GRASP aplicado al problema de la selección de instancias en KDD

Alfonso Fernández Timón[1], Abraham Duarte Muñoz[2] y Ángel Sánchez Calle[3]

Resumen-- **La calidad de los modelos extraídos de una base de datos y que, representan la información almacenada en la misma, depende en gran medida, de la fase inicial de preprocesamiento de dichos datos. Las principales tareas a realizar en esta fase son dos: por un lado, la limpieza de datos inconsistentes y/o redundantes y, por otro, la reducción del tamaño de la misma (debido a la dificultad que conlleva trabajar con gran cantidad de datos). De las distintas técnicas existentes para esta tarea, hemos centrado nuestra atención en la selección de instancias. El objetivo de este trabajo es conseguir un conjunto reducido de dichos prototipos, representativo de conjunto inicial de datos. Para ello hemos desarrollado un procedimiento basado la metodología GRASP y los resultados han sido comparados con los obtenidos en otros trabajos con el objetivo de comprobar la calidad del método.**

Palabras clave- **KDD, Selección de instancias, GRASP.**

I. INTRODUCCIÓN

A medida que el mundo se va globalizando, la cantidad de datos que maneja la sociedad es cada vez mayor. Conforme aumenta el volumen de estos datos, crece la dificultad para comprenderlos y tomar decisiones a partir de ellos.

La disciplina conocida como KDD (*Knowledge Discovery in Databases*) se encarga, mediante un análisis exhaustivo y detallado, de extraer información valida a partir de estos datos [1]. Dicha información se usa en Minería de Datos (*Data Mining* o DM) para la creación de modelos que se puedan utilizar en diversos campos de la ciencia, la ingeniería o la economía [2].

Los modelos extraídos a partir de la Minería de Datos son muy sensibles a la calidad de los datos almacenados. Esta calidad depende de varios factores, entre los destacan la existencia de datos erróneos o el gran tamaño de la base de datos.

Es por ello que una de las primeras fases del KDD es el preprocesamiento de los datos originales, con objeto de mejorar la calidad de los mismos.

En este trabajo hemos desarrollado un algoritmo, basado en la metodología GRASP, con el fin de aplicarlo al preprocesamiento de conjuntos de datos. El objetivo es obtener subconjuntos de los mismos, que los representen de manera fiable y que faciliten la obtención de información a los procesos siguientes en la Minería de Datos.

Por lo tanto, en las primeras secciones expondremos las diferentes técnicas para el preprocesamiento de datos. A continuación describiremos el algoritmo GRASP diseñado para este trabajo y, por último, aplicaremos el mismo sobre diferentes conjuntos de datos, comparándolos con resultados del estado del arte y extrayendo las pertinentes conclusiones

II. PREPROCESAMIENTO EN KDD

En la Minería de Datos existen diferentes técnicas de preprocesamiento [2,3], entre la que podemos enumerar:

- **Reducción de datos:** obtener un conjunto menor, representativo de los datos originales.
- **Limpieza de datos:** eliminar datos erróneos, incompletos o redundantes.
- **Integración de datos:** fusionar bases de datos concernientes al mismo tipo de problema.
- **Transformación de datos:** realizar modificaciones sintácticas sobre los datos sin que varíe su significado.

En este trabajo nos centraremos en la primera de las técnicas, que será la técnica de preprocesamiento que desarrollaremos.

Los modelos extraídos por las técnicas de *Data Mining* son muy sensibles al tamaño del conjunto de datos, pudiendo llegar a ser difíciles de interpretar si el volumen de datos es demasiado grande. Además, a mayor cantidad de datos, hay que emplear más cantidad de tiempo para generar los modelos. Por consiguiente, una de las principales técnicas de preprocesamiento es la reducción del tamaño del conjunto de datos. Respecto a esta técnica existen varias vías de actuación, entre las que destacamos:

- **Selección de Instancias:** se eligen las instancias más representativas del conjunto.
- **Selección de Características:** se eliminan aquellos atributos menos relevantes para la obtención de los modelos.

[1] Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, a1fonso.fernandez@urjc.es
[2] Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, abraham.duarte@urjc.es
[3] Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, angel.sanchez@urjc.es

- **Discretización de Características: c**onsiste en cuantificar atributos numéricos continuos.
- **Agrupamiento de Datos:** agrupar las instancias en conjuntos en función de su afinidad o relación.
- **Compactación de Datos:** método similar a la selección de instancias, donde las instancias de la tabla final cuentan con un nuevo atributo, representativo del número de instancias desechadas más próximas a ella.

