



Prácticas de Reconocimiento de Formas

Numerical Cruncher

Analizador Numérico © 1999 Fernando Berzal Galiano

Clasificador lineal: Distancia euclídea
 Construcción del clasificador... 1002 ms
 Evaluación del clasificador... 4656 ms
 Clasificando muestras... 18146 ms

Inicio

galiko

Datos Edición Clasificación Agrupamiento Ayuda

Clasificador lineal: Distancia euclídea

Clasificador construido a partir de 7034 patrones
 Clasificador evaluado utilizando 3651 patrones

Clase	1	2	3	4	5	%
1	332	5	20	143	0	66.39%
2	2	1000	0	0	0	99.80%
3	3	2	506	32	177	70.27%
4	135	5	10	187	1	55.32%
5	2	88	443	10	548	50.22%

Bondad estimada del clasificador: 70.47% (2573/3651)

Ver

IGAL3

Clasificador lineal: Distancia euclídea

Fernando Berzal Galiano

E.T.S. Ingeniería Informática

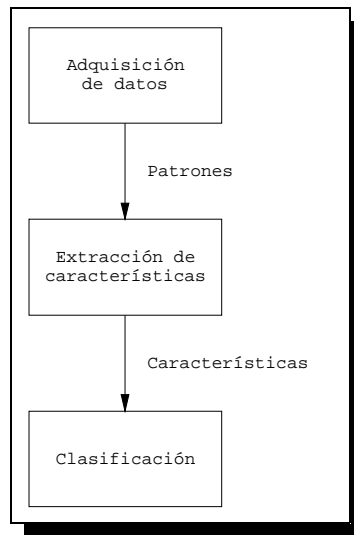
Universidad de Granada

Índice

Introducción	1
Analizador Numérico	3
Ficheros de configuración de datos	4
Secciones comunes	4
Formato ASCII	5
Formato IMAGE	6
Formato JDBC	7
1. Clasificadores paramétricos	8
Experimentación	9
2. Métodos de edición y condensado	11
Experimentación	13
3. Métodos de aprendizaje adaptativo	20
LVQ [Linear Vector Quantization]	20
DSM [Decision Surface Mapping]	22
Experimentación	23
4. Métodos de agrupamiento	28
Método adaptativo	33
Algoritmo de Batchelor y Wilkins (algoritmo de máxima distancia)	35
Algoritmo de las K medias	36
Algoritmo GRASP	39
Algoritmo de agrupamiento secuencial	44
Algoritmo ISODATA	45
Métodos basados en grafos: Matriz de similitud	46
Implementación	48

Introducción

El Reconocimiento de Formas [Pattern Recognition] se utiliza en la transferencia de conocimientos de una o más fuentes (libros, manuales, expertos...) a un formato que permita su tratamiento informático. La configuración típica de un sistema de reconocimiento de formas viene recogida en el siguiente diagrama:



Sistema de reconocimiento de formas

En Reconocimiento de Formas, un **patrón** X es una variable aleatoria n -dimensional. Está compuesto por n características x_i tales que $x_i \in G_i$ para $i = 1..n$. Un patrón se representa como un punto en el espacio de patrones P , el conjunto de todos los valores posibles que puede tomar un patrón X :

$$P = \times_{i=1}^n G_i$$

Los objetivos básicos del **módulo de extracción de características** son reducir la dimensionalidad del problema (selección de características) y cambiar el espacio de representación mediante alguna transformación que resalte alguna propiedad interesante de los datos que facilite la clasificación de éstos (extracción de características).

Si suponemos que todos los patrones a reconocer son elementos potenciales de J clases distintas denotadas ω_j , llamaremos $\Omega = \{ \omega_j \mid 1 \leq j \leq J \}$ al conjunto de las clases informacionales. En ocasiones extenderemos Ω con una clase de rechazo ω_0 a la que asignaremos todos los patrones para los que no se tiene una certeza aceptable de ser clasificados correctamente en alguna de las clases de Ω . De este modo, $\Omega^* = \{ \omega_0 \} \cup \Omega$ es el conjunto extendido de clases informacionales. Un **clasificador** o regla de clasificación es una función $d: P \rightarrow \Omega$ definida sobre el conjunto de patrones tal que para todo patrón X , $d(X) \in \Omega^*$.

Las prácticas realizadas para la asignatura han sido implementadas en Java, un lenguaje de programación derivado del C++ que fue desarrollado por Sun Microsystems (empresa líder en servidores para Internet) en un intento de resolver los problemas que se le plantean a los programadores por la proliferación de sistemas incompatibles. Hay versiones distintas sobre el origen, concepción y desarrollo de Java, desde la que dice que éste fue un proyecto que estuvo durante mucho tiempo por distintos departamentos de Sun sin que nadie le prestara atención hasta la más difundida, que presenta a Java como lenguaje de pequeños electrodomésticos.

Hace algunos años, Sun Microsystems decidió intentar introducirse en el mercado de la electrónica de consumo y desarrollar programas para pequeños dispositivos electrónicos. Tras unos comienzos dudosos, Sun decidió crear una filial, denominada FirstPerson Inc.. El mercado inicialmente previsto para los programas de FirstPerson eran los equipos domésticos: microondas, tostadoras y, fundamentalmente, televisores interactivos. En este mercado, dada la falta de pericia de los usuarios para el manejo de estos dispositivos, se requerían unos interfaces mucho más cómodos e intuitivos que los sistemas de ventanas del momento. También era importante la fiabilidad del código y la facilidad de desarrollo.

James Gosling, el miembro del equipo con más experiencia en lenguajes de programación, decidió que las ventajas aportadas por la eficiencia de C++ no compensaban el gran coste de pruebas y depuración. Gosling había estado trabajando en su tiempo libre en un lenguaje de programación que él había llamado Oak, el cual, aún partiendo de la sintaxis de C++, intentaba remediar las deficiencias que iba observando.

El primer proyecto en que se aplicó este lenguaje recibió el nombre de proyecto Green y consistía en un sistema de control completo de los aparatos electrónicos y el entorno de un hogar. Para ello se construyó un ordenador experimental denominado *7 (Star Seven). El sistema presentaba una interfaz basada en la representación de la casa de forma animada y el control se llevaba a cabo mediante una pantalla sensible al tacto. En el sistema aparecía Duke, la actual mascota de Java. Posteriormente se aplicó a otro proyecto denominado VOD (Video On Demand) en el que se empleaba como interfaz para la televisión interactiva. Ninguno de estos proyectos se convirtió nunca en un sistema comercial, pero fueron desarrollados enteramente en Java. Una vez que en Sun se dieron cuenta de que a corto plazo la televisión interactiva no iba a ser un gran éxito, urgieron a FirstPerson a desarrollar con rapidez nuevas estrategias que produjeran beneficios. No lo consiguieron y FirstPerson cerró en la primavera de 1994.

Pese a lo que parecía ya un olvido definitivo, Bill Joy, cofundador de Sun y uno de los desarrolladores principales del Unix de Berkeley, juzgó que Internet podría llegar a ser el terreno adecuado para disputar a Microsoft su primacía casi absoluta en el terreno del software, y vio en Oak el instrumento idóneo para llevar a cabo estos planes. Tras un cambio de nombre y modificaciones de diseño, el lenguaje Java fue presentado en sociedad en agosto de 1995.

Analizador Numérico (o *Numerical Cruncher*)

Éste es el nombre con el que nos referiremos a la interfaz gráfica integrada que permite probar muchos de los algoritmos de clasificación, edición, condensado y agrupamiento estudiados a lo largo de la asignatura.

Para poder ejecutar el programa se requiere disponer del entorno de ejecución de Java 1.1 y de las Java Foundation Classes 1.1 (también conocidas como SWING), que en la versión 1.2 de Java ya forman parte del estándar. Desde la línea de comandos hemos de escribir (asegurándonos de que el intérprete Java pueda encontrar las clases necesarias fijando la variable de entorno CLASSPATH):

```
java rf.gui.NumericalCruncher [spanish|english] fichero_CFG*
```

El primero de los parámetros es opcional y sirve para especificar el idioma deseado. A continuación se pueden incluir tantos ficheros CFG como deseemos. Aunque no especifiquemos un fichero CFG al inicio, siempre podremos seleccionar el que deseemos a partir de la ventana principal de la aplicación.

Además de aplicación, NC (abreviatura correspondiente a la denominación del Analizador Numérico en inglés) también es un applet, por lo que podremos ejecutarlo desde un navegador de Internet siempre que éste permita applets escritos en Java 1.1 (vg: HotJava) o dispongamos del plugin correspondiente para nuestro navegador estándar (MS Internet Explorer o Netscape Navigator).

También existe la posibilidad de ejecutar cualquiera de los algoritmos implementados directamente desde la línea de comandos tal como se explicará para cada método concreto.

Una de las principales características del programa (posiblemente la única que lo hace interesante) es el hecho de poder manejar datos en distintos formatos: ficheros ASCII estándar, conjuntos de imágenes RAW (8bpp) o bases de datos accesibles a través de JDBC [*Java DataBase Connectivity*].

Ficheros de configuración de datos

Los ficheros de configuración CFG sirven para especificar toda aquella información acerca de los datos (metainformación para ser precisos) que permita al Analizador Numérico acceder e interpretar adecuadamente los mismos.

Estos ficheros poseen una sintaxis similar a la de los ficheros .INI de Windows. Cada sección viene encabezada por una línea que puede comenzar con un corchete `[` como los ficheros INI de Windows o con una almohadilla `#` (algo bastante más común en otros sistemas operativos, p.ej. Linux).

Las distintas secciones de un fichero de configuración pueden estar desordenadas, ya que antes de interpretarse se carga completo en memoria. Además, se pueden incluir tantos comentarios como se desee en líneas que comiencen con un punto y coma `(;)`.

Una vez comentada la sintaxis de estos ficheros podemos pasar a analizar las secciones que nos encontraremos. Algunas de ellas son comunes para todos los formatos de datos mientras que otras son específicas de un determinado tipo.

Secciones comunes

La sección `[DESCRIPTION]` puede incluir una descripción general de la base de datos a la que hace referencia el fichero de configuración. Esta sección puede contener varias líneas, la primera de las cuales será utilizada por el Analizador Numérico como título de la ventana principal (aquella que contiene el menú con todos los algoritmos aplicables).

La sección `[FORMAT]` indica el tipo de base de datos a la que se accede. Los valores permitidos para esta sección son *ASCII* (para ficheros ASCII como los empleados en C4.5 o LVQ_PAK), *IMAGE* (si los datos vienen en un conjunto de imágenes RAW) y *JDBC* (cuando se ha de acceder a los datos estableciendo una conexión remota con un servidor de bases de datos).

La sección `[ATTRIBUTES]` debe contener una lista de los distintos componentes de los patrones. Se ha de especificar un identificador por línea. Este identificador ha de coincidir con el nombre del fichero que contiene la imagen correspondiente al atributo cuando los datos están en formato *IMAGE*. En el caso de que los datos estén almacenados en una base de datos, los identificadores deben coincidir con campos de las tablas almacenadas en el servidor.

La sección `[CLASSIFIER]` incluye el identificador del atributo mediante el cual se clasifican los patrones. Esta sección es opcional. Obviamente, si no se incluye, no se podrán aplicar algoritmos de clasificación, edición y condensado al conjunto de datos sobre el que estemos trabajando.

Por último, la sección `[CLASSES]` sirve para enumerar las distintas clases en que se agrupan los patrones. Igual que la sección anterior, también es opcional.

Formato ASCII

Para este formato se debe especificar, al menos, el fichero que contiene los datos en la sección *[DATA FILE]*. Internamente el programa dividirá aleatoriamente estos datos en dos subconjuntos, un conjunto de aprendizaje y un conjunto de prueba aproximadamente formado por el 30% de las muestras disponibles.

Existe la posibilidad de fijar de antemano los conjuntos de entrenamiento y prueba utilizando las secciones *[LEARNING FILE]* y *[TEST FILE]*, que deben contener los nombres de los ficheros donde se incluyan los datos de aprendizaje y los datos de prueba, respectivamente.

