SISTEMAS
INFORMÁTICOS
2006 / 2007

CLASIFICACIÓN DE TEXTURAS NATURALES MEDIANTE TÉCNICAS DE VISIÓN POR COMPUTADOR

AUTORES:

Javier Vizcaíno Lamas
Álvaro Toledo Gil
Héctor Marco Rubio

Dirigido por:

Prof. Gonzalo Pajares Martinsanz.
Dpto. Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

Resumen del proyecto.....4
Resumen..... 4
Resumen en inglés.....4
Palabras clave.....5
Autorización.....5

Agradecimientos.....5

1 Introducción.....6

1.1 Justificación del proyecto.....6
1.2 Objetivos propuestos.....7
1.3 Organización.....8
1.4 Organización de la memoria.....9

2 Especificación detallada.....10

2.1 Análisis preliminares.....10
2.2 Requisitos.....10
2.3 Planificación y coordinación.....12
2.4 Problemática.....16
2.5 Análisis y planificación de riesgos.....17

3 Descripción de los métodos de clasificación.....20

3.1 Plataformas de desarrollo.....20
3.2 Algoritmos empleados.....20
3.2.1 Algoritmo de Lloyd.....20
3.2.1.1 Idea general.....20
3.2.1.2 Especificación formal.....20
3.2.1.2.1 Inicialización de los centros.....20
3.2.1.2.2 Entrenamiento.....21
3.2.1.2.3 Clasificación.....22
3.2.1.3 Análisis de parámetros.....23
3.2.2 Algoritmo C-Medias Fuzzy.....24
3.2.2.1 Idea general.....24
3.2.2.2 Especificación formal.....24
3.2.2.2.1 Fuzzy clustering..... 24
3.2.2.2.2 Entrenamiento.....26
3.2.2.2.3 Clasificación.....27
3.2.2.3 Análisis de parámetros.....27

3.2.3 Algoritmo Bayesiano.....28
3.2.3.1 Idea general.....28
3.2.3.2 Especificación formal.....28
3.2.3.3 Análisis de parámetros.....31

4 La interfaz gráfica a partir de los algoritmos.....32

4.1 Plataformas de desarrollo.....32
4.2 Evolución por mejoras y necesidades.....33
4.3 Interfaz final.....38
4.3.1 Diagramas UML.....42

5 Resultados.....48

5.1 Pruebas de test.....48
5.2 Cumplimiento de objetivos.....50

6 Conclusiones y perspectivas de futuro.....59

7 Bibliografía.....60

Apéndice I. Breve manual de usuario.....61

Apéndice II. Contenido del CD.....61

Resumen del proyecto

Resumen

El Proyecto que se presenta forma parte de un proyecto más amplio con dos partes bien definidas: 1) implementación de métodos de clasificación de texturas naturales en imágenes digitales y 2) diseño de un interfaz gráfico de comunicación hombre-máquina (IHM). El primer aspecto ha sido abordado por un grupo de trabajo distinto a éste. El segundo es abordado por el grupo de trabajo que presentamos esta memoria.

Uno de los aspectos más relevantes a destacar estaba en el hecho de la necesidad de integración de ambos trabajos en un único proyecto a modo de imitación a lo que realmente supone un proyecto real en la industria.
El IHM recoge los datos procedentes del usuario para proceder a la clasificación por los tres métodos implementados (Fuzzy clustering, Lloyd, clasificador Bayesiano) por este motivo describimos los métodos con el fin de asimilar los parámetros seguidos por cada método.

Tras el proceso el IHM trasfiere los resultados al usuario.

La planificación y los detalles del proceso de desarrollo se detallan más adelante en el apartado de “Especificación “, en el que se ha prestado una especial atención a la parte gráfica.

Abstract

This proyect’s part of a bigger one with two well defined parts: 1) Methods of natural textures used in digital images, 2) design of an interface graphic in the interface man-machine communication.

Item 1) has been approached by a work group different from this one.
Item 2) is approached by the work group introducing this memory.

One of the most relevant aspects was the necessity of integrating both works in one project as a means of an imitation to what a real one is supposed to be in industry.

The MMI (Man Machine Interface) selects data from the users in order to classify the 3 methods to be carried out (Fuzzy clustering, Lloyd, and Bayesian algorithm). That’s why we describe the different methods in order to assimilate the parameters followed by each method. After the process the MMI transfers the results to the user. Planning details are described and shown in detail in “Specifications” where we have paid special attention to the Graphs.

Palabras clave

- Clasificación
- Texturas
- Lloyd
- Fuzzy
- Bayesiano
- JAI

Autorización

Autorizamos a la facultad de Informática de la Universidad complutense de Madrid, así como al resto de los centros adscritos a la Universidad Complutense de Madrid a la utilización y modificación de todos los componentes que forman este proyecto. Siempre y cuando no se derive en una utilización comercial de cualquiera de los componentes o de parte de ellos.

Javier Vizcaíno Lamas

Álvaro Toledo Gil

Héctor Marco Rubio

Agradecimientos

Queremos mostrar ante todo nuestro más profundo agradecimiento al director del proyecto y profesor de la Facultad de Informática de la Universidad Complutense de Madrid, Don Gonzalo Pajares Martinsanz, por aceptarnos en su proyecto para la asignatura de Sistemas Informáticos y por la orientación que nos ha dado, pero sobre todo, querríamos agradecerle el trato personal que ha tenido con nosotros, y esa libertad que nos ha hecho sentir que realmente realizábamos un proyecto tutorado y no una práctica más.

Y por supuesto nuestro agradecimiento a Ramón, Antonio y "Guanchi", integrantes del grupo que desarrollo los algoritmos que han hecho posible que este proyecto saliera adelante, por su colaboración y buena disposición para trabajar conjuntamente.

1 Introducción

1.1 Justificación del proyecto

Actualmente existe un creciente interés en el desarrollo de procedimientos para el tratamiento de las texturas, siendo la clasificación un tema de especial relevancia.

Este interés tiene su fundamento en algunos aspectos tales como:

1) Control de cultivos en agricultura, propiciado por la necesidad de conocer los cultivos programados para la recepción de subvenciones u otros aspectos relacionados.

2) Cómputo y medición de parcelas agrícolas y tipo de cultivos a los que se dedica.

3) Control de riegos agrícolas.

4) Agricultura reprecisión para aplicar herbicida en el tratamiento de malas hierbas de forma selectiva evitando la contaminación medioambiental y la reducción de costes de producción.

5) Evaluación de catástrofes naturales: fuegos, daños por inundaciones, heladas en cultivos agrícolas, nevadas, etc.

6) Detección de cambios en determinadas zonas, principalmente urbanas para el control de edificaciones o impactos medioambientales.

7) Vigilancia en prevención de catástrofes, por ejemplo fuegos o inundaciones.

8) Control de fenómenos meteorológicos como es el retroceso de determinadas playas.

9) Vigilancia: forestal, marítima.

10) Detección de infraestructuras: carreteras, caminos forestales, cañadas reales, etc.

En este sentido, diversas empresas u organismos desarrollan o utilizan aplicaciones para abordar dicha problemática. Por citar sólo algunas podemos mencionar

1) *Digital Image Processing* (Dimap) (http://www.dimap.es/). Las imágenes utilizadas en el proyecto son cortesía del Servicio Territorial de Galicia (SITGA) proporcionadas por Dimap y pertenecientes a la región de Abadín (Lugo) capturadas mediante vuelos aéreos.

2) *Proespacio* (http://www.proespacio.org/) agrupación de empresas del sector aeroespacial donde una de las actividades destacables son aplicaciones mediante el uso de imágenes de satélite. En este consorcio destacan algunas empresas líderes del sector tanto en España como a nivel Internacional: EADS-Espacio, EADS Astrium, CRISA, GMV, Indra Espacio, Sener, Hispasat, IberEspacio, Inasmet, Insa, Mier, NTE, Tecnológica, Rymsa, Hispasat, GTD, Alcatel, CRISA, GTD.

3) Organismos oficiales y Centros de Investigación
   - Consejo Superior de Investigaciones Científicas (CSIC)
   - Instituto Nacional de Técnica Aeroespacial (INTA)
   - Centro de Estudios y Experimentación de Obras Públicas (CEDEX), con el que existen trabajos de colaboración previos por parte de uno de los directores del trabajo (Pajares y col. 2001, Pajares y col. 2002).

La mayoría de las empresas e instituciones anteriormente mencionadas utilizan para llevar a cabo sus aplicaciones principalmente herramientas comerciales tales como:

- ERDAS
- Intergraph
- ENVI-IDL
- ILOG
- PCI
- E-Cognition

Aunque bien es cierto que cada día es mayor la potencialidad de las herramientas mencionadas no es menos cierto que los retos tecnológicos derivados de las aplicaciones mencionadas hacen que en algunos casos la utilización de tales herramientas sea insuficiente para abordar las propuestas de proyectos demandados por los clientes. Este es el caso para una gran parte de las aplicaciones mencionadas en el tratamiento de las texturas naturales donde la **clasificación** de las mismas surge como una tarea fundamental.

En efecto, en general las mencionadas herramientas implementan algunos de los métodos clásicos de clasificación, siendo necesaria la intervención del usuario mediante programación para abordar algunas de las tareas que involucran aspectos de clasificación de texturas, lo cual no siempre es factible o al menos en la medida que cabría esperar debido a múltiples limitaciones.

Además, y lo que es más importante, en muchos casos no existe la posibilidad de llevar a cabo la investigación necesaria para abordar la problemática particularmente cuando los métodos clásicos no producen los resultados esperados.

Por todo lo expuesto anteriormente surge una necesidad importante en el ámbito de las aplicaciones reales para abordar el tema de la clasificación de texturas naturales en imágenes y un reto para la comunidad científica para tratar de mejorar los procedimientos existentes con la mayor flexibilidad posible.

1.2 Objetivos propuestos

- Desarrollo de un interfaz IHM flexible y amigable para la clasificación de texturas naturales en imágenes digitales a modo de ejemplo de las comercialmente existentes
- Coordinar las tareas entre dos grupos de desarrollo
- Aplicar las técnicas de Ingeniería del Software para la realización del diseño (análisis diseño, implementación, integración y pruebas)

- Conseguir un diseño modular y flexible para garantizar su fácil mantenibilidad y actualización, así como su continuidad de futuro.

En resumen, nuestro objetivo era el de poner una capa por encima de estos algoritmos clasificadores, de modo que el usuario pudiese utilizar el programa de una forma intuitiva y agradable.

Para ellos se utilizaron botones en la barra principal, de modo que por medio de los dibujos (iconos) el usuario podía utilizar la aplicación, aun cuando no supiese ni siquiera de la existencia de los algoritmos o su funcionamiento, ya que se adjunta una pequeña pero concisa ayuda en forma de globos.

