uco.es; azafra@uco.es; sventura@uco.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCB.2012.2227470

analysis based on the Bonferroni–Dunn [21] and Wilcoxon [22] nonparametric tests [23], [24] in order to evaluate whether there are significant differences in the results of the algorithms.

The remainder of this paper is organized as follows. The next section presents some definitions and briefly reviews related background. Section III describes the proposed algorithm. Section IV describes the experimental study, whose results are discussed in Section V. Finally, Section VI presents some concluding remarks.

II. Background

This section briefly reviews the most important developments related to distance and instance weighting in NN classifiers and overviews data gravitation models.

A. NN Models

The following proposals focus on improvements of the NN classifier to consider distance weighting and training set transformation because these ideas have been considered when designing the proposed algorithm.

NN classifiers simply perform a class voting among the k closest training instances. Thus, the real problem is to decide which are the k closest instances based on a distance criterion and how they compete for class voting.

Dudani introduced the distance-weighted NN rule (DW-KNN) [25] based on the idea that evidences nearby sample observations are stronger. More specifically, he proposed to weight the evidence of a neighbor close to an unclassified example more heavily than the evidence of another neighbor which is at a greater distance from the unclassified example. Thus, the k closest neighbors $p_1, p_2, \ldots, p_k$ are ordered so that p_1 is the nearest and p_k is the farthest from the unclassified example. The corresponding distances of these neighbors from the unclassified example are $d_1, d_2, \ldots, d_k$. The voting weight for a neighbor p_j is defined as

$$w_j = \left\{ \begin{array}{ll} \frac{d_k - d_j}{d_k - d_1} & \text{if} \quad d_k \neq d_1 \\ 1 & \text{if} \quad d_k = d_1 \end{array} \right\}. \quad (1)$$

Having computed the weights, the DW-KNN assigns the unclassified example to the class for which the weights of the representatives among the k NNs sum to the greatest value.

Gao and Wang applied this idea and proposed a center-based NN classifier (CNN) [26] which uses the distance between train instances and centers of their class as a reference of how far the train instance is from the unclassified example. CNN considers the center-based line passing through an example with known label and the center of the data class. However, this method performs bad classification when the center of the data classes is overlapped.

Wang *et al.* proposed an adaptive k NN algorithm (KNN-A) [27] which involves both a locally adaptive distance measure for identifying the NNs to an unknown example and a weighting scheme that assigns a weight to each NN based on its statistical confidence.

Paredes and Vidal [28], [29] proposed a class-dependent weighted dissimilarity measure in vector spaces to improve the performance of the NN classifier. Under their proposed framework, the accuracy is improved by using a dissimilarity measure such that distances between points belonging to the same class are small while interclass distances are large.

Prototype selection [30] and prototype generation [31] are also commonly employed together with NN classifiers to reduce the data size and to improve the accuracy.

Paredes and Vidal extended their NN model together with a prototype reduction algorithm [32], which simultaneously trains both a reduced set of prototypes and a suitable local metric for them. Starting with an initial selection of a small number of prototypes, their proposal iteratively adjusts both the position (features) of these prototypes and the corresponding local-metric weights.

Zhou and Chen [33] proposed a cam weighted distance (CamNN) to ameliorate the curse of dimensionality which optimizes the distance measure based on the analysis of inter-prototype relationships.

Triguero *et al.* [34] applied differential evolution to the prototype selection problem as a position adjusting of prototypes. Their experimental results advocate for the great performance of SSMA [35] + SFLSDE [36], which results in a combination of a prototype selection stage with an optimization of the position of prototypes, prior to the NN classification. Similarly, Jahromi *et al.* [37] improve the classification rate of the NN rule by adjusting the weights of the training instances, thus reducing the size of the training set.

As observed, these proposals weight the class voting of the k NNs based on the relative distances between data samples and/or perform data reduction via prototype selection or generation. The NN classification process is transparent and easy to understand, implement, and debug. However, most of these methods are very sensitive to irrelevant, redundant, or noisy features because all features contribute to the similarity/distance function and thus to the classification. This problem is enhanced as the dimensionality of the data increases.

B. Data Gravitation Models

The first machine learning algorithm inspired by the physical gravitation was proposed in 1977 by Wright [38] for performing cluster analysis on Euclidean data. More recently, Endo and Iwata [39] proposed in 2005 a dynamic clustering algorithm based on universal gravitation which takes advantage of both global and local information of the data. Although both proposals are focused on clustering, they are inspired by the gravitation principles.

Regarding gravitational classification models, Wang and Chen [9] presented in 2005 an improvement of the NN classifier using simulated gravitational collapse. They simulated the physical process of gravitational collapse to trim the boundaries of the distribution of each class, since the performance of the NN classifier drops significantly with the increase of the overlapping of the distribution of different classes. Their algorithm generates prototypes for the NN classifier by a migration of the original samples.

There are more contributions that apply gravitation theory to classification, such as Yang *et al.* [40] for abnormal network intrusion detection or Zong-Chang [10] who proposed in 2007 a vector gravitational force model derived from the vector geometric analysis of the linear classifier.

The most recent and complete work related to DGC was presented by Peng *et al.* [8] in 2009. They employ weights to describe the importance of the attributes in the distance calculation of the gravitation model. The weights are optimized by a random iterative algorithm named tentative random feature selection (TRFS). Artificial data particles are created from the training data to represent set of samples for the different classes. Gravitation from a particle to the different classes is calculated using the mass of the data particles (number of instances represented) and the distance to its centroid, which represents the mass center of the data particle.

The algorithm creates data particles using the maximum distance principle. However, this method reduces the accuracy, especially in the area away from the data particle centroid and along the border between classes. The gravitation function performs well on standard data, but it suffers severely from imbalanced data, which is one of the major drawbacks concluded by their authors.

III. Algorithm Description

This section presents the proposed algorithm, whose main characteristics are depicted in the following sections. First, the main concepts and principles of DGC are presented. Second, the definition of distance and the procedure to calculate the optimal weights of the attributes for the different classes are presented. Third, the adaptation of the gravitation function to consider the imbalance data problem is discussed. Finally, the main steps conducted by the evolutionary algorithm to obtain the classifier are summarized.

A. DGC Principles

Newton published in 1687 the law of universal gravitation, which states that every point mass in the universe attracts every other point mass with a force that is directly proportional to the product of their masses and inversely proportional to the square of the distance between them. General relativity generalizes Newton's law of universal gravitation, providing a unified description of gravity as a geometric property of space-time. The curvature of space-time is usually represented using embedding diagrams [41]. These diagrams show the gravity well, which represents the gravitational field surrounding an object in space. Each point in space will have a value of the gravitational field proportional to the function

$$\vec{g} = -G\frac{M}{r^2}\vec{u}_r \tag{2}$$

where G is the gravitational constant, M is the mass of the object, and r is the distance to the object.

The principle of DGC is to assign to an instance the class with the highest gravitational field magnitude. Therefore, given a sample $\vec{x}$, the algorithm calculates the gravitation to $\vec{x}$ using the training samples for the different classes and predicts the class with the highest gravitation

$$Class(\vec{x}) = \arg\max g(\vec{x}, k)\ \forall k \in C \tag{3}$$

where $g(\vec{x}, k)$ is the gravitation exerted to a data sample $\vec{x}$ by the instances from the class k and C is the set of classes. There are significant differences between this proposal, NN, and previous gravitation methods, especially concerning how gravitation is calculated considering the distances among the samples. These differences are detailed in the next sections.

B. Definition of Distance

Gravitation is inversely proportional to the square of the distance. Newtonian gravitation usually considers the case of Euclidean geometry. The Euclidean distance between two objects $\vec{x}_1$ and $\vec{x}_2$ is defined as

$$d(\vec{x}_1, \vec{x}_2) = \sqrt{\sum_{i=1}^{f}(x_{1i} - x_{2i})^2} \tag{4}$$

where the number of attributes (features or dimensions) is f. Normalization is an essential preprocessing step in data mining [42], especially concerning instance-based methods [43]. All numerical attributes have to be scaled to [0, 1] for an equitable calculation of distances using the min–max normalization [44]. Otherwise, distances from attributes with larger domains would disrupt others from smaller domains. In order to find a reasonable distance between a pair of samples with nominal attributes, the overlap metric is used for the labels. This metric is defined as

$$\delta(x_{1i}, x_{2i}) = \left\{\begin{matrix} 0 & \text{if} & x_{1i} = x_{2i} \\ 1 & \text{if} & x_{1i} \neq x_{2i} \end{matrix}\right\}. \tag{5}$$

Although the definition of distance from (4) considers all the attributes equally relevant, feature selection has been shown to improve the performance of classification algorithms [45], [46]. Feature selection selects a subset of relevant features and removes most irrelevant and redundant features from the data. Embedded feature selection methods [47] realize that the learning part and the feature selection part cannot be separated.

Peng *et al.* [8] proved that weighting attributes performed well in calculating the gravitation using the relevant attributes. Weighting attributes within the learning process is a way of embedded feature selection by giving more or less importance to the different attributes. However, similar to Paredes and Vidal [28], [29], we consider that the weight of each attribute must be different for each class since the distribution of the samples is not usually similar. Thus, instead of using an attribute weight vector of length f, the proposed algorithm employs a class-dependent attribute-class weight matrix $W[f, c]$, where f is the number of attributes and c is the number of classes.

$$W = \begin{bmatrix} w_{1,1} & \dots & w_{1,c} \\ \dots & \dots & \dots \\ w_{f,1} & \dots & w_{f,c} \end{bmatrix}. \tag{6}$$

Therefore, the distance function from (4) is rewritten to take into account the weights of the attributes for each class

$$d(\vec{x}_1, \vec{x}_2, k) = \sqrt{\sum_{i=1}^{f} w_{i,k} \cdot (x_{1i} - x_{2i})^2} \tag{7}$$

where $w_{i,k}$ is the weight of the attribute i for the class k.

The optimization of the W attribute-class weight matrix is a real optimization problem whose dimension $f \cdot c$ depends on the number of attributes and the number of classes. Accordingly, the CMA-ES is one of the most powerful evolutionary algorithms for difficult nonlinear nonconvex real-valued single-objective optimization [11]. The main advantages of CMA-ES lie in its invariance properties, which are achieved by carefully designed variation and selection operators, and in its efficient self-adaptation of the mutation distribution. CMA-ES does not require a tedious parameter tuning for its application since a strategy for finding good parameters is considered as part of the algorithm design and not as part of its application. Therefore, the proposal employs CMA-ES to optimize the real values of the attribute-class weight matrix instead of the TRFS algorithm proposed by Peng *et al.* [8] to optimize their attribute weight vector.

C. Definition of Gravitation

Gravitation depends on the mass and the distances among the objects. Peng *et al.* [8] perform data transformation by creating artificial data particles which replace the original training samples. Gravitation to a data class is calculated using the superposition principle, i.e., the sum of gravitation to the samples belonging to the data class. Therefore, their approach may fail because of the imbalance data since gravitation from a certain class is extremely strong or extremely weak, misclassifying minority class examples. The problem of learning from imbalanced data is a relatively new challenge that has attracted growing attention from both academia and industry [48]–[50]. This problem is concerned with the performance of learning algorithms in the presence of underrepresented data and severe class distribution skews.

This problem would be enhanced in our proposal since it considers all of the training instances. Gravitation of minority classes would be eclipsed by those of the majority classes. Therefore, the gravitation of a class is weighted by its number of instances and the total number of instances. In this way, a balance between the gravitation of majority and minority classes is achieved. Finally, the gravitation of a sample $\vec{x}$ for a class k is defined as

$$g(\vec{x}, k) = \left(1 - \frac{N_k - 1}{N}\right) \cdot \sum_{i=1}^{N} \frac{1}{d(\vec{x}_i, \vec{x}, k)^2} | \vec{x}_i \in k \quad (8)$$

where N_k is the number of instances of the class k and N is the total number of instances.