Ejemplo: IRIS

[Description]

Iris Plants Database

```
; --- This is perhaps the best known database to be found in the pattern
;      recognition literature.  Fisher's paper is a classic in the field
;      and is referenced frequently to this day.  (See Duda & Hart, for
;      example.)  The data set contains 3 classes of 50 instances each,
;      where each class refers to a type of iris plant.  One class is
;      linearly separable from the other 2; the latter are NOT linearly
;      separable from each other.
; --- Predicted attribute: class of iris plant.
; --- This is an exceedingly simple domain.
```

```
; Number of Instances: 150 (50 in each of three classes)
```

```
; Number of Attributes: 4 numeric, predictive attributes and the class
```

[Format]

ASCII

[Attributes]

sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)

[Classes]

Iris-setosa
Iris-versicolor
Iris-virginica

[Data file]

iris.dat

Formato IMAGE

Cuando los datos se encuentran almacenados como imágenes en formato RAW, se han de indicar obligatoriamente las dimensiones de las imágenes haciendo uso de las secciones habilitadas para ello: *[WIDTH]* y *[HEIGHT]*.

Además se han de especificar los ficheros (también imágenes en el mismo formato) en los que se incluyen las etiquetas de los patrones: *[TRAINING FILE]* para la imagen que contiene todas los patrones etiquetados, *[LEARNING FILE]* para los patrones etiquetados utilizados en la fase de aprendizaje y *[TEST FILE]* para los empleados en la fase de evaluación de los clasificadores construidos.

Ejemplo: IGALIKO

[Description]

Igaliko
Satellite Images

[Format]

Image

[WIDTH]

256

[HEIGHT]

256

[ATTRIBUTES]

IGAL1
IGAL2
IGAL3
IGAL4

[CLASSES]

1
2
3
4
5

[TRAINING FILE]

IGAL_TRA

[LEARNING FILE]

IGAL_LRN

[TEST FILE]

IGAL_TST

Formato JDBC

Cuando para acceder a los datos se ha de establecer una conexión JDBC se han de especificar los siguientes parámetros

- ▶ **[JDBC DRIVER]:** El controlador JDBC del servidor al que estemos accediendo.
- ▶ **[URL]:** La dirección de la base de datos a la que se accede. Esta dirección tendrá generalmente el formato **jdbc:protocolo:subprotocolo:@servidor**
- ▶ **[USER]:** El identificador del usuario en el servidor.
- ▶ **[PASSWORD]:** La clave del usuario en el servidor.
- ▶ **[TABLE]:** El identificador de la tabla que contiene los datos. Recordemos que en **[ATTRIBUTES]** y **[CLASSIFIER]** se han de listar los campos de la tabla que se emplearán en el análisis de los datos.

Ejemplos

	<i>Personal Oracle Lite</i>	<i>Oracle 8 @ FrontDB</i>
[Format]	JDBC	JDBC
[URL]	jdbc:POLite:POLite	jdbc:oracle:oci8:@FRONTDB
[JDBC Driver]	oracle.pol.poljdbc.POLJDBCdriver	oracle.jdbc.driver.OracleDriver
[User]	system	ops\$fberzal
[Password]	manager	*****
[Table]	C45	C45
[Attributes]	Humidity Temperature	Humidity Temperature
[Classifier]	Play	Class
[Classes]	yes no	P N

1. Clasificadores paramétricos

Se han implementado los distintos tipos de clasificadores paramétricos que aparecen en la sección 2.8 del libro de Duda y Hart “*Pattern Classification and Scene Analysis*”, así como el método de los paralelepípedos (un sencillo clasificador paramétrico). Además se ha realizado un clasificador euclídeo en el cual las distancias se normalizan previamente (para evitar que aquellos atributos cuyos rangos sean más amplios no influyan más que los demás al realizar una clasificación).

Todos los clasificadores paramétricos implementados se encuentran dentro del paquete `rf.classification.parametric` y no son más que una especialización de la clase abstracta `rf.classification.Classifier`.

En cada clasificador se halla definida una función `main` para que pueda probarse desde la línea de comandos. La sintaxis para cada uno de ellos viene recogida a continuación:

Clasificador lineal tipo 1: Distancia euclídea

```
java rf.classification.parametric.EuclideanClassifier  
    <fichero_cfg> [apriori]
```

donde `<fichero_cfg>` referencia al fichero de configuración de los datos y `apriori` se emplea para indicar si se utilizan las probabilidades a priori de cada clase dentro del conjunto de entrenamiento.

Clasificador lineal tipo 1 modificado: Distancia euclídea normalizada

```
java rf.classification.parametric.NormalizedEuclideanClassifier  
    <fichero_cfg> [apriori]
```

donde `<fichero_cfg>` referencia al fichero de configuración de los datos y `apriori` se emplea para indicar si se utilizan las probabilidades a priori de cada clase dentro del conjunto de entrenamiento para clasificar los patrones del conjunto de prueba.

Clasificador lineal tipo 2: Distancia de Mahalanobis

```
java rf.classification.parametric.MahalanobisClassifier  
    <fichero_cfg> [apriori]
```

donde `<fichero_cfg>` referencia al fichero de configuración de los datos y `apriori` se emplea para indicar si se utilizan las probabilidades a priori de cada clase dentro del conjunto de entrenamiento a la hora de clasificar.

Clasificador cuadrático

```
java rf.classification.parametric.QuadraticClassifier  
    <fichero_cfg>
```

donde <fichero_cfg> referencia al fichero de configuración de los datos.

Método de los paralelepípedos

```
java rf.classification.parametric.ParallelepipedClassifier  
    <fichero_cfg>
```

donde <fichero_cfg> nuevamente referencia al fichero de configuración de los datos.

Los cálculos matriciales requeridos por alguno de estos clasificadores se realizan utilizando la clase `Matrix` incluida en el paquete `fbg.util.math`. Esta clase recoge un conjunto de rutina traducidas de una librería de cálculo matricial escrita en C hace algún tiempo.

Todos los clasificadores, al ser llamados desde la línea de comandos, muestran en `stderr` la matriz de contingencias resultado de la evaluación del clasificador con la bondad estimada del mismo (global y por clases). Por supuesto, también se pueden construir estos clasificadores desde el Analizador Numérico, la interfaz gráfica integrada que recoge todos los algoritmos implementados, sin más que ir al submenú “*Clasificación ⇒ Modelos Paramétricos*”. En este caso, cuando los datos sean de tipo IMAGE existe además la posibilidad de visualizar el comportamiento de cada clasificador con los patrones no etiquetados.

<i>Bondad estimada de cada clasificador paramétrico para los datos de GALAXY e IGALIKO</i>		
<i>Clasificador</i>	<i>GALAXY</i>	<i>IGALIKO</i>
<i>Lineal tipo 1*</i>	97.55 %	70.47 %
<i>Lineal tipo 1</i>	95.30 %	65.68 %
<i>Lineal tipo 1* normalizado</i>	97.55 %	73.62 %
<i>Lineal tipo 1 normalizado</i>	95.30 %	67.24 %
<i>Lineal tipo 2*</i>	95.10 %	75.10 %
<i>Lineal tipo 2</i>	91.83 %	71.56 %
<i>Cuadrático</i>	97.95 %	79.84 %
<i>Método de los paralelepípedos</i>	97.55 %	73.18 %

* Sin usar las probabilidades a priori de cada clase en el conjunto de entrenamiento

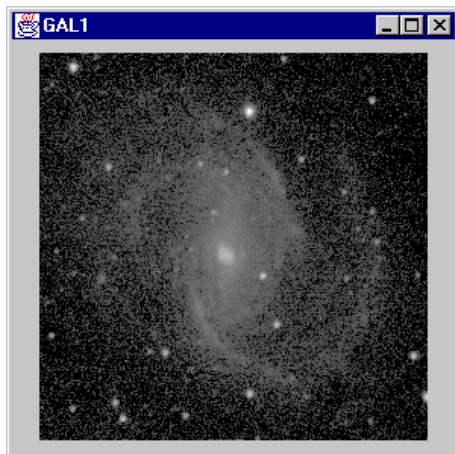
En la tabla de la página anterior se puede apreciar cómo el clasificador cuadrático se comporta mejor que cualquiera de los clasificadores lineales. La dimensionalidad de los datos en estos ejemplos es baja (tres bandas de la galaxia y cuatro de Igaliko), por lo cual el fenómeno de Hughes no hace aparición aquí.

El ADC [Análisis Discriminante Cuadrático] requiere muchas más muestras de entrenamiento que el ADL [Análisis Discriminante Lineal] para obtener resultados similares ya que es más sensible al número de muestras requeridas.

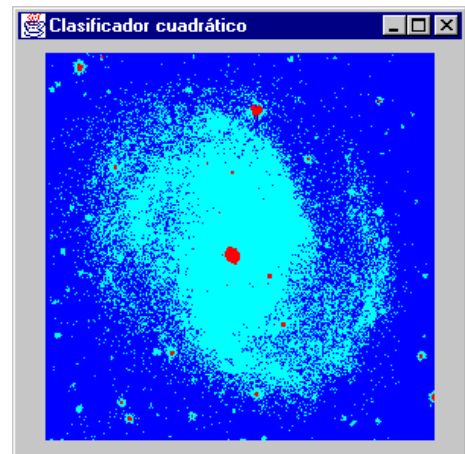
Por ejemplo, para el conjunto de datos de la ionosfera [John Hopkins University Ionosphere Database], el ADC ni siquiera se puede aplicar. No disponemos de suficientes muestras para estimar la matriz de covarianza de los datos adecuadamente (de hecho, ni siquiera podemos calcularle su inversa). En esta base de datos disponemos de 34 atributos y sólo 351 patrones.

Aunque en teoría el error de Bayes decrece conforme la dimensionalidad de los datos se incrementa, en la práctica disponemos de un conjunto fijo y finito de muestras para construir el clasificador (los estimadores están sesgados por las muestras disponibles). La bondad conseguida con un clasificador aumenta con la dimensionalidad de los datos hasta cierto punto, a partir del cual decrece conforme se incorporan nuevas variables (fenómeno de Hughes).

El problema anterior podría solucionarse consiguiendo más muestras de entrenamiento (lo cual no suele ser posible) o eligiendo un subespacio del espacio de patrones (usando técnicas de selección de características).



Una de las bandas de la galaxia en espiral



Resultado obtenido con el clasificador cuadrático

2. Métodos de edición y condensado

Para estudiar el comportamiento de los distintos métodos de edición y condensado se han implementado en el paquete `rf.classification.nn` los clasificadores 1-NN y k-NN, que especializan la clase abstracta `rf.classification.Classifier`. Desde la línea de comandos se puede comprobar su funcionamiento:

Clasificador 1-NN

```
java rf.classification.nn.NNClassifier <fichero_cfg>
```

donde `<fichero_cfg>` referencia al fichero de configuración de los datos.

Clasificador k-NN

```
java rf.classification.nn.kNNClassifier  
  <fichero_cfg> <k> [dataset <fichero_dat>] [multiedit <m> <I>] [Hart]
```

donde `<fichero_cfg>` referencia al fichero de configuración de los datos y `<k>` indica el número de vecinos considerados en la clasificación. Con `dataset <fichero_dat>` se puede especificar el conjunto de entrenamiento que se utilizará (previamente editado y, tal vez, condensado). Con `multiedit <m> <I>` se puede hacer que el conjunto de entrenamiento sea editado antes de proceder a la clasificación del conjunto de prueba. Finalmente, con `Hart` se puede indicar que se aplique el algoritmo de condensado de Hart al conjunto de entrenamiento.

En total se han implementado tres algoritmos de edición: la edición de Wilson, la edición por particiones y el MultiEdit. Así mismo, también se ha codificado en Java el algoritmo de condensado de Hart. El código de estos cuatro métodos de edición y condensado se encuentra en el paquete Java `rf.edit`. La sintaxis requerida desde la línea de comandos para llamar aplicar estos algoritmos aparece a continuación:

EW: Edición de Wilson

```
java rf.edit.WilsonEdition <fichero_cfg> <k>
```

donde `<fichero_cfg>` referencia al fichero de configuración de los datos y `<k>` es el número de vecinos considerado por este algoritmo de edición.

```
java rf.edit.PartitionEdition <fichero_cfg> <m> <k>
```

donde <fichero_cfg> referencia al fichero de configuración de los datos, <m> es el número de bloques en que se dividen los patrones y <k> es el número de vecinos considerado por este algoritmo de edición.