1.3 Organización

Para la elaboración de la aplicación contaríamos con dos grupo, por lo que además de la coordinación interna tendríamos que cuidar mucho la colaboración entre los grupos para no crear esperas ni sobrecargas. Es por esto que se decidió emplear los conocimientos de Ingeniería del Software que se detallarán más adelante.

Como todo proyecto de cierta dimensión, su desarrollo ha estado dividido en varias fases:

- La primera, como es obvio, se dedicó tanto al planteamiento como a la posterior documentación que se requería. La inmensa mayoría de los conocimientos necesarios para el desarrollo del proyecto eran desconocidos para todos los integrantes del grupo.

- La segunda fase se dedicó al diseño, tanto de la interfaz gráfica como de los algoritmos involucrados. Se especificaron tres algoritmos distintos, lo que habría que tener en cuenta para el diseño.
  La interfaz que se obtuvo en esta fase no era ni mucho menos la definitiva, pero ya tenía soporte para toda la funcionalidad que requería la aplicación. Se tenía en ese momento una interfaz funcional.

- La tercera fase fue la realmente importante. En ella se fueron integrando los distintos algoritmos con la interfaz gráfica, al mismo tiempo que esta iba cambiando su aspecto e incrementando su funcionalidad. Fue sin duda la fase en la que más cambios se realizaron y de la que nació la versión que poco más tarde, tras los retoques finales, daríamos como definitiva.

- La cuarta fase fue la dedicada a las pruebas y mejoras. Se realizó un ciclo de pruebas en el que se probaba cada mejora por separado y después se realizaban las pruebas de integración.
  En esta fase se mejoró el aspecto amigable del IHM.

1.4 Organización de la memoria

En esta primera sección se han presentado los aspectos generales que motivan el proyecto junto con los objetivos planteados.

En la sección 2 se especifican los detalles del proyecto a desarrollar comenzando por los requisitos, la planificación y los riesgos previstos inicialmente.

En la sección 3 se describen los tres métodos seleccionados para la clasificación (Fuzzy, Lloyd y clasificador Bayesiano); estos métodos determinan los parámetros requeridos y que deben ser contemplados por el IHM como mecanismo de comunicación.

En la sección 4 se describen los detalles y especificaciones relativas al IHM, que constituye el núcleo del presente proyecto.

En la sección 5 se muestran los resultados de las pruebas de integración realizadas. En la sección 6 se establecen las conclusiones más relevantes junto con las perspectivas de futuro. Tras esta sección se incluyen la bibliografía, un breve manual de usuario y el contenido del CD.

2 Especificación detallada

2.1 Análisis preliminares

Se parte de unos supuestos previos en relación a los conocimientos adquiridos durante nuestra formación, entre ellos destacan aspectos tales como:

- análisis de requisitos
- ingeniería del software
- entornos de programación de propósito general: JAVA, C++
- entornos de programación específicos: Matlab
- planificación de proyectos

Por otra parte, el grupo carece de unos conocimientos específicos en el tema relativo a la clasificación de las texturas naturales en imágenes digitales por lo que en esta fase preliminar nos planteamos la posibilidad de encontrar software de propósito general dedicado específicamente al tratamiento de imágenes. En este sentido el paquete JAI (Java Advanced Images) suministrado por Sun Mycrosystems, cumple suficientemente con esta primera necesidad.

Se opta, conjuntamente con el otro grupo, por elegir el lenguaje JAVA como el más apropiado para la finalidad propuesta. Por otra parte existen en JAVA paquetes de tratamiento matemático como JAMA, que permite minimizar el desarrollo de funciones matemáticas complejas.

2.2 Requisitos

En la primera etapa del proyecto se empleo la mayoría del esfuerzo en identificar tanto los requisitos del sistema, como del software. Para ello el trabajo se dividió en distintas tareas:

- Información útil de la aplicación.
- Funcionalidades contempladas.
- Establecer interfaces específicos.
- Estudios de comportamiento de la aplicación.

Para llevar a cabo estas tareas se aplicaron técnicas de planificación de software, ingeniería de la información, documentación y análisis de requisitos.

Una vez realizadas estas tareas era momento de pasar a la etapa de desarrollo, que estuvo basada en la definición de los siguientes criterios:

- Estructuras de datos a utilizar.
- Diseño e implementación de las interfaces.
- Implementación de las funcionalidades.
- Pruebas incrementales.

Esta etapa estuvo a su vez dividida en varias subetapas que debían ser sucesivas, a saber:

- Diseño de la aplicación.
- Implementación del código que diese soporte al diseño.
- Pruebas que mostrasen el buen funcionamiento del código.

Los requisitos que se recogieron para este proyecto fueron de dos tipos: funcionales y operacionales.

Requisitos funcionales:

Recogemos a continuación los requisitos funcionales para los distintos casos de uso.

- Abrir imagen: el usuario debe poder abrir imágenes desde un archivo y además debe poder seleccionar una de ellas para que sea tratada
- Algoritmo de Lloyd: el usuario mediante un botón puede seleccionar el método de Lloyd para tratar la imagen.
- Algoritmo de Fuzzy: el usuario mediante un botón puede seleccionar el método de Fuzzy para tratar la imagen.
- Algoritmo Bayesiano: el usuario mediante un botón puede seleccionar el método de Bayesiano para tratar la imagen.
- Aprendizaje: Una vez seleccionados los distintos métodos cada uno de ellos presenta la posibilidad de aprender de una imagen, aportada por el usuario, de la que obtendrá los centros con lo que se realizará la clasificación.
- Centros: El método de Lloyd permite al usuario elegir tres formas de obtener los métodos: de forma aleatoria, mediante el método de fuzzy (supervisado) y cargando centros previamente guardados.
- Clasificación: el usuario obtiene una imagen con la clasificación de las texturas de la imagen original que seleccionó, proporcionada por el método que el usuario escogió.
- Guardar imagen: el usuario puede guardar la imagen resultado de la clasificación en los formatos soportados.

Los requisitos operacionales son fundamentalmente de tipo tecnológico, ya que el tratamiento de imágenes es siempre una tarea que requiere una gran cantidad de trabajo, por lo que es necesario disponer de capacidad de cómputo y de suficiente memoria. Los requisitos operacionales se muestran a continuación y muestran el entorno tecnológico supuesto.

Requisitos operacionales:

- Procesador mínimo: Pentium 4 a 1,5 GHz o equivalente
- Memoria RAM mínima: 1024 MB.
- Máquina virtual de java (JVM) (xMin 128MB – xMax 1000 MB)
- JRE 1.5 (Java Runtime Environment)

2.3 Planificación y coordinación

Es en este apartado en el que se hizo uso muy particular de los conocimientos adquiridos sobre la ingeniería del software. Como se ha apuntado anteriormente la planificación no fue una tarea tan sencilla como puede parecer, puesto que en verdad se trataba de tres coordinaciones distintas: la coordinación de cada uno de los dos grupos, esto es, una para el desarrollo de la interfaz gráfica y otra para la de los algoritmos de clasificación y la coordinación que tenía que existir entre los dos grupos.

Uno de los aspectos al que se dio mayor importancia en la planificación general fue a las esperas, esto es, era necesario que ambos grupos cumpliesen con los plazos aproximados para evitar que un grupo estuviese esperando el trabajo del otro sin poder seguir avanzando.

Con esto queda claro que la planificación general estaba totalmente supeditada al buen funcionamiento de las coordinaciones locales de cada grupo, puesto que si en un grupo no se llegaba a tiempo el proyecto empezaba a acumular retrasos que a finales de mayo podrían convertirse en un verdadero contratiempo.

Se decidió optar por un modelo de espiral para la planificación global, dejando la elección de las planificaciones locales a cada uno de los grupos, teniendo en cuenta que las cargas de trabajo no estaban igualmente repartidas en uno y otro grupo.

En este trabajo dedicado al desarrollo de la interfaz gráfica se optó por un modelo de planificación en espiral, puesto que es un modelo de planificación que garantiza los criterios que previamente se habían marcado como preferentes dentro del grupo, a saber:

- Entendibilidad
- Fiabilidad
- Robustez
- Mantenibilidad
- Rapidez

Además, el modelo de espiral es un modelo de desarrollo flexible, y dado que nuestro proyecto requería de una construcción de esta naturaleza era el más adecuado. Era también muy atractiva la adaptabilidad que muestra ante los cambios de requisitos y sobre todo ante las especificaciones parciales, puesto que nuestro proyecto constaba de tres algoritmos que en principio actúan de manera independiente y que por lo tanto podían desarrollarse en distintos momentos. Una vez terminado un algoritmo este podía pasar a formar parte del proyecto parcial sin necesidad de esperar a la terminación del resto.

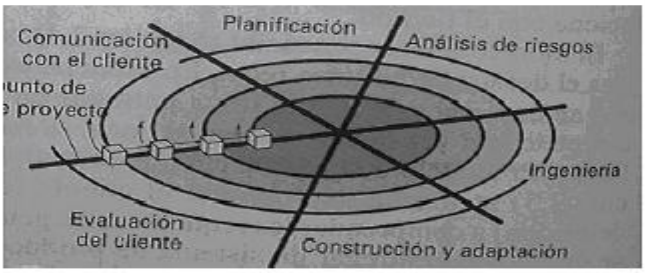
El modelo por el que finalmente nos decantamos es el conocido como “espiral de Boston”. Se basa en los siguientes principios teóricos:

- Comunicación con el cliente y recolección de requisitos
- Planificación
- Análisis de riesgos
- Ingeniería e implementación
- Evaluación por el cliente

El único inconveniente del modelo para no encajar perfectamente con nuestro proyecto era la falta de un cliente real. Pero tampoco fue un verdadero problema porque sustituimos a ese cliente real por uno virtual, el profesor director .Así la comunicación previa con el cliente se cambio por las exigencias que el profesor tenía para el proyecto, y la evaluación final por parte del cliente se cambió por una batería de pruebas internas y por supuesto, por la validación final por parte del profesor.

El ciclo evolutivo en el que se basa este modelo queda claramente reflejado en la gráfica siguiente:

MODELO DE PLANIFICACIÓN EN ESPIRAL DE BOSTON



Planificación detallada. Fases

Como ya se indicó en la descripción del proyecto, este se divide en cuatro fases claramente diferenciables. A continuación daremos más detalles sobre cada una de las fases, incluyendo las fechas aproximadas que comprende cada una, los objetivos que se pretendían abordar y el número de iteraciones previsto para alcanzar dichos objetivos.

FASE DE INICIO

Esta fase está compuesta por una iteración, y tiene las siguientes características:

Fecha de inicio: 10 – 10 - 2006
Fechas de conclusión: 1 – 11 - 2006

Objetivos:

- Concreción con el tutor del alcance y objetivos del proyecto.
- Análisis de riesgos.
- Captura de requisitos de acuerdo con los objetivos acordados con el profesor.
- Establecimiento del plan de fase del proyecto. La planificación fijada estará coordinada con la fijada por el otro grupo que participa en la aplicación.
- Elección del lenguaje de programación a utilizar para realizar el proyecto (JAVA). Dicha elección será llevada a cabo entre los dos grupos que estamos realizando la aplicación.
- Aprendizaje y familiarización con el lenguaje establecido así como con las bibliotecas específicas que se van a usar en la aplicación (JAI, JAMA).
- Selección de objetivos que se deben cumplir en la siguiente fase, en función de los resultados obtenidos en ésta.