Peng *et al.*'s proposal calculates the centroid of data particles, which is the center of masses, for distance computation. However, the weakness of the data class centroid is the loss of information about the shape of the instance cloud (local information), which does not happen when using all of the samples. The aforementioned problem is especially noticeable when the shape of the instance cloud is irregular; hence, the path of the border between the classes cannot be properly fitted. Instead, our proposal deals with all of the training samples. The advantage of using all of the samples to calculate gravitation is that it provides accurate classification in local classification, where there are many closer train instances which provide high gravitation nearby. It also provides good generalization where there are no closer training instances (global information) since gravitation is inversely proportional to the square of the distance, disregarding confidence from those faraway.

D. Evolutionary Algorithm

The main steps conducted by the algorithm to obtain the classifier, the optimal attribute-class weights, and the predictions for a data set are now summarized.

First, all attributes from the data set are scaled and normalized to the range [0, 1] using the min–max normalization. The problem of absolute distance calculation is that the outcome depends on the range and the values of each attribute domain. Thus, distances from attributes with larger domains would disrupt others from smaller domains. Therefore, scaling is used to avoid this problem and to relativize the distance calculation from the different attributes. For every attribute in the data set, we find the maximum value $v_{\max}$ and the minimum value $v_{\min}$. For every instance within the data set and attribute, the normalized attribute value v_{norm} for an original attribute value v is scaled as follows:

$$v_{\text{norm}} = \frac{v - v_{\min}}{v_{\max} - v_{\min}}. \quad (9)$$

Second, we must find the optimal values for the attribute-class weight matrix W in order to discriminate irrelevant or noisy attributes and to enhance truly relevant attributes for each class. Therefore, we encode a population of real array individuals to represent the weights of the attributes for each class. The initial values are set to 0.5, i.e., all attributes are initially considered equally relevant. CMA-ES is run to optimize these values according to the fitness function, which evaluates the accuracy of the candidate gravitation classifiers using the current attribute-class matrix values and the train data. The configuration settings and parameters of CMA-ES are described in Section IV-C. When CMA-ES meets any of the stop criteria, the best attribute-class matrix from the CMA-ES population is employed to build the final classifier.

Finally, the class predictions are obtained from submitting the instances from the data set to the final classifier. The class predictions are compared to the actual class of the instances, obtaining the values of the confusion matrix. The confusion matrix is used to compute the quality performance metrics described in Section IV-B.

IV. Experimental Study

This section describes the details of the experiments performed in order to evaluate the capabilities of the proposal and compares it to other classification methods. The experimental study is divided in two sets of experiments. On the one hand, the proposal is evaluated and compared to seven other instance-based classification techniques over 35 standard classification data sets. On the other hand, the proposal is evaluated over 44 imbalanced data sets and compared to eight other techniques specifically designed for imbalanced classification.[1]

First, the application domains are presented, followed by a description of the performance measures to evaluate. Then, the algorithms used for the comparison and the description of the experimental methodology are presented. Finally, the statistical tests used for validating the results are described.

A. Problem Domains

The data sets used in the experiments have been collected from the KEEL [13] and UCI [14] machine learning repository Web sites, and they are very varied in their degree of complexity, number of classes, number of attributes, number of instances, and imbalance ratio (the ratio of the size of the majority class to the size of the minority class). There are 35 standard and 44 imbalanced data sets. The number of classes ranges up to 11, the number of attributes ranges from 2 to 60, the number of instances ranges from 148 to 12 690, and the imbalance ratio is up to 129.44. Detailed information about the data sets and their number of instances, classes, attributes, and imbalance ratio are shown in the Web site.[1]

The standard data sets are partitioned using the stratified tenfold cross-validation procedure [51], [52]. The imbalanced data sets are partitioned using the stratified fivefold cross-validation procedure to ensure the presence of minority class instances in the test sets. All experiments are repeated with ten different seeds for stochastic methods.

B. Performance Measures

There are many performance measures for classification algorithms depending on the classification subfield problem [53], [54]. Different measures allow us to observe different behaviors, which increases the strength of the empirical study in such a way that more complete conclusions can be obtained from different (not opposite yet complementary) deductions [55], [56]. These measures are based on the values of the confusion matrix, where each column of the matrix represents the count of instances in a predicted class, while each row represents the number of instances in an actual class.

The standard performance measure for classification is the accuracy rate, which is the number of successful predictions relative to the total number of classifications. However, the accuracy rate may be misleading when data classes are strongly imbalanced since the all-positive or all-negative classifier may achieve a very good classification rate. Real-world problems frequently deal with imbalanced data. Therefore, the evaluation of the model should be done by means of other criteria rather than accuracy.

Cohen's kappa rate [17], [18] is an alternative measure to accuracy since it compensates for random hits. The kappa measure evaluates the merit of the classifier, i.e., the actual hits that can be attributed to the classifier and not by mere chance. Cohen's kappa statistic ranges from -1 (total disagreement) to 0 (random classification) to 1 (total agreement). It is calculated by means of the confusion matrix as follows:

$$Kappa = \frac{N\sum_{i=1}^{k} x_{ii} - \sum_{i=1}^{k} x_{i.}x_{.i}}{N^2 - \sum_{i=1}^{k} x_{i.}x_{.i}} \tag{10}$$

where x_{ii} is the count of cases in the main diagonal of the confusion matrix, N is the number of examples, and $x_{.i}$ and $x_{i.}$ are the column and row total counts, respectively. Cohen's kappa rate also penalizes all-positive or all-negative predictions, especially in imbalanced data problems. Kappa is very useful for multiclass problems, measuring a classifier's accuracy while compensating for random successes.

The area under the receiver operating characteristic (ROC) curve (AUC) [19], [20] is a commonly used evaluation measure for imbalanced classification. The ROC curve presents the tradeoff between the true positive rate and the false positive rate [57]. The classifier generally misclassifies more negative examples as positive examples as it captures more true positive examples. AUC is computed by means of the confusion matrix values: true positives (T_P), false positives (F_P), true negatives (T_N), and false negatives (F_N), and it is calculated by relating the true positive and false positive ratio

$$AUC = \frac{1+TP_{\text{rate}}-FP_{\text{rate}}}{2} = \frac{1+\frac{T_P}{T_P+F_N}-\frac{F_P}{F_P+T_N}}{2}. \tag{11}$$

C. Comparison of the Algorithms and Experimental Settings

This section describes the algorithms used in the experimental study and their parameter settings, which are obtained from the KEEL [15] and WEKA [16] software tools. Several instance-based methods, including the most recent DGC method, have been selected and compared to determine whether the proposal is competitive in different domains with the other approaches. Algorithms are compared on equal terms and without specific settings for each data problem. The parameters used for the experimental study in all classification methods are the optimal values from the tenfold cross-validation, and they are now detailed.

1) Classification Methods for Standard Data Sets: Table I summarizes the experimental settings for the standard classification algorithms.

1) DGC [8] employs weights to describe the importance of the attributes in the DGC model. It transforms the data set using data particles, which are constructed using the nearest instances to reduce the complexity of the data set. Gravitation from one particle to the different classes is calculated using the mass of the data particles (number of instances represented) and the distance to its centroid, which represents the mass center of the data particle.
2) KNN [5] classifies an instance with the class with the higher value of the number of neighbors to the instance that belongs to such class. Vicinity is defined as the

[1]The data set description together with their partitions, the algorithms as stand-alone runnable files, and the experimental settings are fully described and publicly available to facilitate the replicability of the experiments and future comparisons with previous or new proposals in the Web site http://www.uco.es/grupos/kdis/kdiswiki/DGC.

TABLE I
EXPERIMENTAL SETTINGS FOR STANDARD CLASSIFICATION

Algorithm	Parameter	Value
DGC	Population size	100
	Number of generations	200
	Distance to centroid threshold	0.2
KNN	Number of neighbours	3
	Distance	Euclidean
KNN-A	Number of neighbours	3
	Distance	Euclidean
DW-KNN	Number of neighbours	3
	Distance	Euclidean
CamNN	Number of neighbours	3
CNN	No parameters	
SSMA-SFLSDE	Population size	30
	Number of evaluations	10000
	Cross prob per bit	0.5
	Mutation prob per bit	0.001
	Number of Neighbours	1
	Distance	Euclidean
	Auxiliar population size	50
	Maximum of iterations	500
	Iterations SFGSS	8
	Iterations SFHC	20
	Fl	0.1
	Fu	0.9
	Tau1	0.1
	Tau2	0.1
	Tau3	0.03
	Tau4	0.07
	Strategy	RandToBest

k instances with lower distances to the instance being classified.

3) Adaptive KNN (KNN-A) [27] is a classifier in which distances are weighted by the distance from the training instance to its nearest instance belonging to a different class. Thus, instances near to the decision boundaries become more relevant.
4) DW-KNN [25] weights the vote of the k NNs regarding their relative distances to the uncategorized example.
5) Cam weighted distance (CamNN) [33] addresses the curse of dimensionality optimizing a distance measure based on the analysis of relations among data samples.
6) Center NN (CNN) [26] classifier is an enhanced 1-NN classifier which uses the distance between train instances and centers of their class as a reference of how far the train instance is from another instance. This algorithm is parameter-free.
7) SSMA-SFLSDE [34] combines a prototype selection stage with an optimization of the position of prototypes, prior to the NN classification.

2) Classification Methods for Imbalanced Data Sets: There are two common approaches to deal with imbalanced data: cost-sensitive learning and resampling methods.

Cost-sensitive learning modifies the algorithm by introducing numeric costs to any formula used to estimate the error of the model. A model builder will take into account the different costs of the outcome classes and will build a model that does not readily dismiss the very much underrepresented outcome.

Resampling is a data preprocessing step, and it aims to redress the balance. Thus, the base classifier used by the model builder does not need to be modified. Resampling methods are divided in two categories: undersampling (remove overrepresented class instances) and oversampling (generate underrepresented class instances). The resampling methods used to balance data class distribution are the following.

TABLE II
EXPERIMENTAL SETTINGS FOR IMBALANCED CLASSIFICATION

Algorithm	Parameter	Value
ADAC2	Pruned	True
	Confidence	0.25
	Instances per leaf	2
	Number of classifiers	10
	Train method	Cost-sensitive
	Cost set-up	Adaptive
NN-CS	Hidden layers	2
	Hidden nodes	15
	Transfer function	Htan
	Eta	0.15
	Alpha	0.1
	Lambda	0.0
	Cycles	10000
CSVM-CS	Kernel	Polynomial
	C	100.0
	eps	0.001
	degree	1
	vgamma	0.01
	coef0	0.0
	shrinking	True
C4.5-CS	Pruned	True
	Confidence	0.25
	Instances per leaf	2
	Minimum expected cost	True
C4.5-RUS	Pruned	True
	Confidence	0.25
	Instances per leaf	2
C4.5-SMOTE	Number of neighbours	5
	Distance	HVDM [65]
	Interpolation	Standard
	Pruned	True
	Confidence	0.25
	Instances per leaf	2
C4.5-SMOTE-TL	Number of neighbours	5
	Distance	HVDM [65]
	Pruned	True
	Confidence	0.25
	Instances per leaf	2

1) Random undersampling (RUS) [58] randomly removes a subset of the overrepresented class to approach the same number as the underrepresented class.
2) Synthetic minority oversampling technique (SMOTE) [59] generates underrepresented class instances from other instances in the original data set by selecting k NNs and using them to perform arithmetical operations to generate new instances.
3) Tomek links (TLs) [60] obtain the instance set belonging to spaces near the decision boundaries. SMOTE is usually employed together with TL (SMOTE-TL) [58] for oversampling with SMOTE and then for removing instances near boundaries using TL.