MultiEdit: Multiedición

```
java rf.edit.MultiEdit <fichero_cfg> <m> <I> [<fichero_dat>]
```

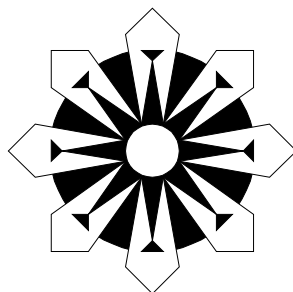
donde <fichero_cfg> referencia al fichero de configuración de los datos, <m> es el número de bloques en que se dividen los patrones, <I> es el número de iteraciones sin descartes utilizado como criterio de finalización y, opcionalmente, <fichero_dat> es el nombre del fichero donde se encuentra el conjunto de entrenamiento inicial (que puede haber sido editado ya). En el fichero `multiedit.dat` se almacena el conjunto editado.

Algoritmo de condensado de Hart

```
java rf.edit.Hart <fichero_cfg> [<fichero_dat>]
```

donde <fichero_cfg> referencia al fichero de configuración de los datos y, opcionalmente, <fichero_dat> es el nombre del fichero de entrada donde hallar el conjunto de entrenamiento inicial (posiblemente editado). En el fichero de salida `hart.dat` es donde se almacenará el conjunto condensado de datos.

Como es obvio, cualquiera de estos algoritmos puede aplicarse desde dentro del Analizador Numérico. El submenú “Edición” recoge los tres algoritmos de edición y el submenú “Clasificación \Rightarrow Vecino más cercano” permite aplicar los clasificadores 1-NN y k-NN.



Experimentación:

k-NN

Influencia del valor de k sobre la bondad de la clasificación k-NN

Para estudiar la influencia del valor de k en la clasificación k -NN se utilizarán los valores 1, 3, 5, 7, 9, 11, 13 y 15. No tiene sentido probar con valores pares de k porque el error asociado a la regla k -NN es el mismo para $2N$ y $2N-1$.

<i>Influencia del valor de k en la clasificación k-NN</i>		
<i>Valor de k</i>	<i>GALAXY</i>	<i>IGALIKO</i>
<i>1</i>	98.36 %	72.39 %
<i>3</i>	98.57 %	75.45 %
<i>5</i>	98.77 %	76.00 %
<i>7</i>	98.57 %	76.82 %
<i>9</i>	98.36 %	77.37 %
<i>11</i>	98.36 %	77.75 %
<i>13</i>	98.36 %	78.11 %
<i>15</i>	98.36 %	78.55 %

Conforme aumentamos el valor de k se reduce el error si disponemos de un número suficiente de muestras de entrenamiento (como sucede en Igaliko), siempre que éstas estén correctamente etiquetadas. Por otro lado, cuantas más muestras tengamos, más tiempo se tardará en realizar la clasificación.

A continuación se muestra una tabla de tiempos que demuestra las mejoras que supone la utilización de un compilador JIT [*Just In Time*] en el Java Development Kit 1.2 frente a la interpretación de los bytecodes realizada en el JDK 1.1:

Tiempos requeridos por los clasificadores k-NN		
Base de datos	JDK1.1	JDK1.2
IRIS	De 0.08 a 0.8 segundos	En torno a 0.1 segundos
GALAXY	De 20 a 30 segundos	Unos 2 segundos
IGALIKO	Algo menos de 8 minutos	Menos de 2 minutos

Los métodos de edición aparecen para solucionar el problema de los patrones mal etiquetados, que intentan eliminar.

Los métodos de condensado procurar reducir el número de muestras sin afectar a la calidad del clasificador para reducir la complejidad computacional del problema.

MULTIEDIT

Influencia del parámetro m sobre el tiempo empleado en la edición y en la bondad de la clasificación 1-NN usando como referencia el conjunto de datos editado

Para analizar la influencia de m se emplearán los valores 3, 4, 5, 6 y 7 y se realizarán tres experimentos por cada valor de m . En todos los casos, el valor del parámetro I será igual a 2.

MULTIEDIT <i>Influencia del parámetro m en el número final de muestras</i>		
Número de particiones	GALAXY 1027 muestras	IGALIKO 7034 muestras
$m = 3$	1010, 1008, 1006	4457, 4511, 4396
$m = 4$	1006, 996, 1003	4349, 4328, 4334
$m = 5$	995, 1004, 991	4236, 4273, 4218
$m = 6$	1000, 993, 997	4147, 4100, 4163
$m = 7$	988, 986, 979	4107, 4079, 4112

Obviamente, cuanto mayor sea m , con más facilidad se descartarán muestras. Además, se consumirá menos tiempo de CPU:

<p style="text-align: center;">MULTIEDIT <i>Influencia del parámetro m en el tiempo de ejecución (hh:mm:ss)</i></p>		
<i>Particiones</i>	<i>GALAXY</i>	<i>IGALIKO</i>
$m = 3$	00:00:30 00:00:42 00:01:00	02:14:04 01:18:47 01:57:15
$m = 4$	00:00:41 00:00:46 00:00:41	01:17:18 01:00:22 01:15:09
$m = 5$	00:00:47 00:00:26 00:00:29	00:47:49 00:58:41 01:29:47
$m = 6$	00:00:19 00:00:46 00:00:31	00:52:57 00:43:19 00:44:08
$m = 7$	00:00:37 00:00:39 00:00:39	01:16:03 00:48:44 00:30:37

Los tiempos muestran lo que parecía lógico suponer en un principio. Cuanto mayor es el valor de m , más particiones se realizan y menos muestras comprenderá cada una de ellas. Al realizar la clasificación k-NN, el número de muestras en el que tendremos que buscar los vecinos más cercanos será menor y, por tanto, el tiempo total de CPU requerido disminuirá (si bien es cierto que probablemente realicemos un mayor número de iteraciones al ir descartando las muestras con mayor facilidad).

Nota: Los tiempos de la tabla de arriba corresponden al JDK1.1 ejecutándose sobre un Pentium 166MHz con 32MB de EDO RAM de 60 ns. Con una versión más reciente del JDK (la 1.2), que utiliza un compilador JIT, se consiguen tiempos más parecidos a los requeridos por un lenguaje compilado como C:

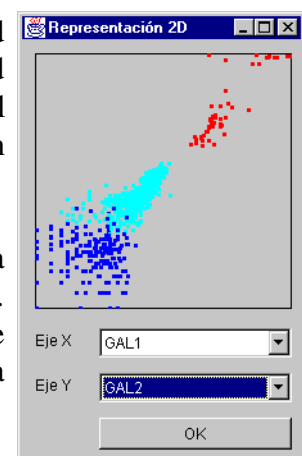
Una característica no deseable del Multiedit es la desaparición de clases que se puede producir debida a la existencia de agrupamientos muy dispersos (varianza elevada) o agrupamientos formados por muestras de distintas clases (cuando hay más etiquetas que clases diferenciables con las características dadas). Por ejemplo, en IRIS es bastante probable que se pierda alguna clase (en este caso, porque hay pocas muestras disponibles de cada clase y al realizar m particiones alguna de ellas podría no contener patrones de todas las clases).

<i>Clasificación 1-NN</i>		
k	<i>GALAXY</i>	<i>IGALIKO</i>
1	98.36 %	72.39 %

MULTIEDIT <i>Influencia del parámetro m en la bondad del clasificador</i>		
<i>Particiones</i>	<i>GALAXY</i>	<i>IGALIKO</i>
$m = 3$	98.57 %	79.51 %
	98.57 %	80.30 %
	98.57 %	78.58 %
$m = 4$	98.57 %	79.07 %
	98.36 %	79.64 %
	98.57 %	79.48 %
$m = 5$	98.36 %	80.49 %
	98.57 %	79.89 %
	98.36 %	79.26 %
$m = 6$	98.57 %	79.86 %
	98.36 %	79.18 %
	98.36 %	79.64 %
$m = 7$	98.57 %	78.80 %
	98.36 %	79.37 %
	98.36 %	78.88 %

En los datos de la galaxia, la edición de éstos no mejora la bondad estimada del clasificador, ya que éstos se pueden clasificar con facilidad (véase las nubes de puntos a la derecha). Por otro lado, en Igaliko, el método de edición sí es bastante útil para limpiar las fronteras de decisión y mejorar la bondad del clasificador.

El valor ideal de m es aquél que permite mejorar al máximo la bondad del clasificador sin que se consuma demasiado tiempo de CPU. Este valor se encuentra en torno a 5 para los datos de Igaliko. Como se ha comentado, no resulta realmente necesario editar los datos de la galaxia en espiral.



Influencia del parámetro I sobre el tiempo empleado en la edición y en la bondad de la clasificación 1-NN usando como referencia el conjunto de datos editado

Para analizar la influencia del parámetro I en el algoritmo de multiedición se utilizarán los valores $I=2$, $I=3$ e $I=4$, realizando nuevamente tres experimentos por cada uno de ellos. El valor de m se mantendrá fijo igual a 3.

GALAXY: 1027 patrones inicialmente			
Valor de I	Tamaño del conjunto editado	Tiempo empleado (en segundos)	Porcentaje de clasificación
2	1006	24.0	98.57 %
	1006	41.7	98.57 %
	1007	42.0	98.57 %
3	1007	47.5	98.57 %
	1005	47.4	98.57 %
	999	53.1	98.36 %
4	1009	53.9	98.57 %
	1003	89.9	98.36 %
	1007	65.4	98.57 %

En todos los casos se clasifican mal sólo 7 u 8 de los 490 patrones incluidos en el conjunto de entrenamiento de los datos de la galaxia en espiral. De hecho, el simple clasificador del vecino más cercano sólo asigna 8 etiquetas erróneas a los datos del conjunto de prueba:

Clase	1	2	3	%
1	13	0	0	100.00%
2	0	277	4	98.57%
3	0	4	192	97.95%

Bondad estimada del clasificador: 98.36% (482/490)

Ver

Así mismo, el tamaño del conjunto editado prácticamente no varía aun cambiando el valor del parámetro I . La mayor parte de los patrones descartados se descartan en las primeras iteraciones del algoritmo, por lo que el parámetro I no influye demasiado en la edición de los datos.

El único efecto destacable del aumento del parámetro I es el incremento el tiempo empleado por el algoritmo de multiedición. Cuanto más aumentemos I más iteraciones se realizarán, aun cuando en esas iteraciones extra no se descarten apenas muestras del conjunto de entrenamiento (como se ha dicho, la limpieza de las fronteras de decisión se realiza sobre todo en las primeras iteraciones del algoritmo).

A continuación se realizarán 15 experimentos para estudiar la media, mediana y desviación típica de los 15 valores de bondad obtenidos aplicando el algoritmo de condensado de Hart. En todos los experimentos se considerará el mismo fichero de entrada, que será, obviamente, alguno de los obtenidos como ficheros de salida con el *Multiedit*.

Algoritmo de condensado de Hart		
	<i>GALAXY</i>	<i>IGALIKO</i>
<i>Tamaño del conjunto inicial</i>	1027	7034
<i>Tamaño del conjunto editado</i>	1005	4355
<i>Tiempo de edición</i>	98 seg	1537 seg
<i>Tamaño del conjunto condensado</i>	4	41
	4	45
	5	48
	6	50
	6	50
	8	51
	8	51
<i>Mediana</i>	9	52
	9	56
	9	56
	9	56
	9	60
	11	60
	11	60
	12	67
<i>Tiempo de condensado</i>	2 seg	7 seg

Algoritmo de condensado de Hart		
	<i>GALAXY</i>	<i>IGALIKO</i>
<i>Bondad de la clasificación</i>	97.95 %	78.06 %
	97.95 %	78.17 %
	97.95 %	78.19 %
	97.95 %	78.27 %
	97.95 %	78.52 %
<i>Mediana</i>	97.95 %	78.58 %
	98.16 %	78.60 %
	98.16 %	78.63 %
	98.36 %	78.69 %
	98.36 %	78.77 %
	98.36 %	78.82 %
	98.36 %	79.07 %
	98.57 %	79.10 %
	98.57 %	79.12 %
	98.77 %	79.23 %
<i>Media</i>	98.22 %	78.65 %
<i>Desviación típica</i>	0.27 %	0.35 %

Como se puede apreciar en la tabla superior, el algoritmo de condensado de Hart consigue reducir notablemente el coste computacional de la clasificación 1-NN a costa de una ligera pérdida en la bondad de la clasificación obtenida (si no existiese solapamiento entre clases esta pérdida sería nula). El algoritmo de Hart reduce de una forma razonable el tamaño del conjunto de referencia, con lo que consigue el objetivo principal del condensado.