FASE DE ELABORACIÓN.

Esta fase está compuesta por una iteración, y tiene las siguientes características:

Fecha de inicio: 2 - 11 - 2006
Fechas de conclusión: 1 - 12 - 2006

Objetivos:

- Diseño global de la aplicación. Este diseño se realizará de forma modular y genérica, facilitando así su integración en cualquier tipo de interfaz especializada, y en particular a la realizada por el otro grupo.
- Especificación de casos de uso y de los servicios prestados por la aplicación.
- Realización de un prototipo muy simple que garantice un soporte para la funcionalidad inicial y así comprobar el buen funcionamiento del diseño escogido.
- Elaboración de la documentación correspondiente a esta fase.
- Selección de objetivos que se deben cumplir en la siguiente fase, en función de los resultados obtenidos en esta.

FASE DE CONSTRUCCIÓN.

Esta fase tiene una duración mayor que las anteriores por ello está formada por dos iteraciones con las siguientes características:

ITERACIÓN 1

Fecha de inicio: 2 - 12 - 2006
Fechas de conclusión: 20 - 1 - 2007

Objetivos:

- Reunión con el tutor y los componentes del otro grupo para poner en común las decisiones tomadas sobre el diseño.
- Concreción de los puntos de comunicación entre el diseño escogido para su integración en el interfaz. Este apartado será realizado de común acuerdo entre los dos grupos del proyecto y se definirán la estructura y el formato de los puntos comunes.
- Revisión de la especificación de requisitos para la ampliación incremental. Los requisitos vendrán dados por las necesidades que plantee el otro grupo.
- Revisión y seguimiento de los riesgos detectados en el proyecto.
- Investigación y búsqueda de soluciones para los riesgos más importantes para la correcta realización del proyecto. En nuestro caso se centra en el manejo de imágenes de gran tamaño y en la limitación de tipos que puede manejar la aplicación (limitaciones de JAVA).
- Elaboración del primer prototipo de la aplicación final.

ITERACIÓN 2

Fecha de inicio: 23 – 2 - 2007
Fechas de conclusión: 20 – 3 - 2007

Objetivos:

- Elaboración del prototipo final de la aplicación. En este prototipo se incorporarán las soluciones encontradas para los puntos críticos de nuestra interfaz.
- Realización de pruebas para comprobar que los errores encontrados se han subsanado realmente y garantizar así una interfaz perfectamente operativa y libre de errores antes de la integración total.
- Integración de la interfaz final con la última versión de los algoritmos facilitados por el otro grupo.
- Presentación del prototipo final de la aplicación al tutor, una vez que se han integrado ambas partes al tutor.
- Modificación de los errores o deficiencias encontradas por el tutor, al prototipo o versión final de nuestra parte de la aplicación. Esta será la versión final de nuestro proyecto.
- Elaboración de la documentación correspondiente a esta fase.

FASE DE TRANSICIÓN.

Esta fase está compuesta por una iteración, y tiene las siguientes características:

Fecha de inicio: 21 - 3 - 2007
Fechas de conclusión: 8 - 6 - 2007

Objetivos:

- Diseño de un plan de pruebas para comprobar el correcto funcionamiento de la aplicación. Estas pruebas se efectuarán con diferentes cargas, y verificarán si la solución obtenida es correcta y si el tiempo de ejecución es aceptable.
- Búsqueda de mejoras para el comportamiento de nuestra interfaz. Aquí se contemplan las posibles mejoras que se pueden añadir, sobre todo de carácter visual.
- Búsqueda de mejoras para el comportamiento de la aplicación conjunta.
- Elaboración de la documentación correspondiente a esta fase y de la elaboración conjunta a presentar por los dos proyectos que forman la aplicación.
- Revisión de documentos realizados hasta la fecha.
- Diseñar un esquema de la memoria principal del proyecto y presentársela al tutor para dar su aprobación e indicar sus deficiencias.
- Realización de la memoria final del proyecto acorde con las recomendaciones sugeridas por el tutor y utilizando los documentos anteriormente elaborados.
- Elaboración del CD para la entrega final con la versión final de nuestro proyecto y de la presentación final con la que se exponga nuestro proyecto al tribunal.

2.4 Problemática

Como se ha mencionado anteriormente la principal problemática con la que nos encontrábamos estaba en relación al tratamiento de imágenes, que era totalmente desconocido para todos nosotros.

Por lo tanto lo primero que había que hacer era documentarse sobre cómo se pueden tratar las imágenes desde JAVA y las herramientas que proporciona para tal efecto.

Una vez encontrada la librería JAI, había que estudiarla en profundidad para poder dar funcionalidad a la interfaz gráfica.
Uno de los mayores problemas con los que nos encontramos fue con el modo en que la JAI almacenaba las imágenes. Esta era una cuestión vital, porque no solo se trataba de guardar y abrir imágenes, sino que además era necesario pasar la imagen original como parámetro al grupo encargado de los algoritmos y estos a su vez debían devolver la imagen tratada para su posterior almacenamiento y visualización a través del IHM.

Por lo tanto ambos grupos tenían que ponerse de acuerdo en la forma en que las imágenes serían almacenadas.

Se optó por guardar las imágenes en forma de vectores de colores aprovechando la estructura facilitada por Java, “ArrayList”, con lo que ahora las imágenes serían vistas como un vector en el que cada componente sería un píxel que contendría los valores RGB (Red - Green - Blue).

Para controlar el tamaño de la imagen se optó por acompañar al vector con el ancho de fila, con lo que sería sencillo tratar la imagen. Bastaría recorrer el vector teniendo en cuenta el ancho.

El tamaño de las imágenes fue uno de los mayores problemas que presentó el proyecto. Debido a las limitaciones de las plataformas de desarrollo la aplicación no se ejecutaba eficientemente cuando las imágenes que se pasaban excedían cierto tamaño. Para un tamaño en píxeles de 600x600 la aplicación funcionaba correctamente y con una velocidad de respuesta bastante aceptable; pero con un tamaño algo superior, como puede ser el estándar 600x800 la aplicación sencillamente se colapsaba. Se trataba de un problema de memoria. Esto nos obligó a buscar una solución.

Al principio se pensó en limitar el tamaño de las imágenes que la aplicación podía tratar, pero está solución requería una pérdida de utilidad, y se restringía el uso que el usuario podía darle a la aplicación. Por lo tanto decidimos buscar otra alternativa.

Tras varias sugerencias pensamos que, si el problema era debido a la plataforma de desarrollo, lo más lógico sería intentar buscar la solución ahí. Después de analizar el entorno de desarrollo (Eclipse) descubrimos una opción que permitía cargar dicho entorno con más memoria de la que asignaba por defecto.

Una vez hecho esto el tamaño de las imágenes que se trataban dejó de ser un problema, y lo mejor de todo era que el usuario no había tenido que sufrir ninguna restricción.

2.5 Análisis y planificación de riesgos

Descripción de riesgos

Riesgo	Descripción
Pérdida de datos	Es un riesgo de consecuencias catastróficas. Perder el código, en su totalidad o en parte, acarrearía gran cantidad de trabajo.
Tecnología desconocida	Debido a lo innovador del proyecto la tecnología con que se trata es del todo, o en parte, desconocida para el grupo de desarrollo.
Falta de comunicación entre los grupos	Hay dos grupos que trabajan distintas partes del proyecto en paralelo, si las distintas partes no se comunican correctamente el proyecto no avanzará como es debido.
Malentendidos en la	Debido a la complejidad del proyecto y a la ambigüedad de los conceptos pueden darse

comunicación	malentendidos entre ambos grupos, sobre todo en la comunicación escrita.
Complejidad del código	Hay algunas funcionalidades que se presentan al grupo como muy atractivas, pero que precisan de una complejidad que no las hace rentables.
Insuficiencia de recursos	Es muy posible que a lo largo de proyecto se encuentren problemas de memoria, debido a que el tratamiento de imágenes es muy exigente.
Renuncia de algún miembro	Puede darse el caso de que algún integrante del grupo, por razones personales, de salud, etc, deba abandonar el desarrollo del proyecto.
Problemas de ensamblado de las partes	Como se ha dicho antes, hay dos grupo trabajando en paralelo. Es posible que a la hora de juntarlo todo se encuentren ligeras diferencias en la aplicación de los estándares.
Bloqueos de grupos	Si un grupo desarrolla más deprisa que el otro y para seguir su trabajo precisa de resultados del otro grupo se producirá una espera.

Tratamiento de riesgos

Riesgo	Probabilidad	Tratamiento
Pérdida de datos	Baja	Cada uno de los miembros dispone de una copia del proyecto y además se guarda otra en un servidor.
Tecnología desconocida	Muy alta	Desgraciadamente es un riesgo difícil de tratar. A medida que se avanza pueden surgir nuevos problemas.
Falta de comunicación entre los grupos	Moderada	Se dispone de un grupo de mensajería que se utiliza para comunicar con todos los demás miembros.
		Se establecieron días fijos para la

Malentendidos en la comunicación	Moderada	comunicación personal y así evitar posibles malentendidos y poder ser más concretos.
Complejidad del código	Alta	Riesgo no tratable. Si la función requiere de un código excesivamente complejo, cambiar la función es más factible.
Insuficiencia de recursos	Alta	Hacer un uso óptimo de los recursos. Si aun así no es suficiente, redefinir la funcionalidad que da problemas o asumir errores.
Renuncia de algún miembro	Muy baja	Si se trata de una única renuncia se continúa con la baja; si son más de una se juntan ambos grupos en uno solo.
Problemas de ensamblado de las partes	Moderada	Se definen estándares para el paso de parámetros y para la interacción de los algoritmos con la parte gráfica.
Bloqueos de grupos	Baja	Se realiza una planificación que reparta una carga de trabajo similar en ambos grupos para evitar así los bloqueos en forma de espera.

3 Descripción de los métodos de clasificación

3.1 Plataformas de desarrollo

Como se ha indicado previamente, tras la evaluación de las ventajas e inconvenientes de Matlab frente a JAVA se optó por llevar a cabo el desarrollo en esta última.

3.2 Algoritmos empleados

A continuación proporcionamos una visión general de los algoritmos que se van a implementar en la plataforma con el fin de identificar las entradas y salidas de cada uno de ellos y su comunicación con el usuario a través del IHM.

3.2.1 Algoritmo de Lloyd

3.2.1.1 Idea general

El método propuesto a continuación es una versión modificada del AGL original y se conoce como algoritmo de aprendizaje competitivo no supervisado en la literatura de redes neuronales.