De entre estas técnicas, se ha desarrollado en este trabajo la selección de instancias, que será detallada en la próxima sección.

III. SELECCIÓN DE INSTANCIAS

Una de las principales técnicas de preprocesamiento para la reducción de datos es la selección de instancias o prototipos [3]. Mediante esta técnica se eliminan, del conjunto inicial, además de las instancias erróneas, aquellas menos representativas. Al final se obtiene un conjunto más reducido de las mismas que, repercute en una reducción en la complejidad tanto en tiempo como en memoria utilizada, a la hora de aplicar la Minería de Datos. Existen diferentes técnicas a la hora de llevar a cabo la tarea de eliminar instancias. Entre ellas podemos destacar:

- **Muestreo:** se escoge un subconjunto de instancias a partir del inicial mediante un proceso aleatorio donde cada muestra presenta una probabilidad de ser escogida. Existen diferentes versiones de esta técnica.
- **Selección de prototipos:** esta metodología propone la creación de un clasificador. Dicho clasificador se irá constituyendo con la inclusión de instancias del conjunto inicial que vayan mejorando su capacidad de clasificación (a comprobar para el resto de instancias del conjunto inicial y haciendo uso de la regla del vecino más cercano).
- **Aprendizaje activo:** Se parte de un conjunto de instancias clasificadas (conjunto de entrenamiento), al que se van añadiendo instancias desde un conjunto no clasificado y que mejoran sus prestaciones.

Podemos encontrar en la literatura algoritmos de diferente naturaleza que abordan este problema [4], aunque una de las técnicas más utilizada ha sido la de los algoritmos evolutivos. Uno de los primeros algoritmos evolutivos aplicado a esta problemática fue desarrollado por Kuncheva en 1995 [5]. Sin embargo, entre los que mejores resultados han deportado se encuentran el PBIL [3], el CHC [6], el SSGA [4] y, por encima de todos, el algoritmo memético SSMA [7].

En este trabajo nos centraremos en la selección de prototipos. Para ello partimos de un conjunto de instancias (que denominaremos tabla original), caracterizadas por una serie de atributos comunes, y clasificadas en función del valor de los mismos en diferentes clases. En la Tabla I podemos observar una base de datos con 6 instancias y dos clases, donde cada instancia (fila) tiene 4 atributos (las cuatro primeras columnas) y pertenece a una clase (última columna).

TABLA I
CONJUNTO DE SEIS INSTANCIA CON 4 ATRIBUTOS

Instancia	Atr. 1	Atr. 2	Atr. 3	Atr. 4	Clase
1	4.6	3.4	1.4	0.3	A
2	6.7	4.1	5.2	0.5	B
3	4.5	2.9	1.8	0.3	A
4	8.0	3.4	1.9	0.2	A
5	7.9	4.6	6.2	0.9	B
6	0.6	3.3	5.2	0.1	A

Nuestro objetivo es construir un conjunto más reducido que el original (que llamaremos tabla reducida), formado por las instancias más representativas del mismo y con la capacidad de clasificar aquellas nuevas instancias que se puedan generar.

Para clasificar una determinada instancia en una clase u otra, utilizaremos la regla del vecino más cercano. Es decir, la nueva instancia será encuadrada en la clase de la instancia, perteneciente a la tabla reducida, más cercana de la que se encuentre. En todos los casos, la distancia implementada fue la euclídea para un espacio de n dimensiones (donde n representa el número de atributos de las instancias).

Para calibrar la calidad de la tabla reducida obtenida tras la aplicación del algoritmo nos basaremos en una función de *fitness f*. Dicha función conjugará la capacidad de clasificación de la tabla final como la reducción llevada a cabo por el algoritmo desde el conjunto inicial al tamaño final de la tabla reducida. La función de *fitness f* será:

$$f = \alpha*(PorcAcierto) + (1-\alpha)*(PorcReducción)$$

donde *PorcAcierto* representa el porcentaje de instancias correctamente clasificadas utilizando únicamente las instancias de la tabla reducida. Por otro lado, *PorcReducción* se define como uno menos el cociente entre el tamaño de la tabla reducida y el tamaño de la tabla original. El parámetro α es un valor entre 0 y 1, que ponderará el peso que se le da a cada una de estas cualidades de la tabla reducida. En este trabajo, el valor tomado para α fue siempre de 0.5, lo que significa que se le da la misma importancia a la reducción y al porcentaje de acierto. En función de la aplicación

concreta, se podría dar más énfasis a cualquiera de las dos magnitudes descritas.