La selección aleatoria de las muestras (implementada barajando inicialmente las muestras disponibles) hace que el algoritmo no sea reproducible y provoca la aparición de pequeñas diferencias en sus distintas ejecuciones sobre el mismo conjunto de datos.

NOTA: En el disco adjunto se incluyen también los resultados obtenidos con otras bases de datos. En las memorias se han elegido GALAXY e IGALIKO al ser éstas las únicas que tienen preestablecida la partición entre conjunto de entrenamiento y conjunto de prueba.

3. Métodos de aprendizaje adaptativo

Los métodos de aprendizaje adaptativo destacan por la sencillez de las heurísticas que utilizan y su rapidez de cálculo. Su principal inconveniente reside en la estimación adecuada de sus parámetros.

LVQ [Linear Vector Quantization]

El conjunto de métodos LVQ (de aprendizaje por cuantificación vectorial) están basados en los mapas autoorganizativos [*SOM: Self-Organizing Maps*]. Se caracterizan por utilizar un número fijo y relativamente bajo de prototipos para aproximar las funciones de densidad de probabilidad de las distintas clases.

Dada una secuencia de observaciones vectoriales (patrones), se selecciona un conjunto inicial de vectores de referencia (codebooks o prototipos). Iterativamente, se selecciona una observación X y se actualiza el conjunto de prototipos de forma que case mejor con X .

Una vez finalizado el proceso de construcción del conjunto de prototipos (codebooks), las observaciones se clasificarán utilizando la regla 1-NN (buscando el vecino más cercano en el conjunto de vectores de referencia).

Inicialización del conjunto de prototipos

La función de inicialización es la encargada de seleccionar apropiadamente un subconjunto de N_p prototipos (siendo N_p uno de los parámetros de los algoritmos LVQ) tomados generalmente del conjunto de observaciones.

El número de prototipos por clase puede ser proporcional al número de muestras por clase (usando el parámetro `apriori` en el *Analizador Numérico*, o el programa `propinit` en *LVQ_PAK*) o el mismo para todas las clases (omitiendo el parámetro `apriori` en *NC* o utilizando el programa `eveninit` en *LVQ_PAK*).

La selección inicial del conjunto de prototipos se realiza procesando secuencialmente las muestras de entrenamiento para cada clase hasta que se haya seleccionado el número de prototipos fijado para cada clase. Una muestra se añade al conjunto de prototipos si su clasificación mediante la regla k -NN es correcta (vg: regla 5-NN). Esta selección inicial del conjunto de prototipos acelera la convergencia del proceso de aprendizaje.

Actualización del conjunto de prototipos

Los diferentes métodos LVQ se diferencian en la forma en que se actualiza el conjunto de prototipos en cada paso del aprendizaje. Dado un patrón X , se realizarán correcciones de premio o castigo sobre el conjunto de prototipos según la clasificación de X se realice correctamente o no. El conjunto de prototipos final no tiene por qué ser un subconjunto del conjunto inicial de patrones.

El número de pasos de aprendizaje r debe ser suficientemente grande y es otro de los parámetros utilizados en los métodos LVQ. Otro parámetro de estos métodos es la función de ganancia o razón de aprendizaje α , que suele ser una función lineal decreciente de la que basta especificar su valor inicial $\alpha(0)$.

Para la realización de estas prácticas se han implementado los métodos LVQ-1 y OLVQ-1. En *LVQ_PAK* también se incluyen los métodos LVQ-2.1 y LVQ-3. No obstante, los resultados obtenidos por los distintos métodos LVQ son muy similares.

LVQ-1

En cada paso de aprendizaje, LVQ-1 modifica únicamente el prototipo más cercano al patrón $X(t)$. Si la clase del prototipo coincide con la de la muestra, el prototipo se acerca a la muestra. En el caso contrario, el prototipo se aleja de ésta (por lo cual este algoritmo tiende a reducir la densidad de prototipos alrededor de las fronteras de decisión).

La dirección de la corrección coincide con la línea recta que une el patrón y su prototipo más cercano. El valor del desplazamiento depende del factor de ganancia $\alpha(t)$, que es mayor en las primeras iteraciones y menor en las últimas. Se recomienda emplear un valor pequeño para $\alpha(0)$, menor que 0.1 (0.02 ó 0.03). Suele bastar con un número de pasos de aprendizaje entre $50N_p$ y $200N_p$, donde N_p es el número total de prototipos (codebooks).

Desde la línea de comandos se puede ejecutar el método LVQ-1 escribiendo:

```
lvq1 <fichero_cfg> <Np> <r> <alpha> [apriori]
```

donde <fichero_cfg> es el fichero de configuración de los datos, <Np> es el número de pototipos deseado (codebooks), <r> es el número de pasos de aprendizaje, <alpha> es el valor inicial de la razón de aprendizaje y *apriori* indica si para cada clase deseamos un número de prototipos proporcional al número de muestras por clase o no (equivalente a utilizar *propinit* o *eveninit* en *LVQ_PAK*).

Obviamente, también se puede probar el método LVQ-1 desde el entorno gráfico integrado del Analizador Numérico.

El método OLVQ-1 es idéntico al LVQ-1 salvo que cada prototipo (codebook) mantiene su razón de aprendizaje propia $\alpha_i(t)$. Los valores iniciales de $\alpha_i(t)$ deben ser más altos que en LVQ-1 (p.ej. 0.3). Desde la línea de comandos basta teclear:

```
olvq1 <fichero_cfg> <Np> <r> <alpha> [apriori]
```

donde <fichero_cfg> es el fichero de configuración de los datos, <Np> es el número de pototipos deseado (codebooks), <r> es el número de pasos de aprendizaje, <alpha> es el valor inicial de la razón de aprendizaje y *apriori* indica si para cada clase deseamos un número de prototipos proporcional al número de muestras por clase o no (equivalente a utilizar *propinit* o *eveninit* en *LVQ_PAK*).

DSM [Decision Surface Mapping]

DSM es un variante de los métodos de aprendizaje adaptivo que consiste en aproximar directamente las fronteras de decisión (para lo que requiere un conjunto editado de datos).

Si una muestra es clasificada correctamente, no se realiza ninguna acción. Cuando se produce un error de clasificación durante la fase de aprendizaje, la corrección se aplica a dos prototipos simultáneamente:

- ▶ Se castiga al prototipo más cercano (que provoca el error de clasificación).
- ▶ Se premia al prototipo más cercano de la misma clase que el patrón considerado.

Lo más probable es que los errores de clasificación se produzcan cerca de las fronteras de decisión. DSM actúa sobre parejas de prototipos situados a ambos lados de las fronteras de decisión y no permite tratar problemas con solapamiento entre clases.

El método DSM se puede probar desde el entorno gráfico integrado y también desde la línea de comandos escribiendo:

```
dsm <fichero_cfg> <Np> <r> <alpha> [apriori]
```

donde <fichero_cfg> es el fichero de configuración de los datos, <Np> es el número de pototipos deseado (codebooks), <r> es el número de pasos de aprendizaje, <alpha> es el valor inicial de la razón de aprendizaje (menor o igual a 0.3) y *apriori* indica si para cada clase deseamos un número de prototipos proporcional al número de muestras por clase o no (equivalente a utilizar *propinit* o *eveninit* en *LVQ_PAK*).

Experimentación

Los experimentos realizados han utilizado exclusivamente el método LVQ-1 sobre las bases de datos de Igaliko y de la galaxia espiral. Los porcentajes de clasificación se ofrecen sobre el conjunto de entrenamiento (%TRA) y sobre el conjunto de prueba (%TST)

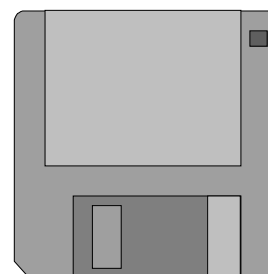
Inicialización del conjunto de prototipos

EVENINIT Igual número de prototipos para cada clase				
N_p	<i>GALAXY</i>		<i>IGALIKO</i>	
	%TRA	%TST	%TRA	%TST
30	98.34 %	98.36 %	68.92 %	68.33 %
60	97.76 %	96.93 %	68.39 %	69.76 %
90	97.66 %	96.32 %	65.55 %	67.40 %

PROPINIT Número de prototipos proporcional al número de muestras de cada clase				
N_p	<i>GALAXY</i>		<i>IGALIKO</i>	
	%TRA	%TST	%TRA	%TST
30	97.56 %	98.16 %	65.95 %	65.32 %
60	97.17 %	96.53 %	69.24 %	70.22 %
90	97.85 %	96.73 %	66.98 %	68.39 %

Nota:

Cada experimento se ha repetido tres veces, aunque siempre produce los mismos resultados, ya que la inicialización del conjunto de prototipos sigue un algoritmo determinista (no probabilístico). En el disco adjunto se pueden consultar todos los resultados obtenidos (incluyendo las distintas tablas de contingencia).

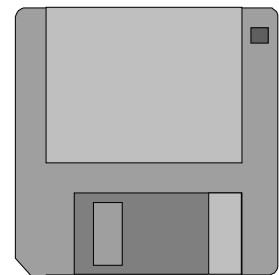


Aunque en principio se deberían obtener siempre mejores resultados cuanto mayor sea el tamaño del conjunto de prototipos, esto no es así. Los resultados obtenidos con un conjunto de prototipos de un tamaño dado dependen en gran medida del conjunto de datos analizado. Por ejemplo, en las imágenes de la galaxia, con sólo tres patrones se conseguirían prácticamente los mismos resultados que con noventa (de hecho, noventa son demasiados si tenemos en cuenta que sólo disponemos de unas mil muestras en el conjunto de entrenamiento).

El uso de `propinit` o `eveninit` tampoco influye demasiado en la bondad estimada del conjunto de prototipos ya que al clasificar se utiliza la regla 1-NN (sólo nos quedamos con el vecino más cercano). Si utilizamos `propinit` corremos el riesgo de que alguna clase pueda quedar sin patrones en el conjunto de codebooks si N_p es pequeño. Con `eveninit`, cuando el tamaño del conjunto de prototipos es grande y el número de muestras de algunas clases es pequeño, nos arriesgamos a que la bondad estimada del conjunto inicial de prototipos se vea deteriorada.

Aprendizaje con LVQ

A continuación se estudiará la influencia de los distintos parámetros asociados al aprendizaje mediante LVQ-1 en el aprendizaje. Se utilizarán como valores inicial de la razón de aprendizaje 0.01 , 0.02 y 0.03 . El número de pasos del aprendizaje tomará los valores $50N_p$, $100N_p$, $150N_p$ y $200N_p$. En el disco adjunto se pueden encontrar los resultados de las tres ejecuciones de cada experimento.



En las siguientes páginas aparecen los resultados obtenidos para las distintas combinaciones de N_p , r y $\alpha(0)$ cuando se aplica el método LVQ-1 a los conjuntos de imágenes de la galaxia espiral (`GALAXY.CFG`) y de Igaliko (`IGALIKO.CFG`). Siempre se utiliza `PROPINIT`.

En las tablas se recogen los porcentajes de clasificación obtenidos sobre los conjuntos de entrenamiento y de prueba (%TRA y %TST), que corresponden a la evaluación de la bondad del aprendizaje (con `accuracy` en `LVQ_PAK`) y a la estimación de la bondad de clasificación, respectivamente.

Conforme aumentan los valores de N_p (número de prototipos) y r (el número de pasos de aprendizaje), suele aumentar la bondad de la clasificación, siempre que estos parámetros tengan valores suficientemente elevados. Sus valores ideales dependen del problema concreto.