La red básica de aprendizaje competitivo consta de una única capa de neuronas, de forma que hay tantas neuronas como número de clases haya especificado el usuario y cada nodo de salida representa una categoría de patrones. Cada neurona tiene asociado un centro; los centros de las neuronas constituyen el objeto del aprendizaje.

3.2.1.2 Especificación formal

3.2.1.2.1 Inicialización de centros

Como primer paso para la clasificación, la red neuronal de aprendizaje competitivo no supervisado requiere que sus centros sean inicializados. Al basar la clasificación en los centros de la red, este paso adquiere una notable importancia, ya que el buen resultado de la clasificación está directamente relacionado con la calidad de los centros iniciales. Es decir, cuanto mejor sean los valores iniciales de los centros mejores resultados se obtendrán en la fase de clasificación.

Para inicializar dichos centros se ofrecen varias técnicas las cuales describimos de mayor a menor calidad:

- ***Fuzzy Clustering****:* Esta técnica de inicialización hace uso del método Fuzzy para obtener una agrupación de píxeles en las clases existentes. De dicha agrupación se obtiene la media aritmética para cada clase obteniendo como resultado el valor inicial del centro asociado a dicha clase..

- ***Random**:* Esta técnica se basa en generar los valores iniciales de los centros de forma aleatoria. La calidad de esta técnica esta sujeta al azar, por ello no es muy recomendable, ya que no asegura el buen funcionamiento del algoritmo en su conjunto. La ventaja de esta técnica, y principal motivo de su implementación, es que la complejidad de cálculo es muy inferior al *Fuzzy Clustering* y por tanto la velocidad de cálculo es notablemente superior.

Además de estas técnicas hemos contemplado la posibilidad de que el usuario haga uso de valores de centros ya creados en procesos anteriores.

3.2.1.2.2 Entrenamiento

Esta fase adquiere su importancia, por el hecho de tener el objetivo de mejorar los centros ya existentes y por consiguiente obtener una notable mejora en los resultados de la clasificación posterior.

La red no supervisada adapta sus centros en base a la regla de aprendizaje competitivo, que hace que las neuronas o clases **compitan por el derecho a responder** por ellas mismas por un determinado tipo de entrada. Esto se puede ver como un sistema muy sofisticado de clasificación, cuyo objetivo es dividir un conjunto de patrones de entrada en un número de clases tal que los patrones de la misma clase exhiben un cierto grado de similitud.

Para evaluar el grado de similitud de los patrones. Algunas de estas distancias pueden ser:

1. Producto interno:

$$\vec{A} \cdot \vec{B} = |\vec{A}||\vec{B}| cos\theta$$

2. Distancia Euclídea con Centros:

$$D(\mathbf{x}(k), \boldsymbol{c}_j(k)) = \left\| \mathbf{x}(k) - \boldsymbol{c}_j(k) \right\|^2$$

En este caso hemos optado por el más común, que es la distancia Euclídea entre el patrón de entrada y los centros existentes en la red. Los patrones de entrada corresponden con cada uno de los píxeles que forman la imagen, por consiguiente los patrones de entrada y los centros van a ser vectores de tres componentes (R,G,B).

El algoritmo de aprendizaje se detalla a continuación, siendo repetido tantas veces como el usuario indique.

1) Inicio: dados los puntos de datos $\mathbf{x}(k)$, $k = 1,2,\ldots$, y centros de salida iniciales $\boldsymbol{c}_j(0)$, $j = 1,\ldots,m$, siendo m el número de clases.

2) Determinar el centro $\boldsymbol{c}_j(k)$ más próximo al punto $\mathbf{x}(k)$

$$j = \arg\min_j \left\| \mathbf{x}(k) - \boldsymbol{c}_j(k) \right\|^2$$

3) Actualizar el centro de salida utilizando las ecuaciones,

$$\boldsymbol{c}_j(k_j + 1) = \boldsymbol{c}_j(k_j) - \gamma(k_j) g\text{rad}\left(\left\| \mathbf{x}(k) - \boldsymbol{c}_j(k_j) \right\|^2 \right)$$

$$k_j = k_j + 1$$

Obsérvese que cada centro podría tener su propia razón de aprendizaje, lo que se indica con k_j en $\gamma(k_j)$, con $j = 1,\ldots,m$. El gradiente se calcularía como,

$$\frac{\partial}{\partial \boldsymbol{c}_j} \left\| \mathbf{x} - \boldsymbol{c}_j \right\|^2 = 2(\mathbf{x} - \boldsymbol{c}_j)$$

La actualización del centro, sólo se lleva a cabo en el caso de que la diferencia entre el nuevo centro y el antiguo, supera la tolerancia establecida por el usuario (por defecto su valor se fija en 10^{-10}).

Pero nosotros hemos optado por una razón de aprendizaje común, la cual es introducida por el usuario, y que por defecto su valor es de 0.1. Con este gradiente, los centros de salida se actualizan cuando $\boldsymbol{c}_j$.

$$\boldsymbol{c}_j(k_j + 1) = \boldsymbol{c}_j(k_j) + \gamma(k_j)\left[\mathbf{x}(k) - \boldsymbol{c}_j(k_j) \right]$$

$$k_j = k_j + 1$$

3.2.1.2.3 Clasificación

Los procedimientos de clasificación no supervisados se basan a menudo en algunas técnicas de clasificación, que forman grupos de patrones parecidos. Para realizar la clasificación es necesario definir, de nuevo, una distancia o medida de similaridad. Nosotros seguimos basándonos en la distancia Euclídea.

1) Inicio: dados los puntos de datos $\mathbf{x}(k)$, $k = 1,2,\ldots$, y centros de salida ya actualizados $\boldsymbol{c}_j$, $j = 1,\ldots,m$, siendo m el número de clases.

2) Determinar el centro $\boldsymbol{c}_j$ más próximo al punto $\mathbf{x}(k)$

$$j = \arg\min_j \left\| \mathbf{x}(k) - \boldsymbol{c}_j(k) \right\|^2$$

Una vez conocido el centro más próximo la entrada se asigna a la clase correspondiente *j*.

3.2.1.3 Análisis de los parámetros

Como consecuencia del análisis del método tanto en aprendizaje como en clasificación a continuación se proporcionan los parámetros utilizados en el algoritmo de Lloyd.

1. Número de clústeres o clases ´*c*´.

Parámetro que establece el número de particiones o clases en que se desea clasificar el conjunto de datos iniciales o muestras. El valor de este parámetro depende del número de clases en las que el usuario quiera clasificar la imagen o imágenes.

2. Numero de iteraciones 'NUMITER'

Este parámetro se corresponde con el número de veces que el usuario quiera repetir la fase de aprendizaje. Cuanto mayor sea este número los centros obtendrán mejores valores iniciales, y por tanto una mejor clasificación. Por el contrario, al aumentar el número de iteraciones se aumenta la complejidad del algoritmo, y por tanto disminuirá la velocidad de cálculo de este.

3. Error permitido o tolerancia ´ε´

Parámetro utilizado en el entrenamiento, este se encarga de definir el grado de precisión de los centros obtenidos por el entrenamiento. Por lo tanto, a valores muy grandes de este parámetro la precisión será muy pequeña. Este parámetro establece un umbral al número de iteraciones, ya que llegado al punto en el cual la diferencia entre el nuevo centro y el actual no supere el valor de tolerancia, la actualización de dicho centro no se llevará a cabo. (Su valor por defecto es 10^{-10}).

4. Razón de aprendizaje $'\gamma'$.

La "cantidad'" de la corrección va a estar determinada por este parámetro. Cualquier función de corrección utiliza la llamada ***función de ganancia*** o ***razón de aprendizaje***, definida como γ: $N \rightarrow R$. La única restricción sobre esta función es que debe proporcionar una secuencia monótona que cumpla $0 < \gamma(t) < 1$. Valores altos de $\gamma(t)$ hacen que el prototipo se acerque mucho al patrón y valores bajos hacen que el acercamiento sea más moderado. En nuestro caso hemos optado porque dicha función sea constante. (Su valor por defecto es 0.1).

3.2.2 Algoritmo C-Medias Fuzzy

3.2.2.1 Idea general

El algoritmo C-medias fuzzy es un algoritmo iterativo de clasificación de imágenes basado en la técnica de agrupamiento Fuzzy Clustering. Este algoritmo está compuesto por dos fases: entrenamiento no supervisado y clasificación.

En el entrenamiento inicialmente se señalan arbitrariamente los centros de las clases, de acuerdo con un número de clases indicadas por el usuario. Los píxeles se asignan al centro más cercano, y se vuelven a calcular los nuevos centros. Este proceso se repite hasta alcanzar un número máximo de iteraciones, o que la modificación entre los valores de pertenencia asignados entre dos iteraciones diferentes sea menor que una determinada tolerancia especificada en el algoritmo. Este método utiliza la distancia espectral mínima para asignar cada píxel a un centro candidato. Una vez concluido el proceso anterior, el valor de los centros resultantes va a ser el utilizado para realizar la clasificación de las imágenes deseadas.

El proceso de clasificación consiste en la obtención del valor de pertenencia de cada uno de los píxeles que componen la imagen con respecto a cada una de las clases definidas, para ello se calculará la distancia de este píxel al centro correspondiente a cada clase. El píxel será asignado a la clase que mayor valor de pertenencia ha obtenido anteriormente.

3.2.2.2 Especificación formal

3.2.2.2.1 Fuzzy Clustering

El objetivo de la técnica de agrupamiento conocida como *Fuzzy Clustering* consiste en dividir *n* objetos $x \in X$ caracterizados por *p* propiedades en *c* "clústeres" o grupos. Supongamos el conjunto de datos $X = \{x_1, x_2, ..., x_n\} \in \Re^p$ un subconjunto del espacio real *p*-dimensional $\Re^p$. Cada $x_k = \{x_{k_1}, x_{k_2}, ..., x_{k_p}\} \in \Re^p$ se denomina vector de características, x_{k_j} es la *j*-ésima característica de la observación x_k. En nuestro caso concreto *X* va a ser el conjunto de todos los píxeles de cada imagen y los x_i van a pertenecer a $\Re^3$, ya que cada píxel va estar caracterizado por las propiedades R,G,B por lo cual ´ *p*´ es igual a tres en nuestra aplicación . El número de clústeres ´c´ será especificado por el usuario, siendo variable dependiendo del tipo de imagen a clasificar.

Puesto que los elementos de un clúster deben ser tan similares entre sí como sea posible y a la vez deben ser tan diferentes a los elementos de otros clústeres como también sea posible, el proceso de clustering se controla por el uso de medidas de similitud basadas en distancias. Así la similitud o la diferencia entre dos puntos x_k y x_l puede interpretarse como la distancia entre esos puntos.