These resampling techniques are combined with a classifier to produce accurate imbalanced classification results. Next, some commonly used imbalanced data classifiers, whose parameters for the experimental study are summarized in Table II, are now detailed.

1) ADAC2 [61] is a boosting algorithm that produces an ensemble of decision trees (C4.5) from a set of given examples. Each instance has an associated cost depending

on the class. It tries to solve the imbalance problem by increasing the weight of the instances from the minority class in each iteration of the boosting algorithm. It can be executed using the cost-sensitive C4.5 or a resampling procedure after the weight update. C2 modification refers to the way in which the costs are introduced in the boosting procedure. The costs are automatically selected by considering the imbalance ratio.

2) NN-CS [62] is a neural network presented by Zhou and Liu, in which they study the effect of sampling and threshold-moving methods that have been shown to be effective in addressing the class imbalance problem, applied to cost-sensitive neural networks.
3) CSVM-CS [63] builds an SVM model with the training data, taking into account the cost associated to the class distribution, and then classifies all test data by means of the trained SVM.
4) C4.5-CS [64] is the C4.5 decision tree algorithm that weights the instances of the input data to consider the associated costs from imbalanced classification.
5) C4.5-RUS preprocesses the data set by using RUS to delete overrepresented class instances randomly. Then, C4.5 decision tree is applied over the transformed data.
6) C4.5-SMOTE preprocesses the data set by using SMOTE to generate underrepresented class instances. Then, C4.5 decision tree is applied over the transformed data.
7) C4.5-SMOTE-TL preprocesses the data set by using SMOTE to generate underrepresented class instances and then removes instances near the boundaries using TL. Finally, C4.5 decision tree is applied over the transformed data.

On the other hand, the proposed DGC+ has been implemented in the JCLEC software [66], it is available as a WEKA module, and its parameters are the ones from the CMA-ES algorithm. CMA-ES does not require a laborious parameter tuning since finding good parameters is considered as part of the algorithm design and not as part of its application. For the application of CMA-ES, the optimal values recommended by Hansen [11] are employed, and it is only necessary to detail some parameter information. CMA-ES employs a chromosome whose dimension (C_L) is the number of classes times the number of attributes. Each gene represents the weight of an attribute for a certain class. The initial solution is set to 0.5 for every gene within the chromosome, and the initial standard deviation is set to 0.3. CMA-ES is run to optimize the attribute-class weight matrix which obtains the highest accuracy. The termination criterion is a drop of the difference of the fitness function values below $1E-13$, and the maximum number of iterations is set to 500. The population size self-adapts to the problem dimension, and it is set to $4 * \log C_L^2$. In CMA-ES, the population size can be freely chosen because the learning rates prevent degeneration even for small population sizes. Small population sizes usually lead to faster convergence, while large population sizes help in avoiding local optima. The number of restarts is two, and the population size is not increased.

D. Statistical Analysis

In order to analyze the results from the experiments, some nonparametric statistical tests are used to validate the results [23], [67]. To evaluate whether there are significant differences in the results of the algorithms, the Iman and Davenport test is performed. This useful nonparametric test, recommended by Demsar [68], is applied to rank the K algorithms over the D data sets according to the F-distribution. If the Iman and Davenport test indicates that the results are significantly different, the Bonferroni–Dunn post hoc test [21] is used to find the significant differences occurring between algorithms in multiple comparisons. It assumes that the performance of two classifiers is significantly different if the corresponding average ranks differ by at least a critical difference value. Finally, the Wilcoxon rank-sum test [22] is used to perform multiple pairwise comparisons among the algorithms [69]. The Wilcoxon rank-sum test statistic is the sum of the ranks for observations from one of the samples.

V. Results

This section presents and discusses the experimental results from the different experimental studies. First, the accuracy and Cohen's kappa rate for standard data sets are compared for the different methods, and the results are validated using nonparametric statistical tests. Second, the AUC and Cohen's kappa rate for imbalanced data sets are compared, and the results are validated. Third, the convergence process of the proposal and the output weights are analyzed. Finally, a discussion is carried out to analyze the reasons for the results obtained.

A. Standard Data Sets

1) Accuracy: Table III shows the average accuracy results from the tenfold cross-validation test for the different data sets (rows) and methods (columns). The proposed gravitation algorithm outperforms the other methods in 16 of the 35 data sets and obtains competitive accuracy results in the other data sets. Next, some interesting comments to highlight the accuracy results for some methods and data sets are discussed.

Hayes-Roth is an artificial data set originally created to test the behavior of prototype-based classifiers. It contains an attribute which was generated at random, adding noise to the data. Therefore, NN methods sensitive to noisy data obtain the worst accuracy since they are not able to filter the noisy attribute. They are KNN, KNN-A, CamNN, and CNN, with the original KNN being the worst with only an accuracy of 0.2500. On the other hand, the DW-KNN is capable of avoiding noisy data, as well as the prototypes generated by SSMA+SFLSDE and DGC data particles. These proposals overcome successfully the noise and perform much better than the other algorithms, proving the good performance of attribute weighting and prototype generation. Finally, DGC+ achieves the highest accuracy for this data set with the best noisy attribute filtering.

The Iman and Davenport test establishes an F-distribution value $= 2.0482$ for a significance level of alpha $= 0.05$. The Iman and Davenport statistic (distributed according to the

TABLE III
ACCURACY RESULTS FOR STANDARD DATA SETS

Data set	DGC+	DGC	KNN	KNN-A	DW-KNN	Cam-NN	CNN	SSMA+SFLSDE
Appendicitis	0.8409	0.8713	0.8427	**0.8882**	0.8227	0.8491	0.7736	0.8509
Australian	0.8374	0.8496	0.8478	0.8391	0.8319	0.8522	0.8174	**0.8594**
Balance	0.8966	**0.8999**	0.8337	0.8943	0.8560	0.8670	0.7855	0.8896
Banana	0.8952	0.8931	0.8864	0.8940	0.8836	0.8858	0.5725	**0.8992**
Bupa	**0.6744**	0.6527	0.6066	0.6257	0.6376	0.5962	0.6316	0.6426
Car	**0.9523**	0.9126	0.9231	0.8797	0.8148	0.9045	0.8773	0.9097
Contraceptive	0.4945	0.4954	0.4495	0.4827	0.4488	0.4719	0.4203	**0.4969**
Dermatology	0.9544	0.9170	**0.9690**	0.9521	0.9633	0.8966	0.9578	0.9410
Ecoli	0.8217	0.7672	0.8067	0.8247	**0.8248**	0.7824	0.6789	0.8009
Flare	0.6655	0.6733	0.6229	0.4634	**0.7148**	0.6041	0.6492	0.7102
German	**0.7322**	0.7020	0.6960	0.7270	0.7210	0.7120	0.7080	0.7100
Glass	0.7036	0.6893	0.7011	0.6498	**0.7218**	0.6102	0.7094	0.7176
Haberman	0.7171	**0.7277**	0.7058	0.6927	0.6863	0.6963	0.6865	0.6994
Hayes-Roth	**0.8400**	0.7738	0.2500	0.3750	0.7063	0.4750	0.4625	0.7562
Heart	**0.8452**	0.8119	0.7741	0.8111	0.7630	0.7704	0.7667	0.8296
Hepatitis	0.8628	0.8343	0.8251	0.8508	0.8108	**0.8651**	0.8233	0.7617
Ionosphere	0.9311	0.6724	0.8518	**0.9372**	0.8747	0.7379	0.8917	0.9088
Iris	**0.9533**	**0.9533**	0.9400	**0.9533**	0.9400	0.9467	0.9267	**0.9533**
Lymphography	**0.8140**	0.8033	0.7739	0.8085	0.7855	0.7530	0.7599	0.7950
Monk-2	**0.9995**	0.9982	0.9629	0.7058	0.8437	0.8494	0.7490	0.9681
New-thyroid	**0.9786**	0.8684	0.9537	0.9721	0.9677	0.8693	0.8753	0.9677
Nursery	**0.9696**	0.9378	0.9254	0.8610	0.8131	0.8637	0.7875	0.8489
Page-blocks	0.9508	0.9268	0.9591	**0.9629**	0.9624	0.8942	0.9543	0.9530
Phoneme	0.8718	0.8471	0.8849	0.8901	**0.8999**	0.8679	0.8830	0.8533
Pima	0.7451	0.6662	0.7319	**0.7476**	0.7163	0.7280	0.6994	0.7463
Saheart	**0.7118**	0.7105	0.6818	0.6927	0.6752	0.7099	0.6407	0.6860
Sonar	0.8487	0.7694	0.8307	0.8798	0.8648	0.7743	**0.8940**	0.8079
Tae	**0.6715**	0.6709	0.4113	0.4375	0.5704	0.5112	0.3983	0.5371
Thyroid	**0.9704**	0.9256	0.9389	0.9396	0.9368	0.9300	0.9250	0.9463
Tic-tac-toe	**0.8549**	0.6906	0.7756	0.7672	0.6848	0.7629	0.7317	0.7348
Vehicle	0.7116	0.6572	0.7175	0.6879	0.7258	0.6170	**0.7423**	0.6456
Vowel	0.9824	0.9788	0.9778	0.9687	0.9909	0.8879	**0.9929**	0.8980
Wine	**0.9731**	0.9706	0.9549	0.9663	0.9438	0.9497	0.9663	0.9438
Yeast	**0.5926**	0.5151	0.5317	0.5539	0.5418	0.4515	0.4670	0.5708
Zoo	**0.9553**	0.9348	0.9281	0.9447	0.9447	0.8919	0.9281	0.9100
Avg. Values	**0.8349**	0.7991	0.7849	0.7865	0.7969	0.7667	0.7581	0.8043
Avg. Ranks	**2.3857**	4.3857	4.8286	3.8429	4.7714	5.8000	5.9143	4.0714

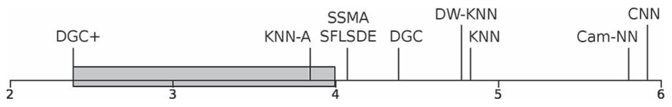


Fig. 1. Bonferroni–Dunn test for accuracy and standard data.

F-distribution with 7 and 238 degrees of freedom) is 9.2342 for accuracy. Thus, the test rejects the null hypothesis, and therefore, it can be said that there are statistically significant differences between the accuracy results of the algorithms.

Fig. 1 shows the application of the Bonferroni–Dunn test to the accuracy rate with alpha $= 0.05$, whose critical difference is 1.5751. This graph represents a bar chart, whose values are proportional to the mean rank obtained from each algorithm. The critical difference value is represented as a thicker horizontal line, and those values that exceed this line are algorithms with significantly different results than the control algorithm, which is the proposed DGC+. Therefore, the algorithms right beyond the critical difference from the proposal value are significantly worse. Observing this figure, all the other methods but KNN-A perform significantly worse than DGC+. DGC+ successfully overcomes the other gravitation method (DGC), which obtains the fourth best ranking and third best average results.

Table IV shows the results of the Wilcoxon rank-sum test for accuracy to compute multiple pairwise comparisons among the proposal and the other methods. KNN-A is the best among the other methods, but DGC+ outperforms it, and there are significant differences between the two algorithms with a confidence level higher than 99%.