La mejora en la bondad de la clasificación conforme aumentamos el valor de r se produce hasta que el método converge (momento a partir del cual el sistema empieza a sobre-aprender). Cuanto mayor sea el valor inicial de la razón de aprendizaje α , la convergencia de LVQ será más rápida.

$$N_p = 30$$

$$r = 50 N_p$$

$\alpha(0)$	GALAXY		IGALI KO	
	%TRA	%TST	%TRA	%TST
0.01	97.66 %	97.95 %	62.52 %	63.40 %
0.02	97.66 %	98.16 %	62.36 %	62.88 %
0.03	97.76 %	98.36 %	61.82 %	62.36 %

$$r = 100 N_p$$

$\alpha(0)$	GALAXY		IGALI KO	
	%TRA	%TST	%TRA	%TST
0.01	97.66 %	98.36 %	63.20 %	63.57 %
0.02	97.85 %	98.16 %	60.84 %	61.02 %
0.03	98.24 %	98.36 %	58.04 %	58.17 %

$$r = 150 N_p$$

$\alpha(0)$	GALAXY		IGALI KO	
	%TRA	%TST	%TRA	%TST
0.01	97.85 %	98.36 %	57.74 %	58.39 %
0.02	98.24 %	98.36 %	58.81 %	59.79 %
0.03	98.34 %	98.36 %	56.56 %	58.50 %

$$r = 200 N_p$$

$\alpha(0)$	GALAXY		IGALI KO	
	%TRA	%TST	%TRA	%TST
0.01	98.05 %	98.16 %	59.98 %	60.23 %
0.02	98.34 %	98.36 %	58.87 %	60.42 %
0.03	98.24 %	98.36 %	57.79 %	58.99 %

$$N_p = 60$$

$$r = 50 N_p$$

$\alpha(0)$	GALAXY		IGALI KO	
	%TRA	%TST	%TRA	%TST
0.01	97.46 %	97.14 %	67.85 %	69.13 %
0.02	98.14 %	97.55 %	66.73 %	67.89 %
0.03	98.63 %	98.16 %	67.72 %	68.85 %

$$r = 100 N_p$$

$\alpha(0)$	GALAXY		IGALI KO	
	%TRA	%TST	%TRA	%TST
0.01	98.24 %	97.95 %	72.20 %	73.40 %
0.02	98.92 %	98.16 %	73.91 %	75.29 %
0.03	98.92 %	98.16 %	74.52 %	75.62 %

$$r = 150 N_p$$

$\alpha(0)$	GALAXY		IGALI KO	
	%TRA	%TST	%TRA	%TST
0.01	98.63 %	98.16 %	75.95 %	76.77 %
0.02	98.92 %	98.16 %	78.12 %	78.71 %
0.03	98.92 %	98.36 %	78.17 %	78.36 %

$$r = 200 N_p$$

$\alpha(0)$	GALAXY		IGALI KO	
	%TRA	%TST	%TRA	%TST
0.01	98.92 %	98.16 %	76.25 %	76.58 %
0.02	98.92 %	98.36 %	77.66 %	78.22 %
0.03	98.83 %	98.36 %	78.04 %	78.77 %

$$N_p = 90$$

$$r = 50 N_p$$

$\alpha(0)$	GALAXY		IGALI KO	
	%TRA	%TST	%TRA	%TST
0.01	98.34 %	97.55 %	68.66 %	69.84 %
0.02	98.73 %	98.16 %	69.71 %	71.04 %
0.03	98.83 %	98.16 %	70.71 %	72.22 %

$$r = 100 N_p$$

$\alpha(0)$	GALAXY		IGALI KO	
	%TRA	%TST	%TRA	%TST
0.01	98.83 %	98.16 %	76.24 %	77.12 %
0.02	98.92 %	98.16 %	77.02 %	77.26 %
0.03	98.92 %	98.16 %	78.41 %	78.88 %

$$r = 150 N_p$$

$\alpha(0)$	GALAXY		IGALI KO	
	%TRA	%TST	%TRA	%TST
0.01	98.83 %	98.16 %	75.83 %	76.60 %
0.02	98.92 %	98.16 %	78.06 %	78.74 %
0.03	98.92 %	98.36 %	78.12 %	79.53 %

$$r = 200 N_p$$

$\alpha(0)$	GALAXY		IGALI KO	
	%TRA	%TST	%TRA	%TST
0.01	98.83 %	98.16 %	75.86 %	76.55 %
0.02	98.92 %	98.16 %	76.69 %	77.48 %
0.03	98.92 %	98.16 %	77.87 %	79.04 %

4. Métodos de agrupamiento

Los métodos de agrupamiento o clustering (arracimamiento en algunas traducciones) constituyen un tipo de aprendizaje por descubrimiento muy similar a la inducción. En el aprendizaje inductivo, un programa aprende a clasificar objetos basándose en etiquetados proporcionados por un profesor (aprendizaje supervisado). En los métodos de agrupamiento no se suministran los datos etiquetados: el programa debe descubrir por sí mismo las clases naturales existentes.

Por ejemplo, el programa AUTOCLASS (Cheeseman, Self, Kelly, Taylor, Freeman y Stutz, “*Bayesian Classification*”, Proceedings AAAI88, 1988) usa razonamiento bayesiano para, dado un conjunto de datos de entrenamiento, sugerir un conjunto de clases plausible. Este programa encontró nuevas clases significativas de estrellas a partir de sus datos del espectro infrarrojo, lo que puede considerarse ejemplo de descubrimiento por parte de una máquina (los hechos descubiertos eran desconocidos para los astrónomos).

Las funciones de densidad de probabilidad suelen tener una moda o un máximo en una región; es decir, las observaciones tienden a agruparse en torno a una región del espacio de patrones cercana a la moda. Las técnicas de agrupamiento analizan el conjunto de observaciones disponibles para determinar la tendencia de los patrones a agruparse. Estas técnicas permiten realizar una clasificación asignando cada observación a un agrupamiento [cluster], de forma que cada agrupamiento sea más o menos homogéneo y diferenciable de los demás.

Los agrupamientos naturales obtenidos mediante una técnica de agrupamiento mediante similitud resultan muy útiles a la hora de construir clasificadores cuando no están bien definidas las clases (no existe un conocimiento suficiente de las clases en que se pueden distribuir las observaciones), cuando se desea analizar un gran conjunto de datos (“divide y vencerás”) o, simplemente, cuando existiendo un conocimiento completo de las clases se desea comprobar la validez del entrenamiento realizado y del conjunto de variables escogido.

Los métodos de agrupamiento asocian un patrón a un agrupamiento siguiendo algún *criterio de similaridad*. Algunas medidas de disimilaridad habituales son la distancia euclídea, la distancia euclídea normalizada, la distancia euclídea ponderada, la distancia de Mahalanobis... Las medidas de disimilaridad deben ser aplicables entre pares de patrones, entre un patrón y un agrupamiento y, finalmente, entre pares de agrupamientos. En el paquete `rf.distance` se encuentran implementadas las distintas medidas de distancia citadas:

- ▶ *Distancia euclídea*: $d^2(X_i, X_j) = (X_i - X_j)^T (X_i - X_j) = \|X_i - X_j\|^2$
- ▶ *Distancia euclídea normalizada*: Igual que la anterior, salvo que los valores de cada variable se normalizan en el intervalo [0,1] para que unas características no influyan más que otras al calcular las distancias.

- *Distancia euclídea ponderada:* $d^2(X_i, X_j) = \sum_{k=1}^d \frac{1}{\sigma_k^2} (X_{ki} - X_{kj})^2$
- *Distancia de Mahalanobis:* $r^2(X_i, X_j) = (X_i - X_j)^T \Sigma^{-1} (X_i - X_j)$
donde Σ^{-1} es la inversa de la matriz de covarianza.

Formulación del problema

Consideremos un conjunto M de ejemplos y K clusters (C_1, C_2, \dots, C_k):

- ☞ $C_i \neq \emptyset$ ($\#C_i > 0$)
- ☞ $i \neq j \rightarrow C_i \cap C_j = \emptyset$
- ☞ $M = \cup C_i = \{X_1, X_2 \dots X_m\}$ tal que $X_i \in \mathbb{R}^n$

Matemáticamente, el problema puede formularse como la minimización de

$$f(W, Z; X) = \sum_{i=1}^m \sum_{j=1}^k w_{ij} \|X_i - Z_j\|^2$$

$$\sum_{j=1}^k w_{ij} = 1, \quad 1 \leq i \leq m$$

$$w_{ij} \in \{0,1\}$$

- X_i es el vector patrón correspondiente al ejemplo i -ésimo ($X_i \in \mathbb{R}^n$).
- Z_j es el centro del j -ésimo cluster ($Z_j \in \mathbb{R}^n$).
- W es la matriz de pertenencia ($m \times k$) tal que w_{ij} es 1 si $X_i \in C_j$ y 0 en caso contrario.
- k es el número de clusters (clases en el problema de clasificación)
- m es el número de ejemplos del conjunto de casos de entrenamiento.

La función f no es convexa, por lo que pueden existir mínimos locales no óptimos.

La minimización de la función f requiere conocer a priori el número de agrupamientos k (si no el problema sería trivial: un agrupamiento para cada caso de entrenamiento). No obstante, existen técnicas que permiten ajustar el número de agrupamientos (fusionando y dividiendo agrupamientos) y tratar elementos discordantes [*outliers* en inglés] debidos, por ejemplo, a ruido en la adquisición de datos.

La agrupamientos detectados dependen del algoritmo empleado, del valor dado a sus parámetros, de los datos utilizados y de la medida de similaridad/disimilaridad adoptada.

Se han propuesto cientos de algoritmos de agrupamiento más o menos específicos. Según se use o no una función criterio se distinguen los algoritmos directos o constructivos (basados en aproximaciones heurísticas) de los algoritmos indirectos o por optimización.

La estructura general de un algoritmo iterativo de agrupamiento directo es la siguiente:

Seleccionar una partición inicial del conjunto de ejemplos en K clusters.
Calcular los centros de los clusters
Mientras que no se estabilicen los clusters (matriz W)
Repetir
Generar una nueva partición (asignando cada patrón al cluster cuyo centro esté más cercano)
Calcular los centroides de los nuevos clusters
Hasta que se obtenga un valor óptimo de la función de evaluación
Ajustar el número de clusters mezclando y separando clusters existentes o eliminando clusters pequeños o "periféricos"

A continuación se citan algunos algoritmos indirectos conocidos que utilizan distintas estrategias de búsqueda: DHB, DHF y algoritmo de Forgy. Véanse las siguientes referencias para obtener más información sobre ellos:

- *A.K.Jain & R.C.Dubes: "Algorithms for Clustering Data". Prentice Hall, 1988.*
- *M.A.Ismael & M.S.Kamel: "Multidimensional Data Clustering Utilizing Hybrid Search Strategies". Pattern Recognition 22, 1989, pgs. 75-89.*

Tras ellos se expondrán algunos algoritmos heurísticos de agrupamiento representativos. No existe ninguna medida objetiva que permita su comparación, por lo que los comentarios acerca de la bondad de cada método son, generalmente, sesgados y subjetivos.

Algoritmo DHB

Búsqueda primero en anchura

Seleccionar arbitrariamente una configuración inicial de los clusters

$n_j = \#C_j$

Calcular los centros de los k clusters Z_i y la función objetivo f

Repetir

Para l de 1 a n // $X_l \in C_i$ && $n_i = \#C_i$

Si $n_i \neq 1$

$\Delta_i = n_i \cdot \|X_l - Z_i\|^2 / (n_i - 1)$

$\Delta_{\min} = \Delta_i$

Para j de 1 a k

Si $j \neq i$

$\Delta_j = n_j \cdot \|X_l - Z_j\|^2 / (n_j + 1)$

Si $\Delta_j < \Delta_{\min}$

$\Delta_{\min} = \Delta_j$

$d = j$

Si $\Delta_{\min} < \Delta_i$

Mover X_l a C_d

Actualizar Z_i y Z_d

Actualizar n_i , n_d y f

Hasta que la función objetivo f no cambie tras n iteraciones

✓ Un patrón X pasa del cluster s al cluster r si se cumplen las siguientes condiciones:

$$\#C_s \frac{\|X - Z_s\|^2}{\#C_s - 1} > \#C_r \frac{\|X - Z_r\|^2}{\#C_r + 1}$$

y

$$\#C_r \frac{\|X - Z_r\|^2}{\#C_r + 1} = \min_{\substack{1 \leq j \leq k \\ j \neq s}} \left\{ \#C_j \frac{\|X - Z_j\|^2}{\#C_j + 1} \right\}$$

Algoritmo DHF

Búsqueda primero en profundidad

Seleccionar arbitrariamente una configuración inicial de los clusters

$n_j = \#C_j$

Calcular los centros de los k clusters Z_i y la función objetivo f

Repetir

Para l de 1 a n // $X_l \in C_i$ && $n_i = \#C_i$

Si $n_i \neq 1$

$\Delta_i = n_i \cdot ||X_l - Z_i||^2 / (n_i - 1)$

Para j de 1 a k

Si $j \neq i$

$\Delta_j = n_j \cdot ||X_l - Z_j||^2 / (n_j + 1)$

Si $\Delta_j < \Delta_i$

Mover X_l a C_j

Actualizar Z_i, Z_j, n_i, n_j y f

Abandonar bucle

Hasta que la función objetivo f no cambie tras n iteraciones

Algoritmo de Forgy

Permite crear clusters adicionales y eliminar elementos discordantes. Cuando el algoritmo utilizado converge con k clusters, se crea un nuevo cluster con centro en el patrón x_i si:

$$\left| d(x_i, Z_q) - \bar{d}_i \right| \leq \bar{d}_i \cdot T_1 \quad 0 < T_1 < 1$$

donde Z_q es el centro del cluster al que pertenece x_i y la distancia media del patrón a los clusters es

$$\bar{d}_i = \frac{1}{K} \sum_{k=1}^K d(x_i, Z_k)$$

Cuanto mayor sea T_1 mayor será el número de clusters que se creen. Cuando el número de patrones de un cluster esté por debajo de T_2 entonces los patrones de este cluster se consideran elementos discordantes y se ignoran.