Una distancia entre dos objetos x_k y x_l es una función que toma valores reales $d : XxX \rightarrow R^+$ cumpliendo:

$$d(x_k, x_l) = d_{kl} \geq 0 ; \;\; d_{kl} = 0 \Leftrightarrow x_k = x_l ; \;\; d_{kl} = d_{lk} \text{ y } \;\; d_{kl} \leq d_{kj} + d_{jl}$$

Si se desea realizar una partición del conjunto *X* en *c* clústeres tendremos $S_i \{i = 1,...,c\}$ subconjuntos. A partir de esta consideración se define lo que se conoce como grado de pertenencia μ_{ik} de cada objeto x_k al subconjunto S_i. Cada uno de estos sí será un conjunto fuzzy por lo cual un objeto puede pertenecer a diferentes subconjuntos. Por lo tanto se puede decir que x_k pertenece a un conjunto S_i con grado de pertenencia μ_{ik} y a S_j con grado de pertenencia μ_{ij}. Todos los valores de referencia tomados para cada x_k están dentro del intervalo continuo [0,1].

Dado $X = \{x_1, x_2,..., x_n\}$ y el conjunto V_{cn} de todas las matrices reales de dimensión *c x n*, con $2 \leq c < n$. Se puede obtener una matriz representando la partición de la siguiente manera $U = \{\mu_{ik}\} \in V_{cn}$. Esta matriz cumple las siguientes condiciones:

1) $\mu_{ik} \in \{0,1\}$ *crisp* o $\mu_{ik} \in [0,1]$ *fuzzy* $\quad 1 \leq i \leq c; \;\; 1 \leq k \leq n$

2) $\sum_{i=1}^{c} \mu_{ik} = 1 \qquad 1 \leq k \leq n$

3) $0 < \sum_{k=1}^{n} \mu_{ik} < n \qquad 1 \leq i \leq c$

La localización de un clúster S_i se representa por su centro $v_i = \{v_{i_1}, v_{i_2},..., v_{i_p}\} \in \Re^p$ con $i = 1,\ldots,c$, alrededor del cual se concentran los objetos. La definición básica de llevar a cabo el problema de la partición fuzzy para $m > 1$ consiste en minimizar una función objetivo según la siguiente expresión:

$$\min z_m(U; v) = \sum_{k=1}^{n} \sum_{i=1}^{c} \mu_{ik}^m \left\| x_k - v_i \right\|_G^2$$

donde *G* es una matriz de dimensión *p x p* que es simétrica y definida positiva. Así se puede definir una norma general este tipo:

$$\left\| x_k - v_i \right\|_G^2 = (x_k - v_i)^t G (x_k - v_i)$$

Diferenciando la función objetivo para v_i (suponiendo constante *U*) y μ_{ik} (suponiendo constante *v*) y aplicando la condición de que $\sum_{i=1} \mu_{ik} = 1$, se obtiene:

$$v_i = \frac{1}{\sum_{k=1}^{n} (\mu_{ik})^m} \sum_{k=1}^{n} (\mu_{ik})^m x_k \qquad i = 1,...,c \quad (1)$$

$$\mu_{ik} = \frac{\left(\frac{1}{\|x_k - v_i\|_G^2} \right)^{2/m-1}}{\sum_{j=1}^{c} \left(\frac{1}{\|x_k - v_i\|_G^2} \right)^{2/m-1}} \qquad i = 1,...,c; k = 1,...,n \qquad (2)$$

donde el exponente m es un parámetro que se conoce como peso exponencial.

3.2.2.2.2 Entrenamiento

De nuevo, el objetivo del aprendizaje son los centros de las clases.

El algoritmo de utilizado para el entrenamiento es el C-Medias Fuzzy. Este algoritmo de agrupamiento no supervisado está basado en el Fuzzy Clustering explicado anteriormente, y a partir de unas muestras iniciales, que en nuestro caso serán los píxeles para el entrenamiento, obtiene los centros correspondientes de los clústeres especificados.

Estos son los pasos establecidos en el algoritmo para obtener los mencionados centros:

1) Elegir c ($2 \le c \le n$), m ($1 < m < \infty$) y la matriz G de dimensión p x p siendo simétrica y definida positiva. Inicializar $U^{(0)}$ y poner t = 0. En nuestro algoritmo G va a ser igual a la matriz identidad debido a que la norma matricial elegida es la norma euclídea, en caso de cambiar la norma el valor de G sería distinto.

2) Inicialización pseudos-aleatoria de cada uno de los centros de las clases v_i con $0 < i \le c$.

3) Calcular los c centros fuzzy de los clústeres a partir de (1) $\{v_i^{(t)}\}$ utilizando $U^{(t)}$.

4) Calcular los nuevos grados de pertenencia de la matriz $U^{(t+1)}$ utilizando $\{v_i^{(t)}\}$ a partir de la condición (2) si $x_k \neq v_i^{(t)}$. De lo contrario

$$\mu_{jk} = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases}$$

5) Elegir una norma matricial, que en nuestro algoritmo ha sido la norma euclídea, y calcular $\Delta = \|U^{(t+1)} - U^t\|_G$. Si $\Delta > \varepsilon$ poner $t = t + 1$ y regresar al paso 2), de lo contrario detener el proceso.

Para limitar el tiempo de ejecución del algoritmo se pondrá un límite de iteraciones al proceso de forma que aunque se cumpla $\Delta > \varepsilon$ se detendrá el proceso.

Los valores de los centros obtenidos tras la finalización del proceso será el posteriormente utilizado para realizar la clasificación.

3.2.2.2.3 Clasificación

Los pasos realizados para cada una de las muestras a clasificar, que en nuestro caso serán píxeles son las siguientes:

1. Cálculo de la distancia de cada muestra a cada centro de los centros de los clústeres. Para calcular esa distancia utilizaremos la distancia euclídea:

$$\left\| x_k - v_1 \right\|_G^2$$

2. Cálculo de la función de pertenencia de la muestra a cada uno de los clústers o clases, para ello sustituiremos el valor dado en el paso 1, en la ecuación (1), dando el valor de la función de pertenencia para cada una de las clases.

3. Clasificación de la muestra con aquel clúster que obtenga un mayor valor de la función de pertenencia.

Análisis de parámetros

Estos son los parámetros utilizados en el algoritmo C-Medias Fuzzy. Estos datos los define cada usuario que ejecute dicho algoritmo con unos valores apropiados para su aplicación.

a. Número de clústeres o clases ´*c*´.

Parámetro que establece el número de particiones o clases en que se desea clasificar el conjunto de datos iniciales o muestras. El valor de este parámetro depende del número de clases en que el usuario quiera clasificar la imagen o imágenes.

b. Peso exponencial ´*m*´

Este parámetro utilizado en uno reduce la influencia del ruido cuando se obtienen los centros de los clústeres, reduce la influencia de los valores pequeños de μ_{ik} (puntos lejos de v_i) frente a valores altos de μ_{ik} (puntos cerca de v_i). Cuanto mayor sea $m > 1$ mayor es dicha influencia. El valor típico para este parámetro es 2.

c. Error permitido o tolerancia ´ε´

Parámetro utilizado en el entrenamiento, este parámetro se encarga de definir el grado de precisión de los centros obtenidos por el entrenamiento. Por lo tanto a valores muy grandes de este parámetro la precisión será pequeña, en caso de establecer un número muy bajo el tiempo de ejecución del entrenamiento puede ser muy elevado al tener que ejecutar un gran número de veces el bucle principal del

algoritmo del entrenamiento, para controlar este número de iteraciones utilizaremos el parámetro 'NumMax' que indica el número máximo de iteraciones posibles. El valor típico para el error permitido suele ser en torno a 0.02.

d. Número máximo de iteraciones 'NumMax'.

Parámetro utilizado en el entrenamiento para establecer un número máximo de iteraciones del bucle principal utilizado en este algoritmo. Con esto conseguimos establecer un tiempo máximo de ejecución, evitando ejecuciones muy costosas debido a una tolerancia extremadamente pequeña o debido a la dificultad de convergencia del algoritmo provocada por una mala elección de los datos de muestra. El valor por defecto establecido para este parámetro es 10.

Algoritmo Bayesiano

3.2.3.1 Idea general

La teoría de la decisión de Bayes es un método estadístico clásico en clasificación de patrones. Se basa en el supuesto de que el problema de la decisión se enfoca en términos probabilísticos y que todas las probabilidades relevantes resultan conocidas.

Se trata de un método supervisado, lo que implica que a partir de un número de clases dado, el algoritmo determina los centros de las mismas y la distribución de las muestras en las clases. La clasificación se la proporciona el algoritmo Fuzzy Clustering que establece una clasificación inicial.

La clasificación de cada píxel de una imagen se determina según la probabilidad de pertenecer a cada una de ellas. Esa probabilidad vendrá dada en nuestro caso por la distancia de Mahalanobis que a su vez se calcula con los datos obtenidos en el entrenamiento.

3.2.3.2 Especificación formal

El algoritmo de clasificación Bayesiana en el ámbito de la clasificación de imágenes nos será de utilidad para analizar una imagen píxel a píxel y determina a qué clase pertenece cada uno según una probabilidad calculada.

Supongamos que no tenemos conocimiento a cerca de los valores R, G, y B del píxel que estamos tratando de clasificar. Utilizando la terminología de la teoría de la decisión, podemos decir que la probabilidad a priori de que un píxel pertenezca a cualquiera de las clases es la misma y cuya suma es la unidad.

Sea $\boldsymbol{y}$ el estado que designa la pertenencia de un píxel a una de las clases, c el número de clases en que hay que dividir la imagen y considerando a $\boldsymbol{y}$ una variable

aleatoria, ya que la pertenencia a uno de los dos estados es impredecible, podemos decir de forma más precisa que suponemos que existe alguna *probabilidad a priori* $P(y=c_i)$ de que el píxel pertenezca a la clase c_i y alguna *probabilidad a priori* de que pertenezca a la clase c_j $P(y=c_j)$ y así con todas las clases que queramos hasta ***c***.

A la hora de decidir sobre la pertenencia de un píxel a una clase concreta, la única información son esas probabilidades a priori y parece razonable utilizar la siguiente *regla de decisión*: La muestra **x** pertenece a aquella clase que genere la mayor probabilidad.

Dado que en nuestro caso conocemos los valores para R, G y B del píxel a clasificar. Estos tres valores se definen como $\mathbf{x} = \{x_R, x_G, x_B\}$**.** Así, diferentes píxeles darán diferentes valores de ***x***, resulta natural por tanto expresar esta variabilidad en términos probabilísticos. En este sentido, consideremos a ***x*** una variable aleatoria continua cuya distribución depende de la clase.