TABLE IV
WILCOXON TEST FOR ACCURACY AND STANDARD DATA

DCG+ vs	R^+	R^-	p-value
DGC	535.0	60.0	1.126E-5
KNN	592.0	38.0	3.840E-7
KNN-A	494.0	101.0	4.648E-4
DW-KNN	536.5	93.5	1.338E-4
Cam-NN	616.0	14.0	6.402E-9
CNN	583.0	47.0	1.253E-6
SSMA+SFLSDE	528.0	67.0	2.344E-5

2) Cohen's Kappa Rate: Table V shows the average kappa results from the tenfold cross-validation test for the different data sets (rows) and methods (columns). DGC+ outperforms the other methods in 15 of the 35 data sets and obtains competitive kappa results in the other data sets.

In the previous section, it was mentioned how noise-sensitive NN methods performed badly over the Hayes-Roth data set. The kappa metric is able to detect these errors and penalizes this behavior with a significantly lower kappa value. Interestingly, the kappa rate of KNN over Hayes-Roth is negative, i.e., the predictions are completely wrong because of the noise, even performing worse than doing random predictions.

TABLE V
COHEN'S KAPPA RATE RESULTS FOR STANDARD DATA

Data set	DGC+	DGC	KNN	KNN-A	DW-KNN	Cam-NN	CNN	SSMA+SFLSDE
Appendicitis	0.4549	0.4710	0.4565	**0.5880**	0.4650	0.3808	0.2773	0.4573
Australian	0.6724	0.6976	0.6917	0.6719	0.6607	0.7035	0.6313	**0.7146**
Balance	0.8083	**0.8144**	0.7004	0.8042	0.7403	0.7559	0.6271	0.7953
Banana	0.7873	0.7825	0.7699	0.7850	0.7642	0.7675	0.1367	**0.7955**
Bupa	**0.3076**	0.2220	0.1944	0.2021	0.2645	0.1024	0.2571	0.2731
Car	**0.8981**	0.8025	0.8264	0.7009	0.5255	0.7935	0.7457	0.8023
Contraceptive	0.1994	0.1952	0.1358	0.1730	0.1517	0.1396	0.1079	**0.2112**
Dermatology	0.9425	0.8939	**0.9608**	0.9394	0.9538	0.8689	0.9468	0.9257
Ecoli	0.7498	0.6585	0.7300	0.7539	**0.7579**	0.6882	0.5620	0.7245
Flare	0.5662	0.5738	0.5101	0.2757	**0.6314**	0.4753	0.5518	0.6261
German	0.2812	0.0092	0.2194	0.2539	**0.3015**	0.1651	0.2880	0.2560
Glass	0.5834	0.5548	0.5887	0.5224	**0.6194**	0.4229	0.6037	0.6069
Haberman	0.0558	0.0315	0.1362	0.0752	0.1374	0.1417	**0.1832**	0.0935
Hayes-Roth	**0.7468**	0.6294	-0.2326	0.0441	0.5217	0.1835	0.1721	0.6146
Heart	**0.6826**	0.6094	0.5420	0.6168	0.5192	0.5420	0.5291	0.6501
Hepatitis	**0.5512**	0.2000	0.4683	0.4069	0.4084	0.3615	0.4672	0.3451
Ionosphere	0.8487	0.1142	0.6494	**0.8595**	0.7083	0.5145	0.7526	0.7986
Iris	**0.9300**	**0.9300**	0.9100	**0.9300**	0.9100	0.9200	0.8900	**0.9300**
Lymphography	**0.6289**	0.6026	0.5507	0.6162	0.5765	0.4993	0.5333	0.5974
Monk-2	**0.9991**	0.9963	0.9254	0.3987	0.6818	0.6965	0.4968	0.9357
New-thyroid	**0.9544**	0.6566	0.8957	0.9362	0.9282	0.7570	0.7073	0.9310
Nursery	**0.9551**	0.9083	0.8907	0.7963	0.7279	0.8001	0.6860	0.7786
Page-blocks	0.7152	0.4457	0.7655	**0.7890**	0.7887	0.5792	0.7445	0.7246
Phoneme	0.6898	0.5889	0.7172	0.7296	**0.7539**	0.6764	0.7144	0.6379
Pima	0.4063	0.0702	0.3892	**0.4259**	0.3576	0.3838	0.3191	0.4196
Saheart	0.3081	0.2708	0.2646	0.2929	0.2510	**0.3335**	0.2056	0.2687
Sonar	0.6943	0.5187	0.6554	0.7549	0.7248	0.5364	**0.7861**	0.6100
Tae	**0.5062**	0.5049	0.1171	0.1559	0.3563	0.2650	0.0946	0.3017
Thyroid	**0.7709**	-0.0002	0.4012	0.3593	0.4235	0.1049	0.3996	0.4608
Tic-tac-toe	**0.6607**	0.1472	0.4149	0.3998	0.1153	0.4243	0.4054	0.3941
Vehicle	0.6152	0.5437	0.6233	0.5844	0.6342	0.4905	**0.6563**	0.5273
Vowel	0.9807	0.9767	0.9756	0.9656	0.9900	0.8767	**0.9922**	0.8878
Wine	**0.9590**	0.9552	0.9318	0.9491	0.9152	0.9228	0.9491	0.9145
Yeast	**0.4695**	0.3309	0.3942	0.4225	0.4070	0.2413	0.3176	0.4430
Zoo	**0.9426**	0.9127	0.9041	0.9254	0.9255	0.8553	0.9043	0.8870
Avg. Values	**0.6664**	0.5320	0.5735	0.5744	0.5885	0.5248	0.5326	0.6097
Avg. Ranks	**2.5571**	4.9857	4.7714	4.1429	4.2143	5.9000	5.3571	4.0714

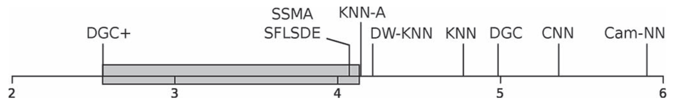


Fig. 2. Bonferroni–Dunn test for kappa rate and standard data.

Another interesting kappa result is found with the thyroid data set. The thyroid accuracy results from Table III do not suggest a noticeable difference in the accuracy performance of the different methods, all around 0.93. However, the kappa results do indicate considerable differences, i.e., DGC+ obtains 0.7709, whereas DGC interestingly obtains -0.0002.

The Iman and Davenport test establishes an F-distribution value $= 2.0482$ for a significance level of alpha $= 0.05$. The Iman and Davenport statistic (distributed according to the F-distribution with 7 and 238 degrees of freedom) is 7.0013 for kappa rate. Thus, the test rejects the null hypothesis, and therefore, it can be said that there are statistically significant differences between the kappa results of the algorithms.

Fig. 2 shows the application of the Bonferroni–Dunn test to kappa for alpha $= 0.05$, whose critical difference is 1.5751. Similar to the case with the accuracy rate, the algorithms right beyond the critical difference from the proposal's value are significantly worse. Observing this figure, all the other methods but SSMA+SFLSDE perform significantly worse than our proposal. Interestingly, the ranking of DGC is relegated to the sixth position.

TABLE VI
WILCOXON TEST FOR KAPPA AND STANDARD DATA

DGC+ vs	R^+	R^-	p-value
DGC	560.0	35.0	5.016E-7
KNN	553.0	77.0	3.154E-5
KNN-A	470.5	124.5	0.002405
DW-KNN	487.0	143.0	0.004030
Cam-NN	606.0	24.0	4.436E-8
CNN	552.5	77.5	3.308E-5
SSMA+SFLSDE	513.0	82.0	9.722E-5

Table VI shows the results of the Wilcoxon rank-sum test for kappa to compute multiple pairwise comparisons among the proposal and the other methods. Even though SSMA+SFLSDE is the algorithm with the second highest ranking in the multiple-algorithm comparison, DW-KNN is the best algorithm from the pairwise comparison versus DGC+. The p value for DGC+ versus DW-KNN is 0.004, which is lower than 0.01. Thus, our DGC+ proposal outperforms it and the others with a confidence level higher than 99% for pairwise comparisons.

B. Imbalanced Data Sets

1) AUC: Table VII shows the average AUC results from the cross-validation test for the different imbalanced data sets (rows) and methods (columns). DGC+ obtains the best results in 14 of the 44 data sets, and CSVM-CS achieves the best results in 12 of the 44 data sets. However, their