El procedimiento aplicado para la obtención de la tabla reducida se puede encuadrar dentro de la metodología GRASP, que pasaremos a desarrollar en la siguiente sección.

IV. ALGORITMO GRASP

GRASP es una técnica de diseño de algoritmos englobada dentro de las metaheurísticas multi-arranque y orientada para problemas de optimización combinatoria [8]. Una versión más evolucionada del método puede consultarse en [9]. Bajo este esquema se encuentran aquellos procedimientos de búsqueda adaptativos, voraces y aleatorizados. Cualquier algoritmo GRASP tiene dos fases bien definidas:

- **Construcción:** esta fase se diseña en base a un algoritmo iterativo en el que, partiendo de una solución vacía y, tras un número de pasos, se va conformando la solución. Dicho proceso es voraz, porque en cada paso intenta incluir en la solución aquella opción que mejor resultado dé. Sin embargo, y para que el proceso no sea determinista, se incluye una componente estocástica. Esta variante aleatoria implica la construcción, en cada iteración, de un conjunto de elementos susceptibles de ser incorporados a la solución que se está construyendo. Dicho conjunto se denomina Lista de Candidatos Restringidos (RCL). En cada iteración, se construye una nueva lista de la cual se elige aleatoriamente uno de los elementos, que pasa a formar parte de la solución que se está construyendo. Cada una de estas incorporaciones influirá tanto en los elementos incluidos en la solución como en los candidatos a formar parte de ella, de ahí que el proceso sea adaptativo.
- **Mejora:** en esta fase se intenta optimizar la solución obtenida en la fase anterior. Para ello se suele recurrir a procedimientos de búsqueda local o a hibridaciones con metodologías más elaboradas.

Por lo tanto, y dentro del marco de una metaheurística GRASP, diseñamos nuestro algoritmo.

El primer paso del algoritmo consiste en inicializar la tabla reducida con los baricentros de las clases existentes en el conjunto inicial de instancias. Para ello agrupamos las instancias según su clase y calculamos una instancia virtual por clase, cuyos atributos son los valores medios de los atributos de todas las instancias de la clase. Sin embargo, estas instancias, que representan los centros de masas de las clases, no son reales. Por lo tanto, incluimos en la tabla reducida (eliminando de la tabla original) y para cada clase, las instancias más próximas a estos baricentros virtuales. De esta forma, la primera tabla reducida tendrá tantas instancias como clases tenga el conjunto inicial.

Con esta tabla reducida y probándola con la tabla original, podemos obtener nuestro primer valor para la función de *fitness*. Esta tabla tendrá el mayor porcentaje posible de reducción ya que tiene que haber como mínimo una instancia por clase siendo, sin embargo, muy bajo el de clasificación.

Para mejorar este porcentaje vamos a ir añadiendo, en cada iteración, instancias a la tabla reducida. Para determinar la instancia a introducir en cada paso seguiremos el siguiente proceso. Calcularemos el *fitness* resultante $f(v)$, para cada una de las instancias v de la tabla original, de su inclusión individual en la tabla reducida. Todas aquellas instancias cuya $f(v)$ sea superior que el *fitness* f_a actual pasarán a formar parte de una lista de candidatos LC:

$$LC = \{v \in Tabla\ Original \mid f(v) > f_a\}$$

Si de esta lista de candidatos LC eligiéramos aquella que más incrementa el *fitness* de la tabla reducida, el proceso sería determinista. Para evitarlo, vamos a generar, a partir de esta Lista de Candidatos, una Lista Restringida de Candidatos LRC. Para ello definiremos como f_{max} y f_{min} los valores máximos y mínimos de fitness $f(v)$ para las instancias de la Lista de Candidatos LC.

$$f_{max} = \max_{v \in LC} f(v)$$

$$f_{min} = \min_{v \in LC} f(v)$$

A partir de aquí construiremos la Lista Restringida de Candidatos, que en términos matemáticos podemos expresar como:

$$LRC = \{u \in LC / f(u) > umbral\}$$

donde el umbral vendrá determinado por la expresión:

$$umbral = f_{min} + \beta * (f_{max} - f_{min})$$

El parámetro β es un valor que cuantifica la aleatoriedad (β=0) o lo voraz (β=1) que es el proceso. En nuestro caso, después de una experimentación previa, elegimos el valor de 0.5. Sólo aquellas instancias que superan este umbral de calidad, forman parte del Lista Restringida de Candidatos.