Sean $p(\mathbf{x} / y = c_j)$ las funciones de *densidad de probabilidad condicionales* para ***x*** dado que el píxel pertenezca a la clase c_i. Supongamos que conocemos tanto las probabilidades a priori como estas últimas funciones de densidad de probabilidad. Suponer además que medimos las componentes R, G y B de un píxel y descubrimos que toma el valor *x*, la pregunta será ¿cómo influye esta medida sobre nuestra actitud con relación a la clase de que se trata? La respuesta nos la proporciona la *regla de Bayes*, considerando que tanto las probabilidades a priori $P(y = c_j)$ como las densidades condicionales para cada clase $p(\mathbf{x} / y = c_j)$ son conocidas o se pueden estimar, es posible determinar para una observación dada **x** la probabilidad de que esa observación pertenezca a una determinada clase. Estas probabilidades, llamadas *probabilidades a posteriori* pueden usarse para construir una regla discriminante:

$$p(y = c_j / \mathbf{x}) = \frac{p(\mathbf{x} / y = c_j) P(y = c_j)}{p(\mathbf{x})}$$

donde

$$p(\mathbf{x}) = \sum_{j=1}^{c} p(\mathbf{x} / y = c_j) P(y = c_j)$$

La regla de Bayes muestra cómo la observación del valor ***x*** cambia las probabilidades a priori a las *probabilidades a posteriori* $p(y = c_j / \mathbf{x})$.

Una vez que se determinan esas probabilidades a posteriori, la siguiente regla de decisión se utiliza para clasificar *x*.

$$\mathbf{x} \in c_i \;\; sii \;\; P(y = c_i/\mathbf{x}) > P(y = c_j/\mathbf{x}) \;\; \forall\, i \neq j, \;\; i, j = 1,2,..c$$

Ahora bien, si nos fijamos en el segundo término de la expresión que obtiene las *probabilidades a posteriori* del teorema de Bayes y eliminado el término no discriminante $p(\mathbf{x})$ (no aporta nada en la decisión), se tiene una forma alternativa de clasificar el vector de atributos **x**:

$$\mathbf{x} \in c_i \;\; sii \;\; P(\mathbf{x}/\, y = c_i)P(y = c_i) > P(\mathbf{x}/\, y = c_j)P(y = c_j) \;\; \forall i \neq j,\; i, j = 1,2,..,c$$

Generalmente las distribuciones de densidad de probabilidad se eligen Normales o Gaussianas. Un caso especial surge cuando las probabilidades a priori son iguales para todas las clases, ya que en esta situación la distancia de Mahalanobis se puede utilizar como función discriminante mediante la siguiente regla de decisión a partir de la regla anterior y teniendo en cuenta el signo negativo en el término exponencial de la función de densidad de probabilidad Normal, así

$$\mathbf{x} \in c_i \;\; sii \;\; d_M^2(\mathbf{x}, \mathbf{m}_i) < d_M^2(\mathbf{x}, \mathbf{m}_j) \;\; \forall i \neq j,\; i, j = 1,2,..,c$$

donde $\mathbf{m}_i$, $\mathbf{m}_j$ son los vectores media de las clases c_i y c_j respectivamente.

Sin pérdida de generalidad, la distancia de un vector $\mathbf{x}_k$ a la clase c_i resulta ser:

$$d_M^2(\mathbf{x}_k, \mathbf{m}_i) = (\mathbf{x}_k - \mathbf{m}_i)^{\mathrm{t}} C_i^{-1} (\mathbf{x}_k - \mathbf{m}_i)$$

En el supuesto de que las matrices de covarianza sean la identidad, la distancia de Mahalanobis al cuadrado resulta ser la distancia Euclídea al cuadrado, en cuyo caso tendríamos,

$$d_E^2(\mathbf{x}, \mathbf{m}_i) = (\mathbf{x} - \mathbf{m}_i)^t (\mathbf{x} - \mathbf{m}_i) = \mathbf{x}^t\mathbf{x} - 2\mathbf{x}^t\mathbf{m}_i + \mathbf{m}_i^t\mathbf{m}_i$$

En la expresión anterior el término $\mathbf{x}^t\mathbf{x}$ no discrimina, ya que se repite en todas las clases, de forma que puede despreciarse. Ahora, si se cambia de signo y se divide por 2 en la ecuación anterior se obtiene la siguiente función discriminante, donde se deduce que la distancia Euclídea al cuadrado mínima hace la expresión siguiente máxima:

$$fd_i(\mathbf{x}) = \mathbf{x}^{\mathrm{t}}\mathbf{m}_{\mathrm{i}} - \frac{1}{2}\mathbf{m}_{\mathrm{i}}^{\mathrm{t}}\mathbf{m}_{\mathrm{i}}$$

Entrenamiento

El entrenamiento, como paso previo a la clasificación Bayesiana consistirá en obtener las medias y matrices de covarianza de las muestras para cada clase

En el caso de que se quisiera clasificar varias imágenes también es posible obtener más muestras una vez hayamos finalizado la clasificación de una imagen, añadiendo a la base de muestras la salida obtenida y refinando por tanto la calidad de las muestras para utilizarlas en posteriores clasificaciones.

Una vez obtenidas las muestras se procede a calcular para cada clase la media y la covarianza como se muestra a continuación:

$$\mathbf{m} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i \qquad C = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{x}_i - \mathbf{m})(\mathbf{x}_i - \mathbf{m})^{\mathrm{t}}$$

El valor de las medias y covarianzas obtenidas serán posteriormente utilizadas para realizar la clasificación.

3.2.3.3 Análisis de Parámetros

Los parámetros necesarios para el algoritmo Bayesiano son los que se muestran a continuación:

1. Número de clases *´c´*.

Se utiliza en el entrenamiento y en la clasificación. Este parámetro indica el número de clases en que se desea clasificar los píxeles de una imagen.

2. Muestras clasificadas

Este parámetro corresponde con las muestras del entrenamiento, sobre las que se calculan medias y matrices de covarianzas.

3. Medias '$\mathbf{m_i}$'y matrices de Covarianza **'$\mathbf{C_i}$'**

Si se han guardado resultados de clasificaciones previas puede ser de gran utilidad utilizar las medias y matrices de covarianzas obtenidas.

4. Parámetros del algoritmo C-Medias Fuzzy.

Puesto que para obtener las muestras iniciales clasificadas para nuestro entrenamiento utilizamos el algoritmo C-Medias Fuzzy, serán necesarios unos valores apropiados para su aplicación.

4 La interfaz gráfica a partir de los algoritmos

Descripción:

La interfaz gráfica es, en todas las aplicaciones, un reflejo de la calidad externa del producto, puesto que es la única que percibe el usuario. Que una interfaz sea más o menos intuitiva, o que tenga una mejor o peor presentación, puede influir en el éxito del producto en el mercado.

Una buena interfaz tiene que dar pleno soporte a toda la funcionalidad de la aplicación; pero no basta con eso, además tiene que ser intuitiva y atractiva.

Una interfaz demasiado sería puede dar una sensación poco intuitiva, y por añadidura, de una funcionalidad escueta. Por el contrario una interfaz muy sobrecargada en colores y formas daría una sensación de poca profesionalidad, y el usuario final tendería a desconfiar del producto.

Al principio del desarrollo del proyecto la interfaz era algo terriblemente sencillo. Consistía tan solo en una pequeña ventana con un botón, que servía como investigación y sobre la cual se pretendía dar soporte a una funcionalidad incremental.

A medida que fue avanzando el proyecto la interfaz fue evolucionando, hasta llegar al estado actual tal y como se presenta en este trabajo. La cual, cuenta en su totalidad con cerca de 3200 líneas de las 5000 aproximadamente que tiene el proyecto completo.

4.1 Plataformas de desarrollo

Elegida la plataforma JAVA por las razones expresadas previamente, el siguiente paso consistía en diseñar un modelo. Con tal propósito nos fijamos en la plataforma de desarrollo Jbuilder que proporciona abundantes componentes, a la vez que estudiamos alguna plataforma comercial como puede ser ERDAS.

Esto nos sirvió de mucha ayuda, pero no estaba falta de complicaciones. Muchas de las librerías que utiliza Jborland son de producción propia, y por consiguiente no son migrables a otras plataformas. Esta fue una de las mayores complicaciones con la que nos encontramos a la hora de diseñar la interfaz. Ganamos mucho tiempo diseñando con Jbuilder, pero debíamos invertir otra cantidad de tiempo en buscar soluciones alternativas para las implementaciones propias de Jbuilder.

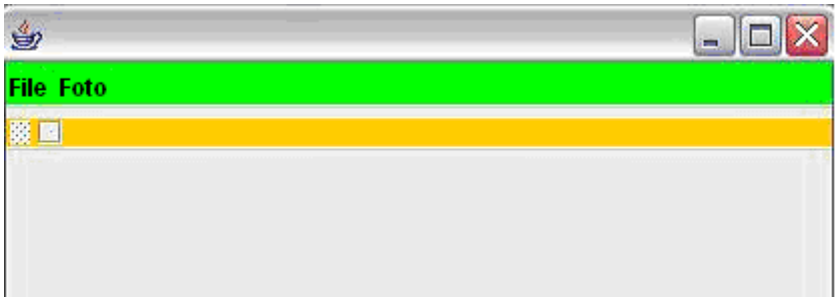
Para todo esto fue fundamental la ayuda on-line que proporciona Sun para su lenguaje, en especial para las librerias JAI.

4.2 Evolución por mejoras y necesidades

Como ya hemos dicho con anterioridad, la interfaz sufrió muchos y muy marcados cambios desde esa primera versión que servía de experimento hasta la versión resultante final.

En este apartado vamos a mostrar como fue esa evolución explicando cuales fueron los cambios y en particular la motivación de los mismos.

Empezaremos por esa primera versión experimental:



Como se puede comprobar, tanto el diseño como la funcionalidad que podía soportar dejaban mucho que desear; pero aun así sirvió para empezar a comprender el mecanismo que utiliza Java para tratar las imágenes, así que podemos decir que fue de gran utilidad real.

A pesar del afecto que nos suscitaba esta primera ventana por lo trascendente, era obvio que no servía, ni siquiera como primera aproximación.

Así que decidimos buscar inspiración en las versiones comerciales que habíamos visto. Estas consistían básicamente en una barra de herramientas con tantos botones como funcionalidades permitía la aplicación. Después cada una de las imágenes que iba a ser tratadas se abrían en ventanas independientes. Así que lo primero que había que hacer era abrir una ventana que contuviese la imagen que se iba a tratar.

Esta era la barra de herramientas que daría soporte a la apertura de una imagen, y que serviría también para las primeras pruebas del primer algoritmo que se terminó, el algoritmo de Lloyd.