TABLE VII
AUC RESULTS FOR IMBALANCED DATA SETS

Algorithm	DGC+	DGC	ADAC2	NN	CSVM	C4.5	C4.5	C4.5	C4.5
Re-sampling / CS	None	None	CS	CS	CS	CS	RUS	SMT	SMT-TL
abalone19	0.6609	0.5000	0.5085	0.4949	**0.7615**	0.5701	0.6248	0.5719	0.5663
abalone9-18	0.8124	0.5000	0.7172	0.6490	**0.8740**	0.6655	0.6892	0.7892	0.7022
ecoli-0-1-3-7_vs_2-6	0.8427	0.7200	0.8154	0.7236	**0.8500**	0.8281	0.7900	0.7318	0.8172
ecoli-0_vs_1	0.9799	0.9642	0.9692	0.9796	0.9671	**0.9832**	0.9796	**0.9832**	0.9761
ecoli1	0.8804	0.7632	0.8763	0.8489	0.9062	**0.9114**	0.8852	0.8926	0.8801
ecoli2	**0.9378**	0.7919	0.8845	0.6880	0.5000	0.8905	0.8693	0.8906	0.8979
ecoli3	**0.8713**	0.6246	0.8478	0.8263	0.7925	0.8326	0.8453	0.8645	0.8502
ecoli4	0.9092	0.5300	0.9280	0.8702	**0.9529**	0.8636	0.8613	0.9513	0.8449
glass-0-1-2-3_vs_4-5-6	0.9215	0.7893	0.9033	**0.9308**	0.8445	0.8777	0.8811	0.9023	0.9102
glass-0-1-6_vs_2	0.6350	0.5000	0.5400	0.4038	0.5000	0.6155	0.6500	**0.7869**	0.6895
glass-0-1-6_vs_5	0.8363	0.5300	0.8800	0.8771	0.5000	**0.9886**	0.9429	0.9157	0.9243
glass0	**0.8650**	0.8553	0.8101	0.6792	0.5074	0.8212	0.8206	0.7754	0.8039
glass1	0.7496	0.7002	**0.7866**	0.6132	0.6264	0.7160	0.7153	0.7434	0.7711
glass2	0.7134	0.5000	0.7099	0.4396	0.5953	0.6416	0.6663	0.7112	**0.7172**
glass4	0.8869	0.5667	0.8706	0.8732	0.9126	0.8431	0.8347	0.8452	**0.9151**
glass5	0.8093	0.5000	0.9732	0.8805	0.9732	0.9427	0.9366	**0.9756**	0.8634
glass6	**0.9091**	0.8300	0.8923	0.9041	0.8725	0.8896	0.8824	0.8923	0.8965
habermanImb	0.6213	0.5062	0.5604	0.6245	0.5382	0.5752	0.6423	**0.6539**	0.6203
iris0	**1.0000**	0.9980	0.9900	**1.0000**	**1.0000**	0.9900	0.9900	0.9900	0.9900
new-thyroid1	**0.9939**	0.8023	0.9464	0.8679	0.9687	0.9746	0.9159	0.9492	0.9718
new-thyroid2	**0.9916**	0.8309	0.9575	0.8829	0.9829	0.9802	0.9071	0.9579	0.9440
page-blocks-1-3_vs_4	0.9395	0.6200	**0.9978**	0.9251	0.8566	0.9789	0.9436	0.9888	0.9774
page-blocks0	0.9412	0.6701	0.8816	0.7299	0.9254	**0.9458**	0.9448	0.9395	0.9388
pimaImb	**0.7394**	0.5274	0.7114	0.7175	0.7289	0.7125	0.7235	0.7134	0.6948
segment0	0.9927	0.9754	0.9826	0.5000	**0.9965**	0.9919	0.9788	0.9914	0.9914
shuttle-c0-vs-c4	0.9975	0.9845	0.9997	0.9070	**1.0000**	0.9997	**1.0000**	0.9994	0.9997
shuttle-c2-vs-c4	0.9743	0.7000	0.9500	0.7585	**1.0000**	**1.0000**	0.9840	0.9588	**1.0000**
vehicle0	0.9460	0.5533	0.9438	0.6801	**0.9493**	0.9289	0.9357	0.9274	0.9281
vehicle1	**0.7585**	0.5243	0.7531	0.6389	0.7546	0.7013	0.6820	0.6814	0.7422
vehicle2	0.9487	0.6782	**0.9729**	0.5945	0.9571	0.9434	0.9326	0.9556	0.9507
vehicle3	0.7516	0.5301	0.7345	0.6385	**0.7904**	0.7283	0.7005	0.7112	0.7463
vowel0	**0.9879**	0.7711	0.9706	0.6817	0.8461	0.9422	0.9438	0.9722	0.9850
wisconsinImb	0.9668	0.8986	0.9653	0.9582	**0.9719**	0.9636	0.9522	0.9579	0.9627
yeast-0-5-6-7-9_vs_4	0.7857	0.5000	0.7610	0.6894	0.5000	0.7243	0.7858	**0.8308**	0.7548
yeast-1-2-8-9_vs_7	0.6416	0.5000	0.6376	0.4307	0.5000	**0.6769**	0.6050	0.5966	0.5755
yeast-1-4-5-8_vs_7	0.5806	0.5000	0.5426	0.4893	0.5000	0.5540	0.5739	**0.5913**	0.5270
yeast-1_vs_7	**0.7594**	0.5000	0.7049	0.5771	0.5000	0.6139	0.6603	0.6871	0.6994
yeast-2_vs_4	0.9103	0.5120	0.9172	0.7330	0.5000	0.8866	0.8856	0.8810	**0.9309**
yeast-2_vs_8	0.7706	0.6650	0.6218	0.6588	0.7664	0.8652	0.7347	0.8415	**0.8686**
yeast1	**0.7183**	0.5116	0.6604	0.5864	0.6749	0.6779	0.7161	0.7112	0.6809
yeast3	**0.9202**	0.5533	0.9108	0.7643	0.8951	0.9117	0.9073	0.9119	0.9148
yeast4	**0.8252**	0.5000	0.7204	0.6554	0.8155	0.7222	0.8194	0.7420	0.7838
yeast5	0.9498	0.5236	0.8833	0.6139	0.9656	0.9330	0.9375	0.9517	**0.9712**
yeast6	0.8530	0.5057	0.7163	0.5891	**0.8758**	0.8082	0.8113	0.8309	0.8462
Avg. Values	**0.8520**	0.6479	0.8251	0.7176	0.7885	0.8321	0.8293	0.8443	0.8414
Avg. Ranks	**2.7273**	8.2841	4.8182	7.1023	4.7159	4.4432	4.8864	4.0114	4.0114

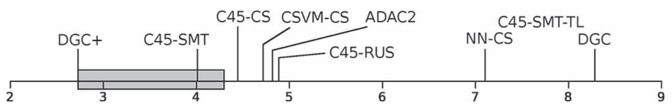


Fig. 3. Bonferroni–Dunn test for AUC and imbalanced data.

average results and rankings are significantly different. On the other hand, DGC stands out for achieving the worst AUC results.

The Iman and Davenport test establishes an F-distribution value $= 1.9653$ for a significance level of alpha $= 0.05$. The Iman and Davenport statistic (distributed according to the F-distribution with 8 and 344 degrees of freedom) is 26.2834 for AUC. Thus, the test rejects the null hypothesis, and therefore, it can be said that there are statistically significant differences between the AUC results of the algorithms.

Fig. 3 shows the application of the Bonferroni–Dunn test to AUC for alpha $= 0.05$, whose critical difference is 1.5905. All the other methods but C4.5-SMT and C4.5-SMT-TL show to perform statistically worse than DGC+.

Table VIII shows the results of the Wilcoxon rank-sum test for AUC. C4.5-SMOTE is the best from the other methods, but our gravitational proposal outperforms it, and the test reports a p value $= 0.0558$. On the other hand, DGC is the worst from among the other methods.

2) Cohen's Kappa Rate: Table IX shows the average kappa results from the cross-validation test for the different imbalanced data sets (rows) and methods (columns). DGC+ obtains the best results in 11 of the 44 data sets, and CSVM-CS achieves the best results in 12 of the 44 data sets. However, their average results and rankings are significantly different. On the other hand, NN-CS and DGC stand out for achieving the worst kappa results over imbalanced data.

The Iman and Davenport test establishes an F-distribution value $= 1.9653$ for a significance level of alpha $= 0.05$. The Iman and Davenport statistic (distributed according to the

TABLE VIII
WILCOXON TEST FOR AUC AND IMBALANCED DATA

Algorithm	DGC+	DGC	ADAC2	NN	CSVM	C4.5	C4.5	C4.5	C4.5
Re-sampling / CS	None	None	CS	CS	CS	CS	RUS	SMT	SMT-TL
abalone19	0.0185	0.0000	0.0052	-0.0010	0.0339	0.0370	0.0092	**0.0465**	0.0375
abalone9-18	0.3652	0.0000	0.3284	0.0934	**0.4341**	0.2958	0.1167	0.3554	0.2325
ecoli-0-1-3-7_vs_2-6	0.6221	0.4781	0.3632	0.0442	**0.7317**	0.5114	0.1509	0.2839	0.2665
ecoli-0_vs_1	0.9652	0.9436	0.9305	0.9598	0.9479	**0.9695**	0.9598	**0.9695**	0.9504
ecoli1	0.6783	0.5979	0.6815	0.5837	0.6926	**0.7577**	0.6843	0.7082	0.6881
ecoli2	**0.8505**	0.6718	0.6581	0.1979	0.0000	0.6803	0.6862	0.6874	0.6871
ecoli3	0.5613	0.3425	0.4968	0.3087	0.3783	0.5136	0.4597	**0.5685**	0.5466
ecoli4	**0.7221**	0.0925	0.7065	0.3152	0.6951	0.5525	0.3312	0.6905	0.5230
glass-0-1-2-3_vs_4-5-6	**0.8141**	0.6612	0.7976	0.7944	0.7441	0.7295	0.7373	0.7958	0.7858
glass-0-1-6_vs_2	0.2049	0.0000	0.0868	-0.1112	0.0000	0.1914	0.1170	**0.3853**	0.2581
glass-0-1-6_vs_5	0.5764	0.0662	0.6259	0.2666	0.0000	**0.8289**	0.4545	0.5164	0.6081
glass0	0.7037	**0.7248**	0.5812	0.2941	0.0181	0.5942	0.5942	0.5113	0.5431
glass1	0.4774	0.4501	**0.5323**	0.1956	0.2139	0.4131	0.4181	0.4468	0.5059
glass2	0.3039	0.0000	**0.3251**	-0.0507	0.0303	0.2171	0.0904	0.2380	0.3188
glass4	**0.6921**	0.1928	0.4202	0.2814	0.5595	0.3244	0.3912	0.4019	0.5872
glass5	0.5256	0.0000	0.7205	0.2263	0.7181	**0.8394**	0.4092	0.6200	0.3958
glass6	**0.8257**	0.7664	0.7412	0.6731	0.7704	0.7154	0.5907	0.7406	0.7972
habermanImb	0.1977	0.0182	0.0946	0.2399	0.0840	0.1110	0.2614	**0.2627**	0.1787
iris0	**1.0000**	0.9969	0.9846	**1.0000**	**1.0000**	0.9846	0.9846	0.9846	0.9846
new-thyroid1	**0.9648**	0.7069	0.8351	0.5898	0.9477	0.9191	0.6985	0.8504	0.9061
new-thyroid2	0.9648	0.7527	0.8923	0.5488	**0.9658**	0.9496	0.7129	0.8337	0.7686
page-blocks-1-3_vs_4	0.8725	0.3544	**0.9645**	0.5537	0.7707	0.9578	0.5443	0.8431	0.7248
page-blocks0	0.7481	0.4696	0.4374	0.2473	0.6402	**0.8254**	0.7302	0.7382	0.6814
pimaImb	0.4504	0.0682	0.3878	0.3943	**0.4551**	0.3976	0.4266	0.4064	0.3486
segment0	0.9827	0.9703	0.9662	0.0000	**0.9843**	0.9788	0.8953	0.9670	0.9670
shuttle-c0-vs-c4	0.9965	0.9830	0.9958	0.3890	**1.0000**	0.9958	**1.0000**	0.9916	0.9958
shuttle-c2-vs-c4	0.8803	0.4000	0.9297	0.1170	**1.0000**	**1.0000**	0.8575	0.6780	**1.0000**
vehicle0	0.8442	0.1530	0.8493	0.2384	**0.8896**	0.8419	0.8092	0.8186	0.7773
vehicle1	0.4281	0.0698	0.4241	0.2159	**0.4350**	0.3580	0.3019	0.3248	0.3978
vehicle2	0.8671	0.4478	**0.9358**	0.1318	0.8943	0.8700	0.8369	0.8838	0.8626
vehicle3	0.3901	0.0868	0.3920	0.2020	**0.4816**	0.3990	0.3229	0.3496	0.3909
vowel0	**0.9768**	0.6730	0.9475	0.1377	0.7627	0.8170	0.7252	0.8708	0.9006
wisconsinImb	0.9230	0.8281	0.9233	0.9164	**0.9392**	0.9114	0.8918	0.9104	0.9113
yeast-0-5-6-7-9_vs_4	0.4028	0.0000	0.3442	0.1385	0.0000	0.3655	0.3684	**0.4471**	0.3591
yeast-1-2-8-9_vs_7	0.0905	0.0000	0.1427	-0.0143	0.0000	**0.1821**	0.0417	0.0843	0.0632
yeast-1-4-5-8_vs_7	0.0600	0.0000	0.0604	-0.0017	0.0000	**0.0809**	0.0481	0.0636	0.0184
yeast-1_vs_7	**0.3169**	0.0000	0.1875	0.0332	0.0000	0.2081	0.1335	0.2388	0.2077
yeast-2_vs_4	**0.7310**	0.0343	0.6773	0.1744	0.0000	0.6510	0.6118	0.6115	0.7178
yeast-2_vs_8	0.4780	0.4607	0.2435	0.0633	0.5404	**0.6652**	0.1453	0.4415	0.4932
yeast1	0.3888	0.0327	0.2443	0.1280	0.3078	0.2834	**0.3989**	0.3849	0.2998
yeast3	0.6824	0.1659	0.6176	0.2064	0.6094	**0.7110**	0.6047	0.6999	0.6763
yeast4	0.2670	0.0000	0.2333	0.0404	0.2363	0.2632	0.1735	0.2865	**0.2872**
yeast5	0.6577	0.0825	**0.7537**	0.0278	0.4475	0.6901	0.4168	0.6504	0.6137
yeast6	**0.4042**	0.0196	0.2124	0.0133	0.2446	0.3669	0.1692	0.3423	0.3663
Avg. Values	**0.6108**	0.3355	0.5609	0.2683	0.4910	0.5899	0.4844	0.5666	0.5598
Avg. Ranks	**2.6818**	7.5341	4.3864	7.7841	4.5682	3.6818	5.9432	4.0000	4.4205

TABLE IX
COHEN'S KAPPA RATE RESULTS FOR IMBALANCED DATA SETS

DGC+ vs	R^+	R^-	p-value
DGC	990.0	0.0	1.14E-13
ADAC2	809.5	180.5	1.325E-4
NN-CS	915.0	31.0	5.40E-10
CSVM-CS	694.5	251.5	0.006719
C4.5-CS	768.0	222.0	0.001082
C4.5-RUS	834.0	156.0	3.128E-5
C4.5-SMT	659.0	331.0	0.055820
C4.5-SMT-TL	676.0	314.0	0.034240

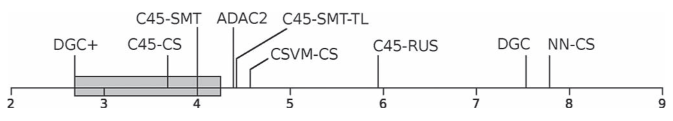


Fig. 4. Bonferroni–Dunn test for kappa and imbalanced data.