Método adaptativo [*adaptive sample set construction*]

El método adaptativo es un algoritmo heurístico de agrupamiento que se puede utilizar cuando no se conoce de antemano el número de clases del problema. Entre sus ventajas se encuentran su simplicidad y eficiencia. Además, las observaciones se procesan secuencialmente. Por desgracia, su comportamiento está sesgado por el orden de presentación de los patrones y presupone agrupamientos compactos separados claramente de los demás.

El primer agrupamiento se escoge arbitrariamente. Se asigna un patrón a un cluster si la distancia del patrón al centroide del cluster no supera un umbral. En caso contrario, se crea un nuevo agrupamiento.

Este algoritmo incluye una clase de rechazo a la hora de clasificar observaciones. Los patrones se asignan por la regla de mínima distancia. Algunos patrones no son clasificados si la distancia al agrupamiento más cercano es mayor que el umbral τ .

Parámetros

τ	Umbral de distancia
θ	Fracción (entre 0 y 1)
$\{X_i\}$	Conjunto de patrones

Variables

A	Número actual de agrupamientos
Z_i	Centroide del agrupamiento i

Algoritmo de agrupamiento

Inicialización: $A=1$, $Z_1=X_1$

Mientras queden patrones por asignar

Obtener el siguiente patrón X y calcular $d(X, Z_i)$ $i=1..A$.

Asignar X al agrupamiento más cercano Z_i si $d(X, Z_i) \leq \theta\tau$

Formar un nuevo agrupamiento con X si $d(X, Z_i) > \tau$: $A++$, $Z_A=X$

Recalcular el centroide Z_i y la varianza C_i del agrupamiento.

Resultados obtenidos para los datos de la galaxia espiral

τ	θ	Agrupamientos	Bondad estimada
10	0.8	347	74.89 %
25	0.8	284	81.83 %
50	0.8	202	87.34 %
75	0.8	176	87.79 %
100	0.5	152	90.81 %
100	0.8	151	90.81 %
100	1.0	150	91.02 %
150	0.5	122	92.65 %
150	0.8	120	92.85 %
150	1.0	117	93.26 %
200	0.8	96	94.89 %
250	0.8	85	95.10 %
1000	0.5	30	98.36 %
1000	0.8	28	97.95 %
1000	1.0	27	98.16 %
10000	0.8	4	96.32 %
25000	0.8	3	95.91 %
100000	0.8	2	94.89 %

☞ En esta tabla se puede apreciar como el parámetro fundamental a la hora de conseguir un “buen” agrupamiento con este algoritmo es el umbral τ . Cuanto mayor sea este umbral, menos agrupamientos se formarán.

☞ El parámetro θ influye menos en el comportamiento final del método adaptativo.

Algoritmo de Batchelor y Wilkins (algoritmo de máxima distancia)

Como sucedía con el método adaptativo, el algoritmo de Batchelor y Wilkins es un método de agrupamiento con número de clases desconocido.

Parámetros

f Fracción de la distancia media entre agrupamientos

Algoritmo de agrupamiento

Primer agrupamiento: Patrón escogido al azar

Segundo agrupamiento: Patrón más alejado del primer agrupamiento

Mientras se creen nuevos agrupamientos

 Obtener el patrón más alejado de los agrupamientos existentes
 (máximo de las distancias mínimas de los patrones a los
 agrupamientos)

 Si la distancia del patrón escogido al conjunto de agrupamientos
 es mayor que una fracción de la distancia media entre los
 agrupamientos, crear un agrupamiento con el patrón seleccionado

Asignar cada patrón a su agrupamiento más cercano

Ejemplo: Galaxia espiral

Fracción f	Agrupamientos	Bondad estimada
0.0	1027	-
0.1	10	98.16 %
0.2	3	93.67 %
≥ 0.3	2	57.34 %

Algoritmo de las K medias (K-MEANS, J.B. MacQueen, 1967)

El algoritmo de las K medias (o K-Means) es probablemente el algoritmo de agrupamiento más conocido. Es un método de agrupamiento heurístico con número de clases conocido (K). El algoritmo está basado en la minimización de la distancia interna (la suma de las distancias de los patrones asignados a un agrupamiento al centroide de dicho agrupamiento). De hecho, este algoritmo minimiza la suma de las distancias al cuadrado de cada patrón al centroide de su agrupamiento.

El algoritmo es sencillo y eficiente. Además, procesa los patrones secuencialmente (por lo que requiere un almacenamiento mínimo). Sin embargo, está sesgado por el orden de presentación de los patrones (los primeros patrones determinan la configuración inicial de los agrupamientos) y su comportamiento depende enormemente del parámetro K.



Seleccionar arbitrariamente una configuración inicial de los clusters

Repetir

 Calcular los centros de los clusters Z_j

 Redistribuir los patrones entre los clusters utilizando la mínima distancia euclídea al cuadrado como clasificador:

$$X_i \in C_j \leftrightarrow ||X_i - Z_j||^2 < ||X_i - Z_l||^2 \quad \forall l \neq j$$

Hasta que no cambien los centros de los clusters



Ejemplo: Galaxia espiral

K	Bondad estimada
3	94.89 %
5	92.04 %
7	98.16 %

NOTA: La convergencia del algoritmo depende de la configuración inicial de los clusters. El algoritmo es eficiente y encuentra siempre un óptimo local, por lo que podemos utilizarlo como técnica de búsqueda local en algoritmos GRASP [*Greedy Randomized Adaptive Search Procedure*] o GLS [*Genetic Local Search*].

Enfriamiento simulado [Simulated Annealing]

➤ *S.Z.Selim & K.Alsultan: "A Simulated Annealing for the Clustering Problem". Pattern Recognition 24, 1991, pgs. 1003-1008.*

➤ *R.W.Klein & R.C.Dubes: "Experiments in Projection and Clustering by Simulated Annealing". Pattern Recognition 22, 1989, pgs. 213-220.*

El problema del agrupamiento se puede resolver con cualquier técnica general que permita realizar una exploración del espacio de búsqueda (el espacio de las posibles asignaciones). Una forma de resolverlo (bastante ineficiente por cierto) es utilizar un algoritmo de enfriamiento simulado:

Simulated Annealing

Sólido que se enfría
Configuración actual
Perturbación de la configuración actual
Energía E asociada a la configuración
Aceptación de la configuración si disminuye E
Aceptación de una configuración con mayor energía

Clustering

Problema de optimización
Asignación actual de patrones a clusters
Generación de una nueva asignación
Función objetivo J
Aceptación de la asignación si mejora J
Aceptación de la asignación si se espera una mejor

Nuestra función objetivo puede ser, por ejemplo, la suma de las distancias al cuadrado de los patrones a los centroides de los agrupamientos (la función que minimiza el K-Means).

Simplemente hemos de definir un método de generación de vecinos (construcción de una nueva asignación a partir de la asignación actual), fijar un esquema de enfriamiento (constante proporcional en este caso), establecer el valor de temperatura inicial, escoger la temperatura final del proceso de enfriamiento y, por último, tener algo de paciencia para obtener buenos resultados...

Generación de una nueva asignación

```
flag = falso

Mientras flag = falso

    Para i de 1 a n

        Generar un número aleatorio  $u \sim U(0,1)$ 

        Si  $u > P$ 
            flag = true
             $S_i = \{ c \mid 1 \leq c \leq C, c \neq a_i \}$ 
            Seleccionar aleatoriamente un elemento  $c$  de  $S_i$ 
            Asignar el patrón  $i$  al cluster  $c$  [ $a_i := c$ ]
```

Algoritmo de agrupamiento con enfriamiento simulado

Inicialización:

- Fijar T_0 (temperatura inicial), ε (temperatura final) y μ (factor [<1])
- Seleccionar un conjunto arbitrario de C clusters (vectores A_{best} y $A_{\text{candidate}}$)
- Calcular la función objetivo: J_{best} y $J_{\text{candidate}}$
- $T = T_0$

Mientras $T \geq \varepsilon$

 Generar una nueva asignación A_{trial} (con valor J_{trial})

 Si $J_{\text{trial}} > J_{\text{candidate}}$

 Generar un número aleatorio $y \sim U(0,1)$

 Si $y \leq \exp(-(J_{\text{trial}} - J_{\text{candidate}})/T)$

$J_{\text{candidate}} = J_{\text{trial}}$

$A_{\text{candidate}} = A_{\text{trial}}$

 si no

$A_{\text{candidate}} = A_{\text{trial}}$

$J_{\text{candidate}} = J_{\text{trial}}$

 Si $J_{\text{trial}} < J_{\text{best}}$

$A_{\text{best}} = A_{\text{trial}}$

$J_{\text{best}} = J_{\text{trial}}$

 count = 0

 si no

 count++

 Si count $\geq N$

$T = \mu T$

El enfriamiento simulado es menos dependiente de la configuración inicial de los clusters que el algoritmo K-MEANS y, además, no se queda estancado en óptimos locales. El esquema de enfriamiento $T_i = 0.9T_{i-1}$ obtiene prácticamente los mismos resultados que otros criterios de convergencia más lenta (vg: criterio de Cauchy, criterio de Boltzmann...). Aun a riesgo de realizar una peor exploración del espacio de búsqueda, en muchas ocasiones es preferible obtener una buena solución en un tiempo limitado.

El esquema de generación de vecinos empleado para el algoritmo SA tiende a mantener siempre una fracción $1/k$ de las muestras en cada uno de los k clusters, algo que no sucede con el K-MEANS. Además, no todos los agrupamientos han de tener el mismo número de casos.

En definitiva, el enfriamiento simulado es mucho más lento que el sencillo algoritmo de las K medias y suele obtener peores resultados.

GRASP [Greedy Randomized Adaptive Search Procedure]

GRASP es una técnica de los años 80 que tiene como objetivo resolver problemas difíciles en el campo de la optimización combinatoria. Esta técnica dirige la mayor parte de su esfuerzo a construir soluciones de alta calidad que son posteriormente procesadas para obtener otras aún mejores.

Los algoritmos GRASP son algoritmos de tipo iterativo en los que cada iteración incluye una fase de construcción de una solución y otra de postprocesamiento en la cual se optimiza la solución generada en la primera fase.

Se puede establecer una analogía con el problema de la Programación Lineal, donde primero se construye una solución factible y después se aplica el algoritmo Simplex. Sin embargo, en GRASP se le da bastante importancia a la calidad de la solución generada inicialmente.

La estructura básica de un algoritmo GRASP es la siguiente:

```
Mientras no se satisfaga el criterio de parada
```

```
    Construir una solución greedy aleatoria
```

```
    Aplicar una técnica de búsqueda local a la solución greedy  
    aleatoria obtenida en el paso anterior (para mejorarla)
```

```
    Actualizar la mejor solución encontrada
```

Se puede extender este algoritmo si se le añade un operador de mutación (mecanismo de generación de vecinos) al procedimiento GRASP básico:

```
Mientras no se satisfaga el criterio de parada
```

```
    Solución = Solución greedy aleatoria
```

```
    Para i de 1 a L
```

```
        Solución = Búsqueda local (Solución)
```

```
        Actualizar la mejor solución (si corresponde)
```

```
        Solución = Mutación(Solución)
```

Algoritmo greedy

Como algoritmo greedy se puede utilizar una versión simplificada del algoritmo de Batchelor y Wilkins. Como centros de los agrupamientos se escogerán patrones del conjunto de entrenamiento de forma que el patrón escogido en cada momento sea el más alejado a los centroides ya fijados (siendo la distancia a un conjunto de centroides el mínimo de las distancias a cada centroide). Obviamente, el primer centroide ha de escogerse de forma aleatoria.