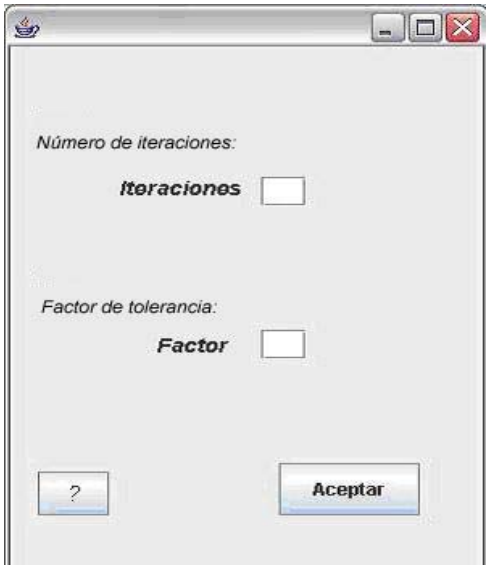
Se puede observar que el diseño seguía siendo pobre, pero ya apuntaba cuál sería el estilo que tendría finalmente.

Como es lógico, en un principio no se sabía muy bien como estarían constituidos los algoritmos al final de la aplicación, así que se pactó con el otro grupo que el algoritmo de Lloyd recibiría como parámetros una matriz que

contendría la imagen y dos enteros que tendría que introducir el usuario para indicar el número de iteraciones que se desean realizar y el factor de tolerancia.

Volvíamos a encontrarnos con la necesidad de implementar rápidamente una ventana que diese soporte a estas especificaciones, a fin de poder realizar las pruebas con todo ensamblado lo antes posible.

La ventana en cuestión tenía el siguiente aspecto:



La ventana era aun demasiado tosca, pero como ocurrió con la primera barra, nos fue muy útil para detectar errores y proponer mejoras.

Una de estas mejoras, consistía en el hecho de poder seleccionar correctamente una ventana cuando había varias abiertas. Parecía una trivialidad, pero hasta que no pusimos el algoritmo a trabajar no nos dimos cuenta. Cuando este error fue subsanado pasamos al siguiente paso.

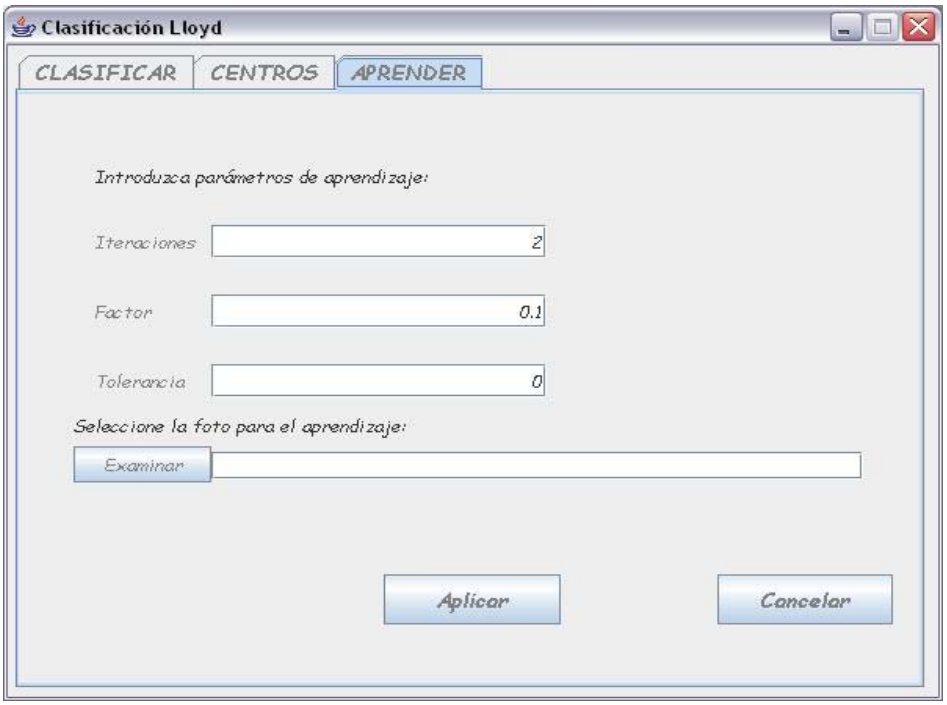
Los otros dos algoritmos estaban aún sin terminar, por lo que se decidió ir dando forma al de Lloyd, que ya estaba totalmente operativo.

Lo primero que se hizo fue acordar con el otro grupo cómo conocería el usuario los parámetros que necesitaba el algoritmo. Después acordamos todas las funcionalidades a las que había que dar soporte.

Para este algoritmo en cuestión eran tres: primero debía poder clasificar las texturas, después se le tenía que dar opción al usuario de elegir los centros que debía usar el algoritmo, y por último había que dar soporte al aprendizaje.

En un principio se pensó en hacer menús desplegables, pero al final se optó por un menú de carpetas, puesto que nos parecía más interesante a todos los niveles, tanto de presentación como de codificación.

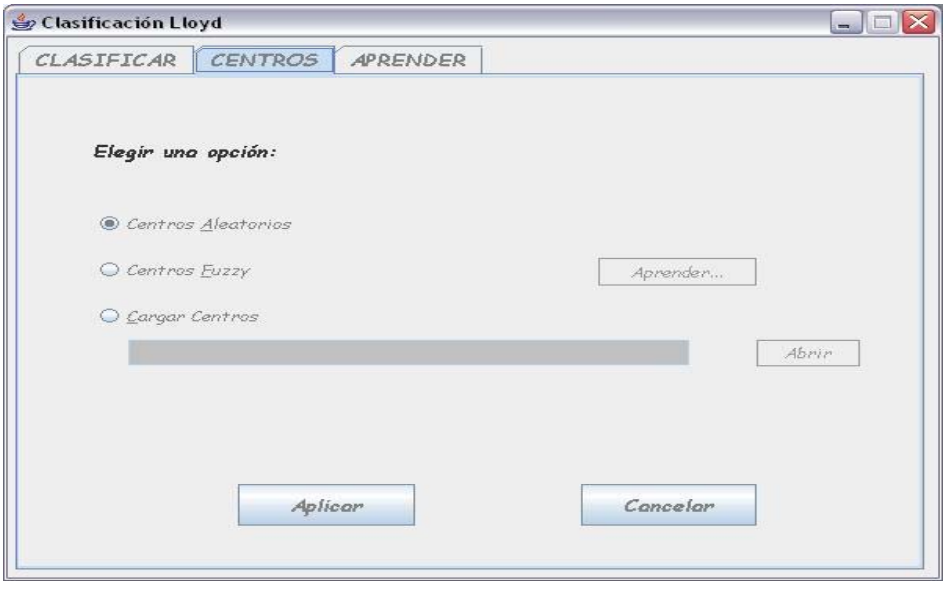
Las ventanas que daban soporte a estas nuevas funcionalidades tenían la siguiente forma:



Esta ventana proporciona la opción de aprendizaje del algoritmo. El usuario debe elegir la imagen o imágenes con las que quiere que el algoritmo aprenda, y además le debe decir el numero de iteraciones que el algoritmo debe realizar como máximo, así como el factor y la tolerancia.

Como es más que probable que el usuario final desconozca el funcionamiento implícito del algoritmo, al menos en un principio, se le da una breve pero concisa ayuda sobre los campos que debe rellenar, aunque estos tienen unos valores asignados por defecto.

La segunda pestaña nos da la opción de entrar en el menú de centros:



Los centros son valores que el algoritmo utiliza para la clasificación. Cuanto más precisos son estos centros mejor hace la diferenciación entre las distintas texturas presentes en la imagen.

Al usuario se le proporcionan tres opciones:

- Actuar con los centros elegidos de forma aleatoria, opción que viene predeterminada para obtener resultados inmediatos.

- Se da también la opción de que los centros sean calculados después de un aprendizaje aplicando otro de los algoritmos, el de Fuzzy. Esta opción es la más precisa, pero se necesita más tiempo de ejecución.

- Por último se proporciona la opción de cargar unos centros previamente calculados.

Las opciones que precisan de cálculos previos estás desactivadas hasta que el usuario realice las operaciones pertinentes.

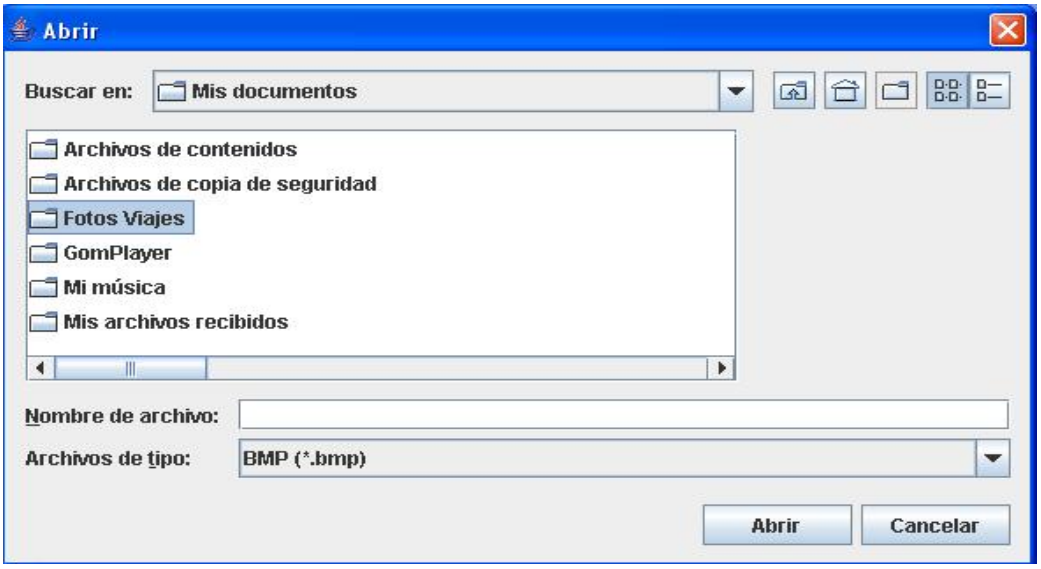
Prácticamente la totalidad de las funciones que requería el método de Lloyd tenían ya un soporte gráfico. Todos los parámetros, las ayudas y opciones estaban ya implementadas, pero faltaba un pequeño detalle: guardar los resultados obtenidos por el algoritmo.

Al principio nos encontramos con bastantes problemas, sobretodo de tipos, debido a que Java tan solo soporta un número limitado de tipos de imágenes, esto es, png, jpg y tif, por lo que hubo que diseñar un filtro que nos permitiese guardar un formato de píxeles como una imagen de cualquier tipo.

La tarea, aunque sencilla en apariencia, no lo fue en absoluto en la práctica y al inicio nos encontramos con que se guardaba un icono, que por desgracia no contenía ninguna información.