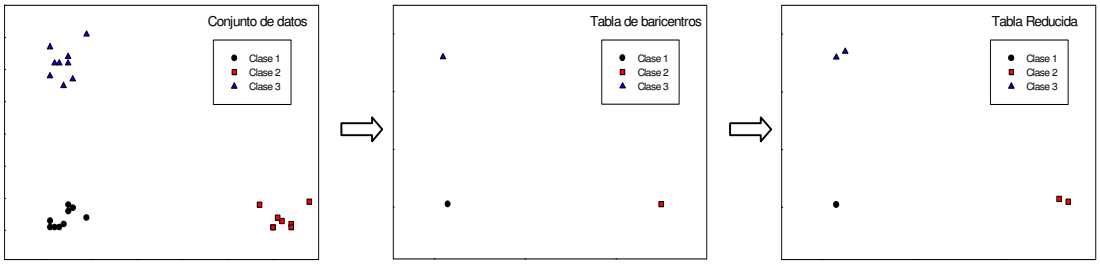

Fig. 1. Descripción gráfica del algoritmo desarrollado en este trabajo.

A continuación, se elige, de manera aleatoria, una de las instancias de esta Lista Restringida de Candidatos, pasando a formar parte de la tabla reducida y siendo eliminada de la tabla original. La inclusión de esta instancia modificará la clasificación de algunas instancias de la tabla original, de ahí el carácter adaptativo del proceso.

Este procedimiento será iterativo, incluyéndose una nueva instancia en cada paso del algoritmo. Dicho proceso acabará en el momento en que no haya candidatos para constituir una nueva Lista de Candidatos ya que su inclusión conduciría a reducir el *fitness* obtenido hasta el momento. La Figura 1 muestra una representación gráfica del proceso compuesto por 27 instancias con 2 atributos y clasificadas en 3 clases.

Una vez obtenida la solución constructiva hay que dar paso a la fase de mejora. Dicha fase está basada en el método constructivo anteriormente presentado, constando de una parte de extracción y otra de inserción de instancias en la tabla reducida. Sobre esta tabla, ya construida en la fase anterior, se seleccionan aquellas instancias que, al ser extraídas de la misma mejoran el *fitness*. Todas ellas se incluyen en una lista de candidatos, de la cual se elige una aleatoriamente. Este proceso se repite (con la construcción de nuevas listas de candidatos en cada iteración) hasta que esta lista no tiene candidatos. A continuación, se reproduce el método de la parte constructiva, añadiendo instancias (vía lista de candidatos) a la tabla reducida que mejoraran su *fitness*. Este proceso se repite hasta que no existen candidatos ni para su exclusión ni para su inserción en la tabla reducida. Ya que el procedimiento desarrollado se enmarca dentro de la metodología GRASP, el proceso de construcción más mejora se repite un número determinado de veces. En nuestro caso, se fijó dicha cantidad en 20 ejecuciones independientes.

V. Resultados Experimentales

Para la realización de este trabajo, se emplearon dos tipos de conjuntos de datos: pequeños y medianos. Todos estos conjuntos fueron tomados del repositorio de la UCI, disponible en la siguiente dirección: http://archive.ics.uci.edu/ml/. En la Tabla II se muestran las características de los conjuntos de pequeño tamaño.

TABLA II
Conjuntos de datos pequeños

NOMBRE	Instancias	Atributos	Clases
Glass	214	9	6
Iris	150	4	3
Lymphography	148	18	4
Monk	432	6	2
Pima	768	8	2
Wine	178	13	3
Wisconsin	683	9	2

Estas bases de datos se refieren a temas tan dispares como *Wisconsin*, donde los datos tomados a los pacientes de un hospital valen para clasificarlos entre personas enfermas y sanas. Por otro lado, la base de datos *Iris* muestra ejemplares de flores de este género, donde los atributos son los tamaños de los pétalos y los sépalos y que son clasificas en tres tipos de subespecies distintas.

Todas estas bases de datos no superan el millar de ejemplares. El otro tipo de conjuntos analizados en este trabajo tienen un tamaño superior, alcanzado las decenas de millar de ejemplares y sus características quedan resumidas en la Tabla III.