Mecanismo de generación de soluciones greedy aleatorias

En la construcción de soluciones greedy aleatorias se utiliza una lista de candidatos [*RCL: Restricted Candidate List*] en la que se incluyen los mejores aspirantes a formar parte de la solución. De esa lista se escoge un candidato aleatoriamente.

Un posible algoritmo greedy aleatorio consiste en utilizar como RCL la lista de los patrones más alejados a los centroides ya escogidos. El tamaño de esta lista puede ser, por ejemplo, igual al 5% de las muestras disponibles.

El algoritmo greedy aleatoria es análogo al algoritmo greedy clásico, teniendo en cuenta que, en cada momento, se escoge aleatoriamente uno de los patrones más alejados al conjunto de centroides ya establecido.

Técnica de búsqueda local

La estrategia de búsqueda local se utiliza para mejorar la solución obtenida mediante en el mecanismo de generación de soluciones greedy aleatorias. Mientras la solución no sea un óptimo local, se encuentra una solución mejor entre los vecinos de la solución actual.

Como técnica de búsqueda local se puede emplear, por ejemplo, el conocido algoritmo de las K medias, cuya convergencia depende de la configuración inicial de los clusters.

Operador de mutación (algoritmo GRASP extendido)

En el algoritmo GRASP extendido es necesario considerar un operador de mutación que se aplicará sobre la solución actual cada vez que finalice la ejecución del algoritmo de búsqueda local.

Una solución se representa por un vector en el cual la componente i -ésima indica el agrupamiento al que pertenece el patrón i -ésimo. Para mutar una solución se altera cada componente del vector solución con una probabilidad P . Como es lógico, las componentes del vector se mantendrán con una probabilidad $1-P$. Como valor de P escogeremos 0.3 para aplicar una mutación fuerte que permita una buena exploración del espacio de búsqueda.

Pseudocódigo del operador de mutación

```
flag = falso

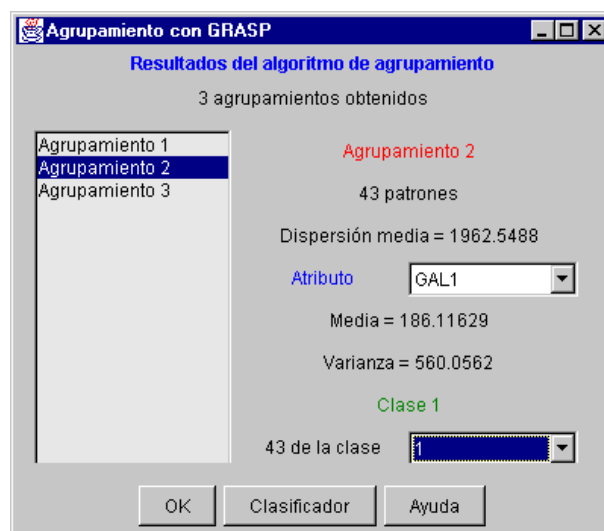
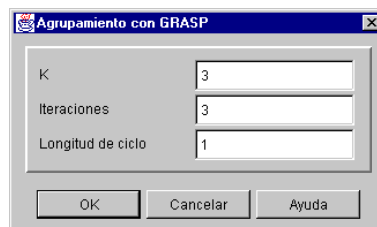
Mientras flag = falso

    Para i de 1 a n

        Generar un número aleatorio  $u \sim U(0,1)$ 

        Si  $u > P$ 
            flag = true
             $S_i = \{ c \mid 1 \leq c \leq C, c \neq a_i \}$ 
            Seleccionar aleatoriamente un elemento  $c$  de  $S_i$ 
            Asignar el patrón  $i$  al cluster  $c$  [ $a_i := c$ ]
```

Los algoritmos GRASP no requieren estructuras de datos especialmente complejas y conducen siempre a buenas soluciones. De hecho, con la base de datos de la galaxia espiral, con poner ejecutar el algoritmo unas pocas iteraciones (sin necesidad de utilizar el operador de mutación) se consigue siempre la división del conjunto de datos en tres agrupamientos que corresponden a las tres clases del problema:



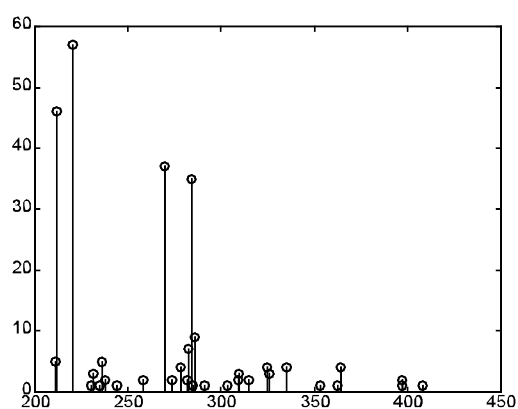
A continuación se mostrarán los valores obtenidos para distintos parámetros a lo largo de las iteraciones de un algoritmos GRASP. Para ilustrar la distribución de estos valores se utilizarán cinco particiones distintas de la **base de datos TITANIC** (por su simplicidad) y el **algoritmo GRASP básico** (sin operador de mutación).

La tabla siguiente recoge el mínimo, el máximo y la moda de las distribuciones de cada uno de los parámetros para las ejecuciones del algoritmo GRASP básico aplicadas sobre la tabla TITANIC cuando el número inicial de clusters es igual a 5 ($K=5$). Recuérdese que el valor del parámetro J (la suma de las distancias al cuadrado) se está intentando minimizar:

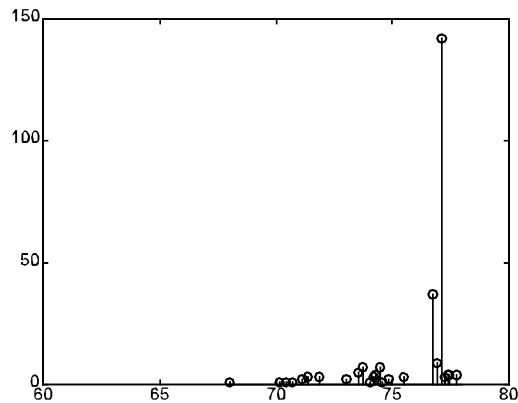
Datos	J			%TRA			%TST		
	min	mod	max	min	mod	max	min	mod	max
1	211.21	220.58	408.42	60.03	77.13	77.78	68.43	78.55	79.61
2	190.13	223.51	410.61	68.16	77.39	78.95	66.16	77.95	79.31
3	188.33	221.61	419.07	68.10	77.91	79.53	68.28	76.74	77.95
4	179.94	221.87	399.60	68.29	77.19	78.62	67.82	78.40	80.06
5	188.67	224.86	424.28	68.23	77.71	78.69	66.01	77.19	78.25

Se puede apreciar a partir de los datos de esta tabla que la moda de las distribuciones de los distintos parámetros suele encontrarse muy cerca del máximo obtenido. Esta moda corresponde con la solución que se obtendría con un algoritmo greedy simple (seguido de la técnica de búsqueda local correspondiente). La inclusión de cierta aleatoriedad en el algoritmo GRASP permite obtener de vez en cuando soluciones mejores a costa de obtener también algunas soluciones peores que las que se obtendrían con el algoritmo simple.

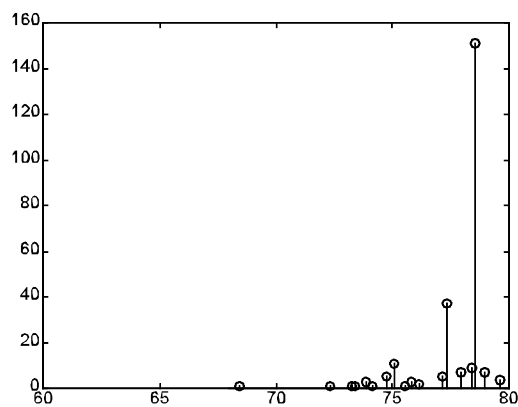
Los diagramas siguientes, contruidos utilizando MATLAB, muestran las distribuciones de los distintos valores:



Distribución de J



Porcentaje de clasificación en el conjunto de entrenamiento



Porcentaje de clasificación en el conjunto de prueba

✓ Las metaheurísticas como la utilizada en la técnica GRASP permiten mejorar las soluciones obtenidas en problemas complejos. Los algoritmos GRASP consiguen mejorar los resultados obtenidos por el algoritmo de las K medias para el problema del agrupamiento. GRASP consigue soluciones de alta calidad.

✓ El uso adecuado de la lista de candidatos RCL (Restricted Candidate List) es esencial para el algoritmo GRASP, ya que de esa forma se consiguen distintos puntos de partida para el algoritmo de búsqueda local concreto, el K-Medias en nuestro caso. Dado que este algoritmo específico del problema es altamente dependiente de la configuración inicial de los clusters, el método de construcción de soluciones greedy aleatorias debe proporcionar buenos puntos de partida en distintas regiones del espacio de búsqueda (lo que consigue con la RCL).

✓ En cuanto al algoritmo greedy empleado, se puede apreciar que no obtiene soluciones excesivamente buenas aunque sí proporciona buenos puntos de partida para un algoritmo de búsqueda local como el algoritmo de las K Medias, el cual obtiene resultados netamente superiores a los que obtenía cuando se partía de una configuración inicial aleatoria.

Algoritmo de agrupamiento secuencial

La utilización del método de agrupamiento secuencial es algo más compleja que la de algoritmos como el K-Means o el de Batchelor y Wilkins. Igual que ellos, realiza cálculos sencillos, procesa los patrones secuencialmente y los resultados obtenidos depende del orden de presentación de los patrones (está sesgado por los primeros patrones).

Su funcionamiento puede ajustarse gracias a un amplio conjunto de parámetros. Los valores adecuados para los parámetros son difíciles de establecer a priori, por lo que se suele emplear un mecanismo de prueba y error.

Parámetro	Descripción
K	Máximo número de agrupamientos
R	Umbral de distancia para crear agrupamientos
C	Umbral de distancia para mezclar agrupamientos
M	Longitud del lote: Número de patrones considerados entre procesos de mezcla)
T	Umbral para la eliminación de agrupamientos (porcentaje respecto a M)

El algoritmo selecciona arbitrariamente el centro del primer agrupamiento. Posteriormente procesa secuencialmente los demás patrones. Calcula la distancia del patrón actual al agrupamiento más cercano (a su centroide). Si ésta es menor o igual a R se asigna el patrón a su agrupamiento más cercano. En caso contrario, se crea un nuevo agrupamiento con el patrón actual.

Cada M patrones, se mezclan agrupamientos por cercanía (se mezclan dos agrupamientos si la distancia entre ellos es menor que C). Si, tras la mezcla por cercanía, quedan más agrupamientos que los deseados por el usuario (K), se mezclan los agrupamientos por tamaño (se mezclan los agrupamientos con menos del T% de M miembros con sus agrupamientos más cercanos). Si aún quedan demasiados agrupamientos, se mezclan los agrupamientos más cercanos hasta obtener el número deseado de agrupamientos K.

El proceso de mezcla nos asegura que al final obtenemos el número deseado de agrupamientos y no más (como solía suceder en el método adaptativo o en el algoritmo de Batchelor y Wilkins).

Algoritmo ISODATA

ISODATA es el acrónimo de *Iterative Self-Organizing Data Analysis Techniques* (con la A añadida para hacer pronunciable el nombre), un iterativo método de agrupamiento que, como ya sucedía con el método de agrupamiento secuencial, requiere un considerable esfuerzo para ajustar adecuadamente todos sus parámetros. Además, éstos pueden modificarse en cada iteración del algoritmo.

Parámetro	Descripción
K	Número deseado de agrupamientos
A	Número inicial de agrupamientos
n	Umbral del número de patrones para la eliminación de agrupamientos
s	Umbral de desviación típica para la división de un agrupamiento
c	Umbral de distancia para la unión de agrupamientos
L	Máximo número de mezclas en una iteración
I	Máximo número de iteraciones permitidas

Inicialmente se seleccionan los centros de A agrupamientos.