F-distribution with 8 and 344 degrees of freedom) is 28.8119 for kappa. Thus, the test rejects the null hypothesis, and therefore, it can be said that there are statistically significant differences between the kappa results of the algorithms.

TABLE X
WILCOXON TEST FOR KAPPA AND IMBALANCED DATA

DGC+ vs	R^+	R^-	p-value
DGC	984.0	6.0	1.59E-12
ADAC2	751.0	239.0	0.002298
NN-CS	941.0	5.0	2.27E-12
CSVM-CS	740.0	206.0	9.228E-4
C4.5-CS	649.0	341.0	0.073020
C4.5-RUS	964.0	26.0	1.22E-10
C4.5-SMT	771.0	219.0	9.418E-4
C4.5-SMT-TL	826.0	164.0	5.104E-5

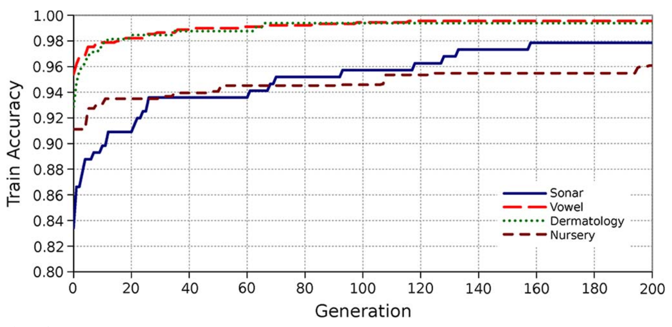


Fig. 5. Convergence rate.

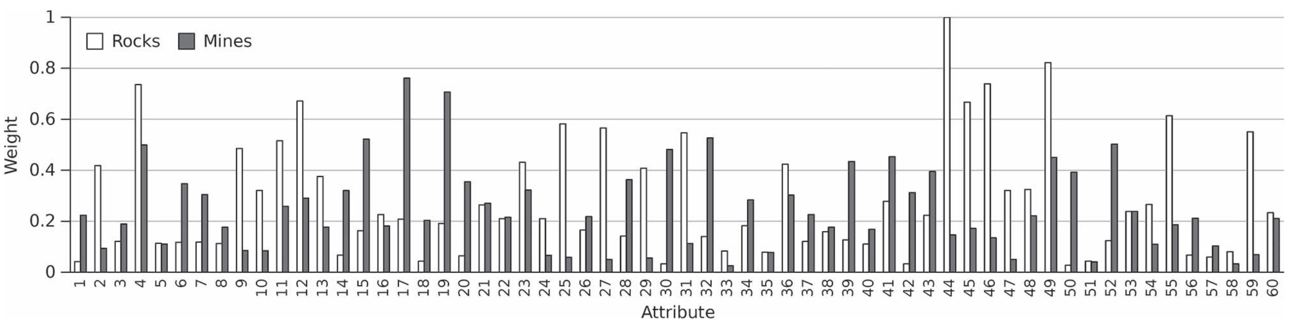


Fig. 6. Sonar weights.

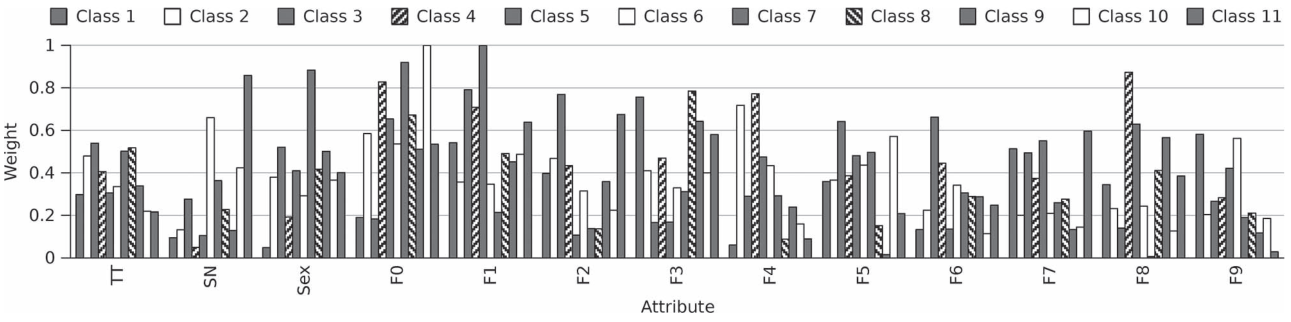


Fig. 7. Vowel weights.

Fig. 4 shows the application of the Bonferroni–Dunn test to kappa for alpha $= 0.05$, whose critical difference is 1.5905. C4.5-SMT is competitive with DGC+, together with C4.5-CS, but C4.5-SMT-TL and the other methods perform statistically worse than DGC+ regarding the Cohen's kappa rate.

Table X shows the results of the Wilcoxon rank-sum test for kappa. C4.5-CS is the best compared to the other methods, but our gravitational proposal outperforms it, and the test reports a p value $= 0.0730$. On the other hand, NN-CS and DGC are the worst from among the other methods.

C. Convergence Analysis

As mentioned in the description of the algorithm, the proposal assigns a weight to each attribute and class, and these weights allow us to recognize relevant attributes and thus improve the classification performance by giving less importance to those attributes that introduce noisy information into the learning process. This section presents the convergence rate of the best fitness values over the data sets sonar (2 classes and 60 attributes), vowel (11 classes and 13 attributes), nursery (5 classes and 8 attributes), and dermatology (6 classes and 34 attributes). These data sets have been selected because of their high dimensionality regarding the product of the number of classes and the number of attributes, which represents the genotype length of the CMA-ES algorithm, whose lengths are 120, 143, 40, and 204, respectively. Fig. 5 shows the best fitness values along the generations. The results at generation 0 indicate the initial accuracy when all of the attributes are considered with the same weight (0.5). The CMA-ES algorithm iterates, finding better weights for the different classes and attributes over the generations. When CMA-ES meets any of the stop criteria, the best solution from the CMA-ES population is selected, and its attribute-class matrix is the one shown in the previous section. The step-size control from CMA-ES effectively prevents premature convergence yet allows fast convergence to an optimum.

D. Attribute-Class Weighting Outputs

In this section, we evaluate and visualize the allocation of the weights assigned to the attributes over the data sets analyzed in the previous section. The weights assigned are shown in Figs. 6–9, and they are valued within the range [0, 1], where 0 stands for unconsidered attributes and 1 for very important ones. The most relevant attributes detected in sonar are the frequency bands 4, 44, 46, and 49 for the rock class and 17 and 19 for the mine class, whereas bands 5, 33, 51, 57, and 58 are not considered for any class. Regarding vowel data set, attributes F0 and F1 are the most relevant to classify class 10 and class 5, respectively. On the other hand, F5 is the less relevant attribute for class 9. The weights from the nursery data set indicate that the social behavior is not considered (but used for predicting the class spec_prior), whereas the health attribute is the most relevant (but for priority class). Finally, the high deviation among the weight bars of the dermatology data set indicate that the algorithm has been able to discriminate many attributes, weighted lower than 0.2, and few attributes have been considered truly relevant, weighted higher than 0.5.

Finally, it can be concluded that having different weights for each class is a convenient idea as seen in the examples (since an attribute is not equally relevant for all of the classes), which differs with conventional feature selection algorithm behaviors.

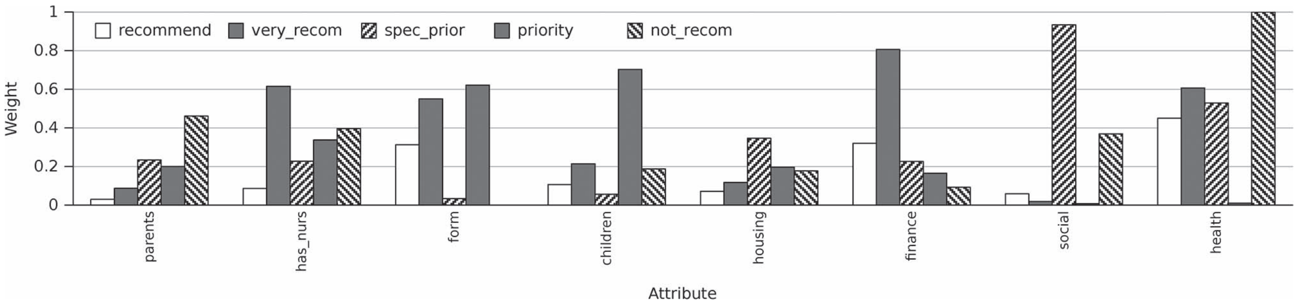


Fig. 8. Nursery weights.

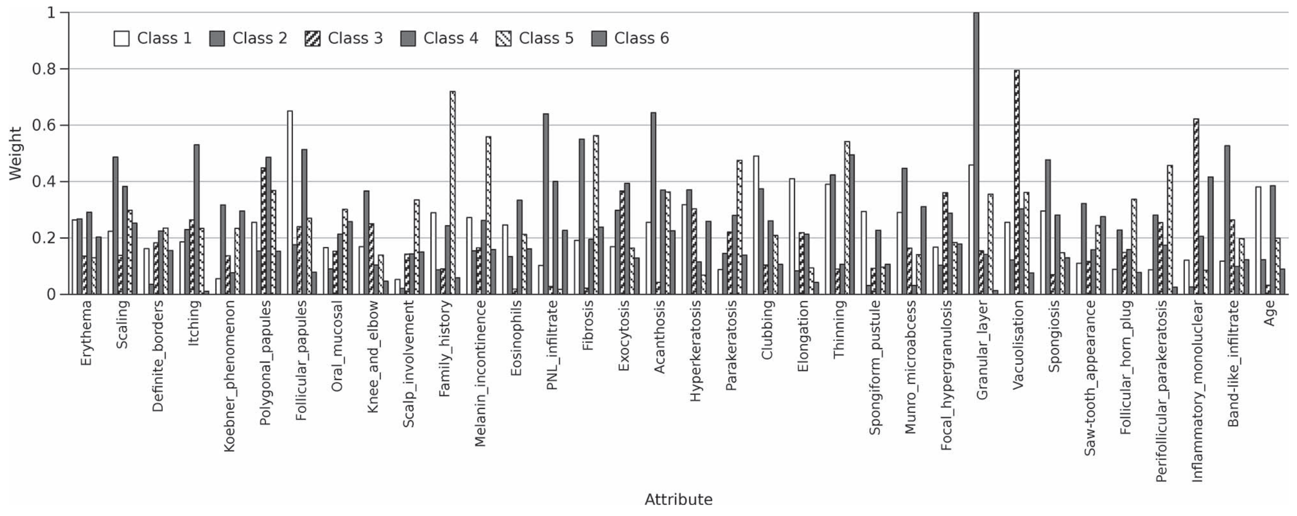


Fig. 9. Dermatology weights.