TABLA III
Conjuntos de datos Medianos

NOMBRE	Instancias	Atributos	Clases
Pen Based Digits	10992	16	10
Satimage	6435	36	6
Thyroid	7200	21	3

Estas bases de datos tratan temas como *Thyroid*, que determina, en función de los parámetros representados por los atributos, cuando un paciente padece hipotiroidismo. Para cada una de estas bases de datos más numerosas se dispone de un conjunto adicional de instancias, que denominaremos como conjunto *test,* que nos servirá para corroborar la capacidad de clasificación de la tabla reducida

obtenida como resultado de la aplicación del algoritmo GRASP.

El procedimiento anteriormente presentado fue desarrollado en Java, llevándose a cabo las ejecuciones en un ordenador Pentium IV Core 2 Duo 2.27 GHz y 2 GB de memoria RAM.

En similitud con el trabajo con el cual vamos a comparar la calidad de nuestros resultados, para cada uno de los conjuntos se realizaron 3 ejecuciones. Los resultados que se muestran son los valores medios de las tres ejecuciones.

En la Tabla IV aparecen los resultados medios para los conjuntos de tamaño pequeño junto con el tiempo medio por ejecución.

TABLA IV
RESULTADOS OBTENIDOS PARA CONJUNTOS DE TAMAÑO PEQUEÑO

NOMBRE	T (s)	Acierto (%)	Reducción (%)
Glass	1.48	83.03 ± 1.85	94.39 ± 2.14
Iris	0.08	99.31 ± 0.01	96.67 ± 0.01
Lymphography	0.42	87.15 ± 0.38	93.40 ± 0.39
Monk	5.28	99.19 ± 0.85	95.29 ± 0.35
Pima	11.73	78.74 ± 0.70	98.61 ± 0.46
Wine	0.62	94.85 ± 2.10	91.77 ± 2.62
Wisconsin	3.19	97.65 ± 0.01	99.71 ± 0.01

En algunos de los conjuntos hay resultados que se repiten con cierta frecuencia (mismas instancias en la tabla reducida final) dándose casos como los de los conjuntos *Iris* y *Wisconsin* donde las tres soluciones aportadas por el algoritmo han sido idénticas. Además, el diseño del algoritmo hace que los mejores resultados se consigan en las primeras iteraciones. En el caso del conjunto Iris, la tabla reducida final es capaz de clasificar todas las instancias bien a excepción de una. En el caso de *Wisconsin*, la tabla reducida final tiene dos instancias (una por clase) que, curiosamente, no son los baricentros de las mismas. Por otro lado, en uno de los GRASP para el conjunto *Monk*, el porcentaje de acierto fue del 100%. En todos los casos los tiempos se encuentran en el intervalo de centésimas y decenas de segundos.

En la Tabla V aparecen los resultados obtenidos para los tres conjuntos de datos medianos junto con los tiempos medios de ejecución.

TABLA V
RESULTADOS OBTENIDOS PARA CONJUNTOS DE TAMAÑO MEDIANO

NOMBRE	T (s)	Acierto (%)	Reducción (%)
Pen-Based Digits	13146	98.72 ± 0.02	98.71 ± 0.06
Satimage	5015	92.39 ± 0.07	98.94 ± 0.12
Thyroid	528	94.57 ± 0.11	99.80 ± 0.06

Destaca el nivel de reducción tan elevado que produce el algoritmo, donde conjuntos de datos de miles de instancias son representados por una tabla reducida con el 1% de las instancias. En este caso los tiempos de ejecución pueden alcanzar varias horas.

Estos resultados así expuestos no dan una visión completa de la bondad del método hasta que no son comparados con resultados pertenecientes al estado del arte. Es por ello que hemos comparados los mismos con los publicados en el trabajo de García y colaboradores [7], cuyo algoritmo memético SSMA desarrollado fue aplicado sobre estos conjuntos de datos. Los resultados correspondientes a los conjuntos de datos pequeños son mostrados en la Tabla VI. Hemos añadido una última fila a la tabla en la cual se encuentran los valores medios del *fitness* para este tipo de instancia. Para poder hacer una comparativa directa, se resalta en negrita el mejor valor obtenido para cada base de datos. Atendiendo a estos resultados, se puede deducir que los resultados obtenidos por el algoritmo GRASP de este trabajo se encuentran a la altura de los existentes en el estado del arte. No obstante, este comportamiento debería de corroborarse con los resultados obtenidos para conjuntos medianos, ya que la dificultad de problema aumenta a medida que aumenta el tamaño del conjunto.