En cada iteración

Se fijan los valores de los distintos parámetros del algoritmo

Se asigna cada patrón al agrupamiento más cercano

Se eliminan los agrupamientos con menos de n patrones

Si el número actual de agrupamientos es pequeño (menor o igual que $K/2$), dividimos los agrupamientos más dispersos (siendo la dispersión de un agrupamiento la distancia media de sus patrones al centroide del cluster) por la componente de máxima dispersión (respetando el umbral mínimo s).

En las iteraciones pares o cuando el número actual de agrupamientos es elevado ($> 2K$), unimos como máximo L pares de agrupamientos cuya separación entre ellos quede por debajo del umbral de distancia c

Métodos basados en grafos: Matriz de similitud

El principal inconveniente de la mayor parte de los algoritmos heurísticos es su dependencia del orden en que se le presentan los patrones. Los métodos basados en grafos, igual que los algoritmos GRASP, intentan evitar este hecho pero su coste computacional los hace inaplicables en muchas ocasiones.

La matriz de similitud

La matriz de similitud se emplea para mostrar el grado de similitud entre un conjunto de patrones. Se construye una matriz S simétrica de tamaño $N \times N$, siendo N el número de patrones del conjunto de entrenamiento. $S[i,j]$ toma el valor 1 si la distancia entre los patrones i y j queda por debajo de un umbral preestablecido θ . En caso contrario, $S[i,j]$ vale 0. Por lo tanto, con un bit por celda podemos almacenar la matriz de similitud.

Agrupamiento basado en la matriz de similitud



Mientras queden patrones en la matriz de similitud S

Seleccionar la fila i de la matriz de similitud S que contenga más unos. Si hay varias, escoger una al azar.

Crear un agrupamiento con los patrones j tales que $S[i,j] = 1$

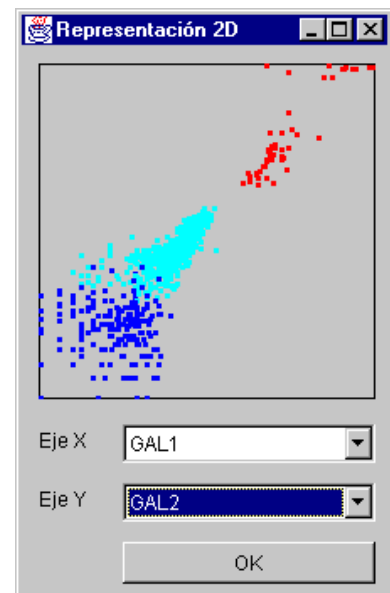
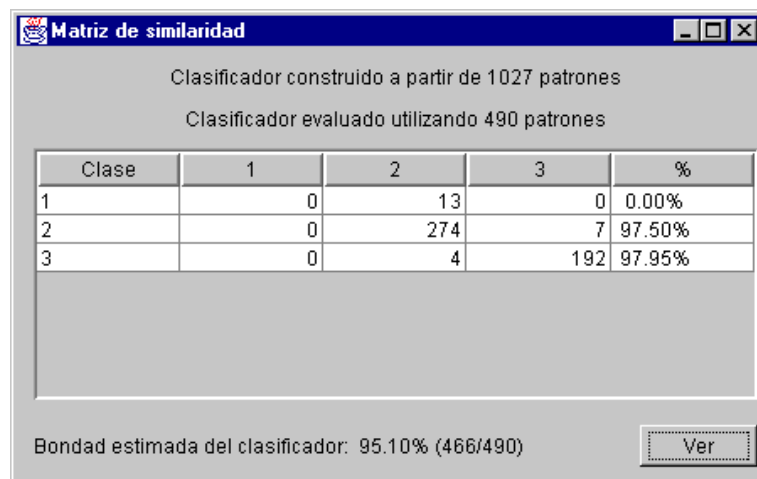
Añadir al agrupamiento todos aquellos patrones k tales que $S[j,k] = 1$, siendo j un patrón incluido en el nuevo agrupamiento hasta que no se puedan añadir más patrones a dicho agrupamiento.

Reducir la matriz de similitud: Eliminar de S todas las filas y columnas correspondientes a patrones incluidos en el agrupamiento recién creado.



Ejemplo: Galaxia espiral

Umbral de distancia	Agrupamientos	Bondad estimada
100	86	92.04 %
500	10	95.10 %
1000	4	93.06 %
1500	3	91.83 %
2000	1	57.34 %



Resultado obtenido con el umbral de distancia igual a 500

El método de agrupamiento basado en la matriz de similitud produce un resultado curioso, si no sorprendente, cuando es aplicado a la base de datos de la galaxia espiral. A partir de un umbral de distancia relativamente pequeño, no se consigue ningún agrupamiento correspondiente a la clase 1 debido a la distribución de las muestras en el conjunto de entrenamiento (que se puede visualizar en el Analizador Numérico tal como se muestra en la imagen de la derecha).

Detalles de implementación

Todos los algoritmos han sido implementados en Java y pueden ejecutarse desde la línea de comandos o desde la interfaz gráfica integrada denominada *NC* (*Analizador Numérico* en castellano, *Numerical Cruncher* en inglés).

Existen versiones del Analizador Numérico para el JDK1.1 (que requiere la disponibilidad de las Java Foundation Classes, JFC 1.1, SWING 1.0.1 o superior) y para el JDK1.2. El JDK1.2 consume más recursos pero incluye un compilador JIT [*Just-In Time*] que acelera notablemente la ejecución de los algoritmos.

A continuación se ofrece una breve descripción de los distintos paquetes y ficheros en los que se ha estructurado el código fuente:

<i>Paquete</i>	<i>Descripción</i>	<i>Líneas</i>
fbg.gui	Rutinas para la interfaz gráfica	884
fbg.gui.html	Visualizador HTML	529
fbg.tda	TDAs	973
fbg.util	Utilidades	256
rbg.util.db	Acceso a bases de datos	3110
fbg.util.math	Rutinas matemáticas	643
rf	Rutinas genéricas	1302
rf.classification	Clasificadores	124
rf.classification.lvq	Métodos de aprendizaje adaptativo	1075
rf.classification.nn	Clasificadores k-NN	417
rf.classification.parametric	Clasificadores paramétricos	1041
rf.clustering	Métodos de agrupamiento	2195
rf.data	Acceso a los datos	1888
rf.distance	Métricas de distancia	421
rf.edit	Algoritmos de edición y condensado	687
rf.gui	Interfaz gráfica (NC)	6681
		22226

<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
BarChart.java	Diagrama de barras	328
GUI.java	Configuración del interfaz	100
Message.java	Mensajes	16
test.java	Programa de prueba	130
XYConstraints.java	Auxiliar de XYLayout	112
XYLayout.java	Layout Manager	198
		884

<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
HTML.java	Panel HTML	401
HTMLFrame.java	Ventana HTML	128
		529

<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
Arbol.java	TDA Árbol / Bosque	268
ArbolBinario.java	TDA Árbol Binario	157
Cola.java	TDA Cola	85
Lista.java	TDA Lista	125
NodoArbol.java	Nodo de un árbol	70
NodoLista.java	Nodo de una lista	28
Pila.java	TDA Pila	75
test.java	Programa de prueba	165
		973

<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
Benchmark.java	Medición de tiempos	67
Environment.java	Entorno (vg. idioma)	41
Help.java	Interfaz estándar de ayuda	15
Output.java	Interfaz estándar de salida	12
Text.java	Textos (ASCII)	121
		256

<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
Campo.java	Atributo de una tabla	98
CampoCHAR.java	Atributo de tipo CHAR	78
CampoNUMBER.java	Atributo de tipo NUMBER	132
Clustering.java	Métodos de agrupamiento	474
Conexion.java	Conexión JDBC	51
Dominio.java	Dominio de un atributo	35
DominioCHAR.java	Dominio (atributo de tipo CHAR)	32
DominioNUMBER.java	Dominio (atributo de tipo NUMBER)	44
Dominios.java	Contenedor de dominios	460
SyntheticData.java	Generador de datos sintéticos	83
Tabla.java	Tabla	453
Thresholding.java	Umbralización automática	895
Tupla.java	Tupla de una tabla	81
Valor	Valor	28
Valores	Contenedor de valores	166
		3110

<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
Matrix.java	Cálculo matricial	643
		643

<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
ConfigFile.java	Fichero de configuración genérico	273
DataConfiguration.java	Configuración de los datos	547
ImageViewer.java	Visualizador de imágenes	244
Statistics.java	Estadísticas	238
		1302

<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
Classifier.java	Clasificador	124
		124

<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
dsm.java	Decision Surface Mapping	363
lvq1.java	LVQ-1	368
olvq1.java	OLVQ-1	344
		1075

<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
kNNClassifier	Clasificador k-NN	286
NNClassifier	Clasificador 1-NN	131
		417

<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
EuclideanClassifier	Clasificador lineal Distancia euclídea	204
MahalanobisClassifier	Clasificador lineal Distancia de Mahalanobis	222
Normalized EuclideanClassifier	Clasificador lineal Distancia euclídea normalizada	234
ParallelepipedClassifier	Método de los paralelepípedos	200
QuadraticClassifier	Clasificador cuadrático	181
		1041

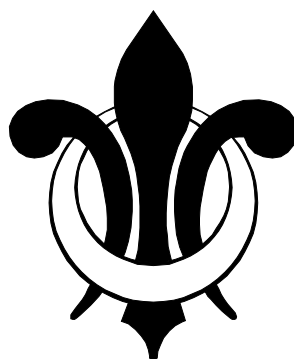
<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
AdaptiveClustering	Agrupamiento adaptativo	158
BatchelorWilkins	Algoritmo de Batchelor y Wilkins	186
Cluster	Modelo de cluster	337
ClusteringAlgorithm	Modelo de algoritmo de agrupamiento	109
GRASPClustering	Algoritmo GRASP	385
ISODATA	Algoritmo ISODATA	379
kMeans	Algoritmo de las K Medias	164
SequentialClustering	Agrupamiento secuencial	240
SimilarityMatrix	Algoritmo basado en la matriz de similaridad	237
		2195

<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
AsciiData	Datos en formato ASCII	345
AsciiDataset	Conjunto de datos ASCII	139
Data	Modelo de datos	96
Dataset	Modelo de conjunto de datos	94
ImageBand	Banda de una imagen	255
ImageData	Datos en formato IMAGE	264
ImageDataset	Conjunto de datos de imágenes	157
Pattern	Patrón	143
TableData	Datos de una base de datos (JDBC)	244
TableDataset	Conjunto de datos (acceso vía JDBC)	151
		1888

*Menú principal del Analizador Numérico*

<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
Distance	Modelo de distancia	37
EuclideanDistance	Distancia euclídea	75
MahalanobisDistance	Distance de Mahalanobis	108
Normalized EuclideanDistance	Distancia euclídea normalizada	111
Weighted EuclideanDistance	Distancia euclídea ponderada	90
		421

<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
Hart	Condensado de Hart	188
MultiEdit	Multiedición	161
PartitionEdition	Edición por particiones	179
WilsonEdition	Edición de Wilson	159
		687



<i>Fichero</i>	<i>Descripción</i>	<i>Líneas</i>
AboutBox	Acerca de...	116
Chart2D	Diagrama 2D	150
ColorMap	Mapa de colores	30
Dialog2INT	Diálogo estándar (2 enteros)	174
DialogAdaptiveClustering	Diálogo agrupamiento adaptativo	311
DialogApriori	Diálogo ¿usar probabilidades a priori?	106
DialogBatchelorWilkins	Diálogo agrupamiento B&W	299
DialogGRASPClustering	Diálogo algoritmo GRASP	351
DialogISODATA	Diálogo algoritmo ISODATA	422
DialogK	Diálogo estándar (1 entero)	138
DialogKMeans	Diálogo algoritmo de las K Medias	300
DialogSequentialClustering	Diálogo agrupamiento secuencial	383
DialogSimilarityMatrix	Diálogo agrupamiento basado en grafos	300
Frame2D	Visualización 2D	168
FrameClasses	Estadísticas por clases	290
FrameClassifier	Evaluación de clasificadores	388
FrameClusters	Evaluación de métodos de agrupamiento	613
FrameImage	Visualización de imágenes	106
FrameStats	Estadísticas globales	291
Logo	Emblema UGR	89
MainMenu	Menú principal	1248
NumericalCruncher	Programa principal	408
		6681