TABLE XI
TIME PERFORMANCE: DGC+ VS DGC

Data set	Train Instances	DGC Particles	Evaluation Time (ms) DGC+	Evaluation Time (ms) DGC
Banana	4770	16	8.1707	0.0783
Ionosphere	316	308	5.1187	4.8494
Iris	135	50	0.3381	0.1433
Nursery	11664	9141	107.3627	61.6712
Page-blocks	4924	328	31.0672	1.7721
Sonar	187	187	8.0320	8.0131
Thyroid	6480	1179	87.0068	12.2841
Vowel	891	614	7.3176	4.5143

E. Time Performance: DGC+ vs DGC

Even though the experimental results advocate for the higher accuracy of the proposed DGC+ over the previous gravitation method DGC, it is interesting to analyze the time performance of both methods, considering the data reduction of DGC by means of the data particle creation, which transforms the train data into a subset of prototypes to reduce the complexity of the data set. Table XI shows the number of train instances, the number of artificial data particles created by DGC, and the evaluation time of a candidate classifier for both DGC+ and DGC over a subset of representative data sets. Evaluation times for all data sets are shown in the Web site provided in [1].

The computational complexity of the gravitation calculation is similar for both methods. However, in original DGC theory, the data particle is employed to obtain higher classification speed but at the cost of the accuracy, whereas DGC+ uses all train instances for the calculation of the gravitation. Therefore, the higher speed of the evaluation of a DGC classifier will depend on the size of the reduction of the data, which, in turn, depends on the original data distribution. Thus, while some data sets can be reduced to many fewer data particles, such as banana, page-blocks, or thyroid, others cannot be simplified. In this way, it would be recommended to apply a data reduction preprocessing prior to DGC+ classification when the computation time due to the large number of instances exceeds the target time of the researcher.

F. Discussion

This section analyzes the results from the exhaustive experimental study with respect to data problem statistics which can be helpful in showing the proposal's performance, advantages, and disadvantages under specific conditions. The proposed DGC+ is generally better than other methods and achieves the best accuracy, Cohen's kappa rate, AUC, and average and ranking results for the algorithms and data sets used in the experimental study.

On the one hand, regarding standard data performance, the 35 data sets comprise an average number of 12.48 attributes, 1495.65 examples, and 3.74 classes. DGC+ obtains the first and second best results in 22 of the 35 data sets, whose data subset comprises an average number of 10.54 attributes, 1564.04 examples, and 3.22 classes. DGC+ worst results are among the fifth and sixth in 6 of the 35 data sets, whose data subset comprises an average number of 8 attributes, 2032 examples, and 3.33 classes.

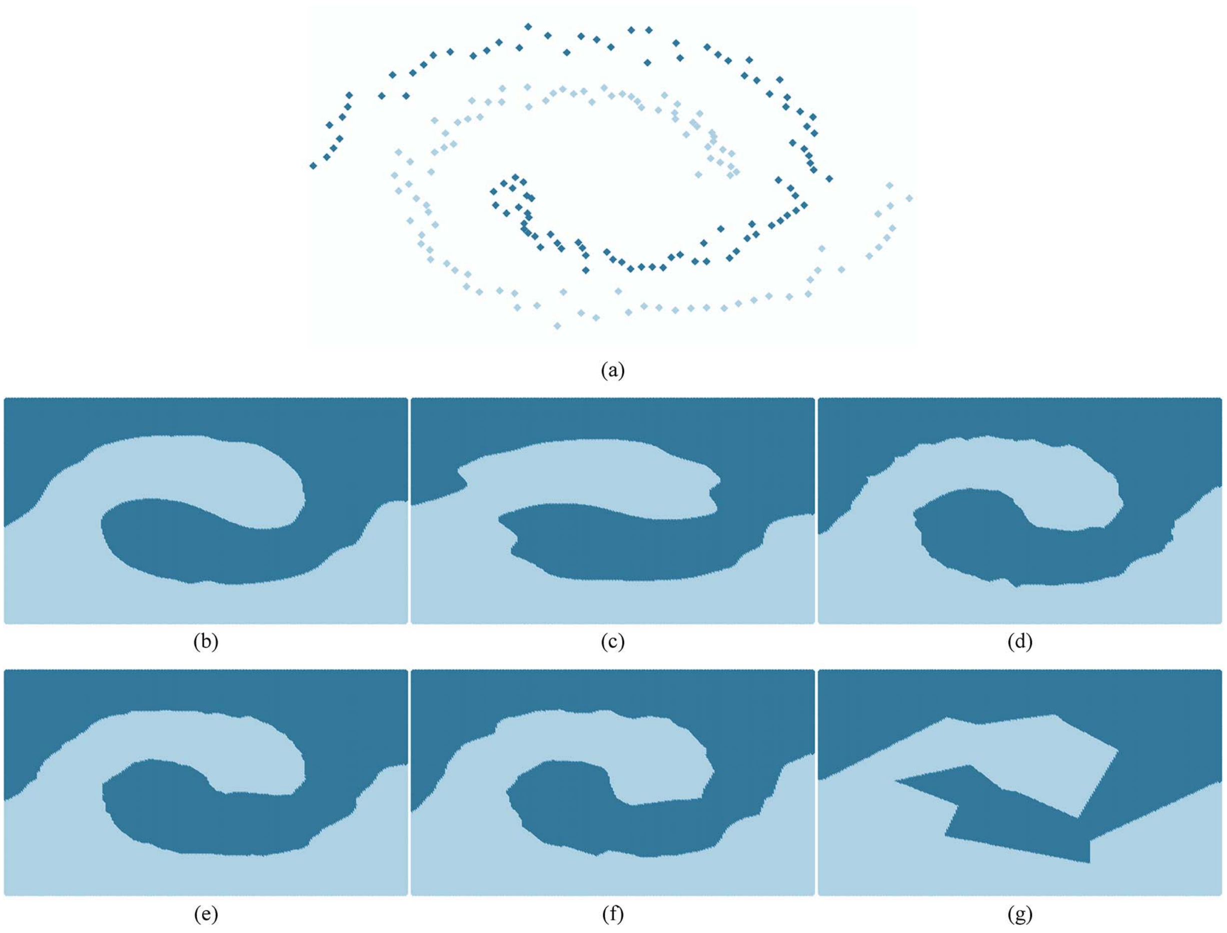


Fig. 10. Two spiral data set predictions. (a) Data set. (b) DGC+. (c) DGC. (d) KNN. (e) KNN-A. (f) DW-KNN. (g) SSMA+SFLSDE.

On the other hand, regarding imbalance data performance, the 44 data sets comprise an average number of 9.13 attributes, 831.5 examples, and an imbalance ratio of 14.55. DGC+ obtains the first and second best results in 26 of the 44 data sets, whose data subset comprises an average number of 9.42 attributes, 869.76 examples, and an imbalance ratio of 15.78. DGC+ worst results are among the sixth, seventh, and eighth in 4 of the 44 data sets, whose data subset comprises an average number of 9.25 attributes, 674.75 examples, and an imbalance ratio of 17.98.

These results do not indicate solid evidences to conclude that the proposal will perform badly in certain classification domains. However, the lack of minority class examples in extremely high imbalance data classification might result to a condition where the minority class examples do not sufficiently represent other minority class examples. On the other hand, attribute-class weight learning has demonstrated to overcome noisy and irrelevant data. Thus, the proposal will perform accurate classification in these domains, better than those from other methods without distance weighting or feature selection.

Finally, it is interesting to show the smoothness of the classification border between data class predictions. Fig. 10 compares the classification performance of the best ranked classification methods used in the experimental study for standard data sets, over the two spiral data sets with Gaussian noise. The proposed DGC+ obtains accurate and smooth border classification. DGC is noticed to suffer from data particle prototype generation, achieving a worse local smoothness. KNN methods provide less smooth predictions than DGC+ by means of sawtooth predictions. SSMA+SFLSDE classifies according to the coarse prototypes generated.

VI. Conclusion

In this paper, a DGC algorithm called DGC+ has been presented. The proposal includes attribute-class weight learning for distance weighting to improve classification results. The weights were optimized by means of the CMA-ES algorithm, which showed to perform an effective learning rate of optimal weights for the different attributes and classes, ignoring noisy attributes and enhancing relevant ones. The effects of gravitation around the instances allowed an accurate classification considering both local and global data information, providing smooth classification and good generalization. Gravitation was successfully adapted to deal with imbalanced data problems. The proposal achieved better classification accuracy, Cohen's kappa rate, and AUC results than other well-known instance-based and imbalanced classification methods. The results were validated using multiple and pairwise nonparametric statistical tests, whose reports support the statistically significant better performance of the proposal.

References

[1] M. Paliwal and U. A. Kumar, "Neural networks and statistical techniques: A review of applications," *Expert Syst. Appl.*, vol. 36, no. 1, pp. 2–17, Jun. 2009.

[2] A. Widodo and B. S. Yang, "Support vector machine in machine condition monitoring and fault diagnosis," *Mech. Syst. Signal Process.*, vol. 21, no. 6, pp. 2560–2574, Aug. 2007.

[3] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Mach. Learn.*, vol. 6, no. 1, pp. 37–66, Jan. 1991.

[4] P. G. Espejo, S. Ventura, and F. Herrera, "A survey on the application of genetic programming to classification," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 40, no. 2, pp. 121–144, Mar. 2010.

[5] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 1, pp. 21–27, Jan. 1967.

[6] I. Kononenko and M. Kukar, *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Cambridge, U.K.: Horwood Publ., 2007.

[7] B. Li, Y. W. Chen, and Y. Q. Chen, "The nearest neighbor algorithm of local probability centers," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 38, no. 1, pp. 141–154, Feb. 2008.

[8] L. Peng, B. Peng, Y. Chen, and A. Abraham, "Data gravitation based classification," *Inf. Sci.*, vol. 179, no. 6, pp. 809–819, Mar. 2009.

[9] C. Wang and Y. Q. Chen, "Improving nearest neighbor classification with simulated gravitational collapse," in *Proc. ICNC*, 2005, vol. 3612, pp. 845–854.

[10] Y. Zong-Chang, "A vector gravitational force model for classification," *Pattern Anal. Appl.*, vol. 11, no. 2, pp. 169–177, May 2008.

[11] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, Jun. 2001.

[12] N. Hansen, "The CMA evolution strategy: A comparing review," in *Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms*, J. A. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, Eds. New York: Springer-Verlag, 2006, pp. 75–102.

[13] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *J. Multiple-Valued Logic Soft Comput.*, vol. 17, pp. 255–287, 2011.

[14] A. Frank and A. Asuncion, UCI machine learning repository, Univ. California, School Inf. Comput. Sci., Irvine, CA. [Online]. Available: http://archive.ics.uci.edu/ml/citation_policy.html

[15] J. Alcalá-Fdez, L. Sánchez, S. García, M. del Jesus, S. Ventura, J. Garrell, J. Otero, C. Romero, J. Bacardit, V. Rivas, J. Fernández, and F. Herrera, "KEEL: A software tool to assess evolutionary algorithms for data mining problems," *Soft Comput.—Fusion Found., Methodol. Appl.*, vol. 13, no. 3, pp. 307–318, Oct. 2009.