TABLA VI
COMPARATIVA PARA CONJUNTOS DE TAMAÑO PEQUEÑO

NOMBRE	GRASP	SSMA
Glass	**88.71** ± 0.22	88.35 ± 1.09
Iris	**97.99** ± 0.01	97.00 ± 0.78
Lymphography	**91.77** ± 0.21	75.23 ± 1.47
Monk	97.24 ± 0.39	**97.44** ± 0.91
Pima	88.78 ± 0.11	**89.77** ± 0.48
Wine	93.31 ± 0.33	**97.57** ± 0.38
Wisconsin	**98.68** ± 0.01	98.52 ± 0.11
Valor Medio	93.78 ± 0.09	91.98 ± 0.33

La Tabla VII muestra la comparativa del algoritmo GRASP de este trabajo y el algoritmo memético SSMA para el conjunto de entrenamiento. En este caso también se ha añadido la fila correspondiente a los valores medios.

TABLA VII
COMPARATIVA PARA CONJUNTOS DE TAMAÑO MEDIANO

NOMBRE	GRASP	SSMA
Pen-Based Digits	**99.17** ± 0.02	98.65 ± 0.22
Satimage	**95.66** ± 0.03	95.17 ± 0.29
Thyroid	**97.18** ± 0.03	97.08 ± 0.11
Valor Medio	**97.34** ± 0.02	96.97 ± 0.13

De nuevo, el algoritmo GRASP reproduce unos resultados similares a los obtenidos por el algoritmo memético. Para acabar la etapa de comparativas de nuestros resultados con el algoritmo SSMA, vamos a comparar el porcentaje de clasificación de nuestro algoritmo cuando se aplica sobre un conjunto desconocido de instancias como es el conjunto *test*. Estos resultados aparecen en la Tabla VII, donde también se muestran los resultados obtenidos por el algoritmo SSMA a la hora de clasificar los conjuntos test de cada base de datos.

TABLA VIII
COMPARATIVA PARA CONJUNTOS TEST

NOMBRE	GRASP	SSMA
Pen-Based Digits	96.98 ± 0.05	98.04 ± 0.60
Satimage	89.60 ± 0.15	89.39 ± 0.89
Thyroid	94.13 ± 0.07	93.82 ± 0.54
Valor Medio	93.57 ± 0.06	93.75 ± 0.40

Estos últimos resultados sirven para ratificar de nuevo que el algoritmo aquí desarrollado se encuentra al nivel del algoritmo SSMA.

VI. CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo se ha desarrollado un algoritmo GRASP con objeto de aplicarlo al preprocesamiento de bases de datos. Concretamente su aplicación ha ido orientada hacia el problema de la selección de instancias representativas del conjunto de datos. El algoritmo diseñado ha sido aplicado sobre varias bases de datos de diferentes tamaños y los resultados han sido comparados con los obtenidos por otros algoritmos. La comparativa ha mostrado que los resultados obtenidos por nuestro algoritmo reproducen los disponibles en el estado del arte.

Como futuros trabajos se encuentran el trabajar con conjuntos de mayor tamaño y el aplicar el algoritmo a problemas de Visión Artificial y Biometría.

REFERENCIAS

[1] Alex A. Freitas, Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer (2002).
[2] Jiawei Han and Micheline Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann (2006).
[3] S. Baluja. Population-based incremental learning, Carnegie Mellon Univ. Pittsburgh, PA, CMU-CS-94-163 (1994).
[4] Jose Ramon Cano, Francisco Herrera and Manuel Lozano, Using Evolutionary Algorithms as Instance Selection for Data Reduction in KDD: An Experimental Study, IEEE Transactions on Evolutionary Computation 7 (2003) 561-575.
[5] L.I. Kuncheva. Editing for the k-nearest neighbors rule by a genetic algorithm. Pattern Recognition Letters 16 (1995) 809-814.
[6] L.J. Eshelman, The adaptive search algorithm: How to have safe search when engaging in non-traditional genetic recombination, in *Foundations og Genetic Algorithms-1*, Morgan-Kauffman (1991) 265-283.
[7] Salvador García, Jose Ramon Cano and Francisco Herrera, A memetic algorithm for evolutionary prototype selection: A scaling up approach, Pattern Recognition 41 (2008) 2693-2709.
[8] T. A. Feo and M.G.C. Resende, Greedy adaptive search procedures. Journal of Global Optimization 6 (1995) 109-133.
[9] Mauricio Resende and Celso Ribeiro, Greedy randomized adaptive search procedures. Chapter 8 In: Glover, F., Kochenberger, G. (Eds.), Handbook of Metaheuristics, Kluwer Academic Publishers (2003), 219-249.