[16] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemannr, and I. H. Witten, "The WEKA data mining software: An update," *SIGKDD Explorat.*, vol. 11, no. 1, pp. 10–18, Jun. 2009.

[17] A. Ben-David, "Comparison of classification accuracy using Cohen's weighted kappa," *Expert Syst. Appl.*, vol. 34, no. 2, pp. 825–832, Feb. 2008.

[18] A. Ben-David, "About the relationship between ROC curves and Cohen's kappa," *Eng. Appl. Artif. Intell.*, vol. 21, no. 6, pp. 874–882, Sep. 2008.

[19] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognit.*, vol. 30, no. 7, pp. 1145–1159, Jul. 1997.

[20] J. Huang and C. X. Ling, "Using AUC and accuracy in evaluating learning algorithms," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 3, pp. 299–310, Mar. 2005.

[21] O. J. Dunn, "Multiple comparisons among means," *J. Amer. Stat. Assoc.*, vol. 56, no. 293, pp. 52–64, Mar. 1961.

[22] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometr. Bull.*, vol. 1, no. 6, pp. 80–83, Dec. 1945.

[23] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Inf. Sci.*, vol. 180, no. 10, pp. 2044–2064, May 2010.

[24] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*. London, U.K.: Chapman & Hall, 2007.

[25] S. A. Dudani, "The distance-weighted k-nearest-neighbor rule," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. SMC-6, no. 4, pp. 325–327, Apr. 1976.

[26] Q. Gao and Z. Wang, "Center-based nearest neighbor classifier," *Pattern Recognit.*, vol. 40, no. 1, pp. 346–349, Jan. 2007.

[27] J. Wang, P. Neskovic, and L. N. Cooper, "Improving nearest neighbor rule with a simple adaptive distance measure," *Pattern Recognit. Lett.*, vol. 28, no. 2, pp. 207–213, Jan. 2007.

[28] R. Paredes and E. Vidal, "A class-dependent weighted dissimilarity measure for nearest neighbor classification problems," *Pattern Recognit. Lett.*, vol. 21, no. 12, pp. 1027–1036, Dec. 2000.

[29] R. Paredes and E. Vidal, "Learning weighted metrics to minimize nearest-neighbor classification error," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 7, pp. 1100–1110, Jul. 2006.

[30] S. Garcia, J. Derrac, J. R. Cano, and F. Herrera, "Prototype selection for nearest neighbor classification: Taxonomy and empirical study," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 3, pp. 417–435, Mar. 2012.

[31] I. Triguero, J. Derrac, S. Garcia, and F. Herrera, "A taxonomy and experimental study on prototype generation for nearest neighbor classification," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 1, pp. 86–100, Jan. 2012.

[32] R. Paredes and E. Vidal, "Learning prototypes and distances: A prototype reduction technique based on nearest neighbor error minimization," *Pattern Recognit.*, vol. 39, no. 2, pp. 180–188, Feb. 2006.

[33] C. Zhou and Y. Chen, "Improving nearest neighbor classification with cam weighted distance," *Pattern Recognit.*, vol. 39, no. 4, pp. 635–645, Apr. 2006.

[34] I. Triguero, S. García, and F. Herrera, "Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification," *Pattern Recognit.*, vol. 44, no. 4, pp. 901–916, Apr. 2011.

[35] S. García, J. R. Cano, and F. Herrera, "A memetic algorithm for evolutionary prototype selection: A scaling up approach," *Pattern Recognit.*, vol. 41, no. 8, pp. 2693–2709, Aug. 2008.

[36] F. Neri and V. Tirronen, "Scale factor local search in differential evolution," *Memetic Comput.*, vol. 1, no. 2, pp. 153–171, Jun. 2009.

[37] M. Z. Jahromi, E. Parvinnia, and R. John, "A method of learning weighted similarity function to improve the performance of nearest neighbor," *Inf. Sci.*, vol. 179, no. 17, pp. 2964–2973, Aug. 2009.

[38] W. E. Wright, "Gravitational clustering," *Pattern Recognit.*, vol. 9, no. 3, pp. 151–166, Oct. 1977.

[39] Y. Endo and H. Iwata, "Dynamic clustering based on universal gravitation model," *Model. Decisions Artif. Intell.*, vol. 3558, pp. 183–193, 2005.

[40] B. Yang, L. Peng, Y. Chen, H. Liu, and R. Yuan, "A DGC-based data classification method used for abnormal network intrusion detection," in *Proc. ICONIP*, 2006, vol. 4234, pp. 209–216.

[41] J. Giblin, D. Marolf, and R. Garvey, "Spacetime embedding diagrams for spherically symmetric black holes," *Gen. Relativ. Gravit.*, vol. 36, no. 1, pp. 83–99, Jan. 2004.

[42] L. Al Shalabi, Z. Shaaban, and B. Kasasbeh, "Data mining: A preprocessing engine," *J. Comput. Sci.*, vol. 2, no. 9, pp. 735–739, Sep. 2006.

[43] N. K. Visalakshi and K. Thangavel, "Impact of normalization in distributed k-means clustering," *Int. J. Soft Comput.*, vol. 4, no. 4, pp. 168–172, 2009.

[44] J. Han, *Data Mining: Concepts and Techniques*. San Francisco, CA: Morgan Kaufmann, 2005.

[45] G. V. Lashkia and L. Anthony, "Relevant, irredundant feature selection and noisy example elimination," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 34, no. 2, pp. 888–897, Apr. 2004.

[46] B. Apolloni, S. Bassis, and A. Brega, "Feature selection via Boolean independent component analysis," *Inf. Sci.*, vol. 179, no. 22, pp. 3815–3831, Nov. 2009.

[47] T. N. Lal, O. Chapelle, J. Western, and A. Elisseeff, "Embedded methods," *Stud. Fuzziness Soft Comput.*, vol. 207, no. 1, pp. 137–165, Jan. 2006.

[48] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.

[49] H. Kaizhu, Y. Haiqin, K. Irwinng, and M. R. Lyu, "Imbalanced learning with a biased minimax probability machine," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 36, no. 4, pp. 913–923, Aug. 2006.

[50] R. Akbani, S. Kwek, and N. Japkowicz, "Applying support vector machines to imbalanced datasets," in *Proc. 15th ECML*, 2004, pp. 39–50.

[51] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. 14th Int. Joint Conf. Artif. Intell.*, San Francisco, CA, 1995, vol. 2, pp. 1137–1143.

[52] T. Wiens, B. Dale, M. Boyce, and G. Kershaw, "Three way k-fold cross-validation of resource selection functions," *Ecolog. Model.*, vol. 212, no. 3/4, pp. 244–255, Apr. 2008.

[53] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf. Process. Manag.*, vol. 45, no. 4, pp. 427–437, Jul. 2009.

[54] Q. Gu, L. Zhu, and Z. Cai, "Evaluation measures of the classification performance of imbalanced data sets," *Commun. Comput. Inf. Sci.*, vol. 51, no. 1, pp. 461–471, Oct. 2009.

[55] A. Fernández, S. García, J. Luengo, E. Bernado-Mansilla, and F. Herrera, “Genetics-based machine learning for rule induction: State of the art, taxonomy, and comparative study,” *IEEE Trans. Evol. Comput.*, vol. 14, no. 6, pp. 913–941, Dec. 2010.
[56] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera, “An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes,” *Pattern Recognit.*, vol. 44, no. 8, pp. 1761–1776, Aug. 2011.
[57] A. Fernández, M. J. del Jesus, and F. Herrera, “On the 2-tuples based genetic tuning performance for fuzzy rule based classification systems in imbalanced data-sets,” *Inf. Sci.*, vol. 180, no. 8, pp. 1268–1291, Apr. 2010.
[58] G. Batista, R. Prati, and M. Monard, “A study of the behavior of several methods for balancing machine learning training data,” *SIGKDD Explorat.*, vol. 6, no. 1, pp. 20–29, Jun. 2004.
[59] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic minority over-sampling technique,” *Artif. Intell. Res.*, vol. 16, no. 1, pp. 321–357, Jun. 2002.
[60] I. Tomek, “Two modifications of CNN,” *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 6, no. 11, pp. 769–772, Nov. 1976.
[61] Y. Sun, M. Kamel, A. Wong, and Y. Wang, “Cost-sensitive boosting for classification of imbalanced data,” *Pattern Recognit.*, vol. 40, no. 12, pp. 3358–3378, Dec. 2007.
[62] Z. H. Zhou and X. Y. Liu, “Training cost-sensitive neural networks with methods addressing the class imbalance problem,” *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 1, pp. 63–77, Jan. 2006.
[63] Y. Tang, Y.-Q. Zhang, and N. Chawla, “SVMS modeling for highly imbalanced classification,” *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 39, no. 1, pp. 281–288, Feb. 2009.
[64] K. M. Ting, “An instance-weighting method to induce cost-sensitive trees,” *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 3, pp. 659–665, May/Jun. 2002.
[65] D. R. Wilson and T. R. Martinez, “Reduction techniques for instance-based learning algorithms,” *Mach. Learn.*, vol. 38, no. 3, pp. 257–286, Mar. 2000.
[66] S. Ventura, C. Romero, A. Zafra, J. A. Delgado, and C. Hervás, “JCLEC: A java framework for evolutionary computation,” *Soft Comput.—Fusion Found., Methodol. Appl.*, vol. 12, no. 4, pp. 381–392, Oct. 2007.
[67] S. García, D. Molina, M. Lozano, and F. Herrera, “A study on the use of non-parametric tests for analyzing the evolutionary algorithms’ behaviour: A case study,” *J. Heurist.*, vol. 15, no. 6, pp. 617–644, Dec. 2009.
[68] J. Demšar, “Statistical comparisons of classifiers over multiple data sets,” *J. Mach. Learn. Res.*, vol. 7, no. 1, pp. 1–30, Jan. 2006.
[69] S. García and F. Herrera, “An extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons,” *J. Mach. Learn. Res.*, vol. 9, no. 12, pp. 2677–2694, Dec. 2008.

Alberto Cano (M’10) was born in Cordoba, Spain, in 1987. He received the M.Sc. degree in computer science from the Department of Computer Science and Numerical Analysis, University of Cordoba, Cordoba, where he is currently working toward the Ph.D. degree.

His research is performed as a member of the Knowledge Discovery and Intelligent Systems Research Laboratory and is focused on general purpose GPU systems, parallel computing, soft computing, machine learning, and data mining and its applications.

Amelia Zafra (M’08) was born in Pamplona, Spain, in 1982. She received the B.Sc. and Ph.D. degrees from the University of Granada, Granada, Spain, in 2005 and 2009, respectively.

She is an Associate Professor of computer science and artificial intelligence with the University of Cordoba, Cordoba, Spain. Her teaching is devoted to artificial intelligence, bioinformatics in computer science, and data mining. Her research is performed as a member of the Knowledge Discovery and Intelligent Systems Research Laboratory and is focused on soft computing, machine learning, and data mining and its applications.

Sebastián Ventura (M’07–SM’09) was born in Cordoba, Spain, in 1966. He received the B.Sc. and Ph.D. degrees from the University of Cordoba, Cordoba, Spain, in 1989 and 1996, respectively.

He is an Associate Professor of computer science and artificial intelligence with the University of Cordoba, Cordoba, Spain, where he heads the Knowledge Discovery and Intelligent Systems Research Laboratory. He is the author or coauthor of more than 90 international publications. He has also been engaged in 11 research projects (being the coordinator of two of them) supported by the Spanish and Andalusian Governments and the European Union, concerning several aspects of the area of evolutionary computation, machine learning, and data mining and its applications.