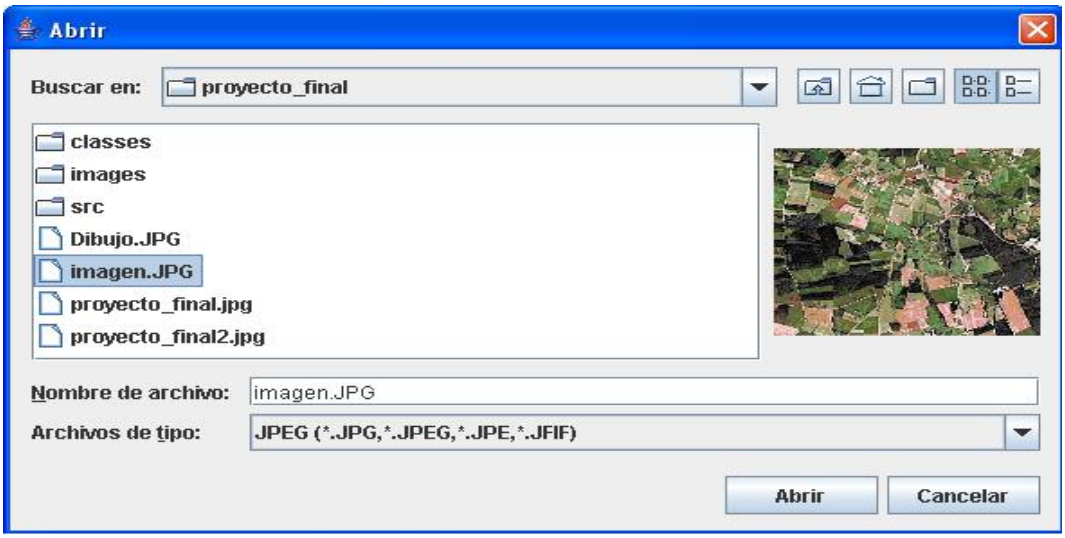
Tras diversas correcciones y depuración se llegó a conseguir la función deseada.

Al igual que ocurría con el diálogo de salvar, nos encontramos con ciertos problemas a la hora de disponer de una imagen. La funcionalidad de abrir una imagen en una ventana fue de las primeras que se obtuvieron, como es lógico. Esto, junto con el rápido avance que era necesario para cubrir las necesidades que nos transmitía el otro grupo, hicieron que la función de abrir una imagen quedase un poco abandonada. Por este motivo, una vez que todas las funcionalidades estuvieron implementadas, decidimos que había que mejorar ese diálogo de apertura, que era demasiado simple, como se puede apreciar en la imagen:



Nos pareció que una mejora muy útil sería darle al usuario la opción de poder ver una pequeña vista previa de las imágenes que estaban disponibles para ser tratadas. Al igual que pasaba con el diálogo de guardar, las cosas parecían más fáciles de lo que luego fueron a posteriori. Hubo que consultar varias páginas dedicadas a las librerías gráficas de Java, pero finalmente se obtuvo un resultado realmente satisfactorio, primeramente tan sólo para los formatos de imágenes soportados por Java, y más tarde para todos los demás.

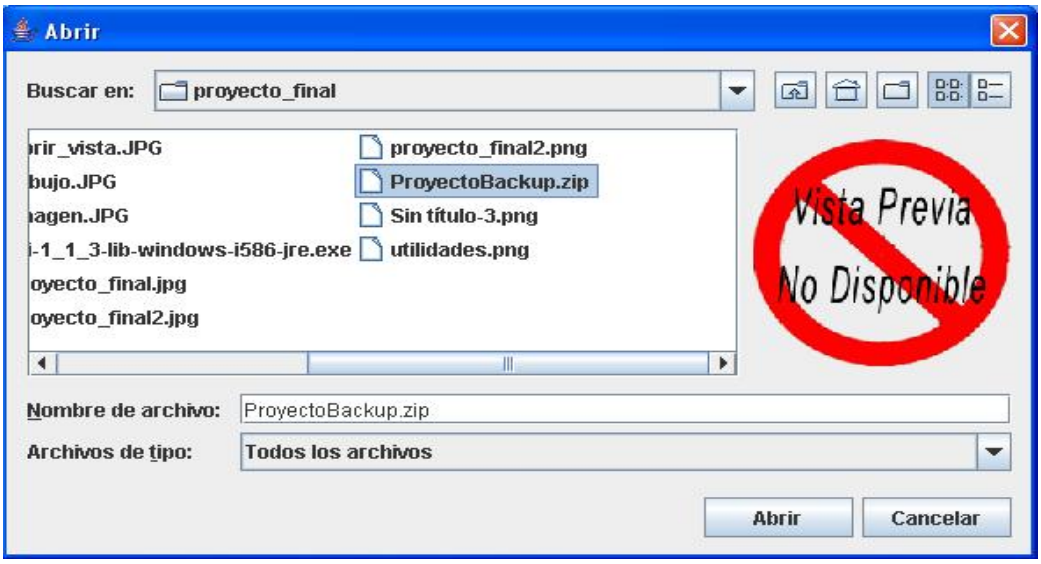
El resultado se puede ver en esta captura, donde se aprecia en el lateral derecho cómo aparece una vista previa de la imagen seleccionada:



Ahora quedaba tan sólo un pequeño detalle: tratar las imágenes que por el momento no estaban disponibles, hasta que se implementase el filtro, así como los archivos que no son imágenes.

Para ello pensamos que sería buena idea mostrar en la misma vista previa un pequeño aviso que alertase de que no se disponía de una vista previa para esa imagen, ya fuese porque no era de un formato soportado, o porque no se tratase de una imagen propiamente dicha.

El resultado, terriblemente intuitivo, se muestra en la siguiente captura:



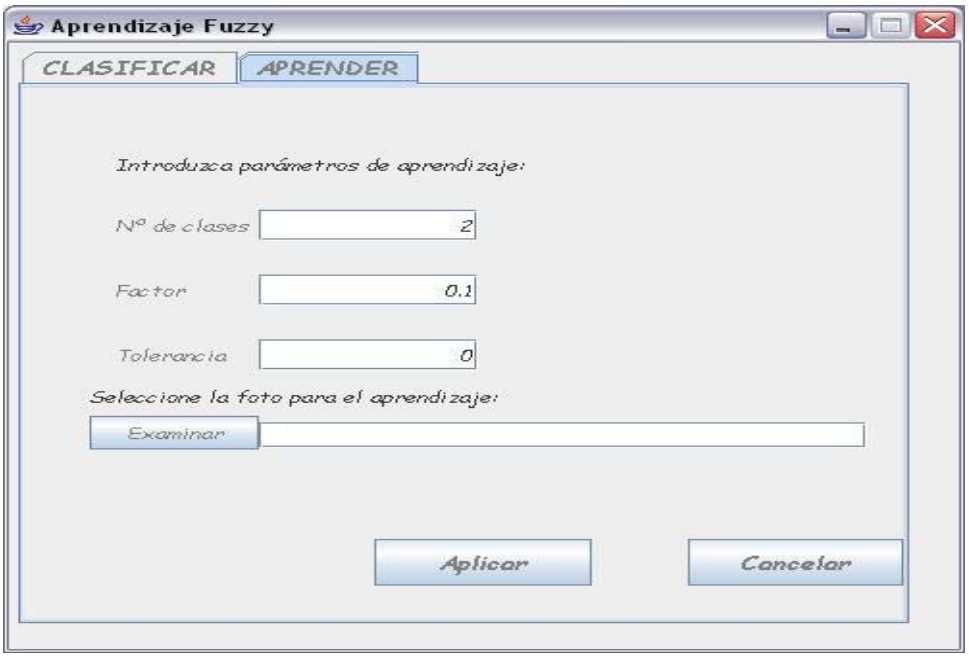
4.3 Interfaz final

Una vez que el algoritmo de Lloyd se ejecutaba correctamente, podíamos asegurar al otro grupo que cualquiera de los otros dos algoritmos se integrarían correctamente y que las funcionalidades que ellos requerían estaban ya implementadas.

Al ser los tres algoritmos muy parecidos en cuanto al proceso de recepción de parámetros era de esperar que sus necesidades lo fueran también, y por lo tanto lo que era bueno para uno lo sería para los demás. Es por esto que decidimos reutilizar código y partes ya empleadas en la integración de ese primer algoritmo. Aunque los parámetros eran distintos, las funciones eran básicamente las mismas desde el punto de vista del interfaz. Todos los algoritmos clasificaban una imagen que tenía que ser pasada como parámetro, lo único que cambiaba era el modo de hacerlo, por lo que la única diferencia era la necesidad de unos parámetros u otros.

Por lo tanto la ventana que decidimos utilizar para los algoritmos Fuzzy y bayesiano era muy similar a la de Lloyd, con los cambios pertinentes para el manejo de la información.

La ventana tenía este aspecto:



Como puede observarse en la imagen, estos algoritmos, que a nivel gráfico son exactamente iguales por requerir datos similares y proporcionar al usuario opciones también similares en todos ellos está presente el aprendizaje. Este aprendizaje planteó una serie de problemas que se tratarán más adelante, en la sección "Problemática".

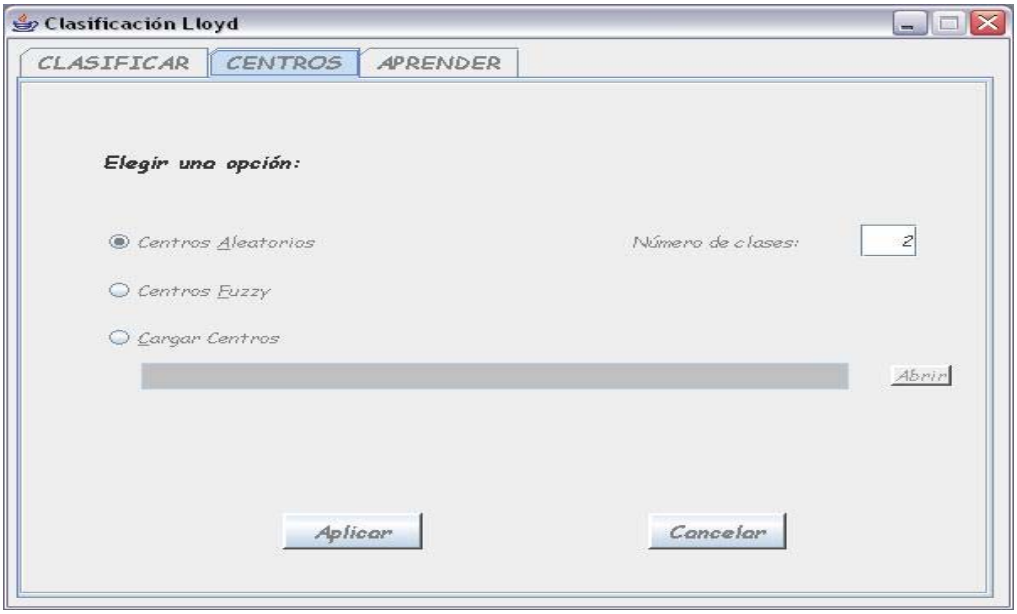
Ya éramos capaces de abrir una o más imágenes, viendo una pequeña vista previa de cada una, elegir de entre ellas la que queríamos tratar y utilizarla para hacer la clasificación utilizando cualquiera de los tres algoritmos implementados. Pero además estas imágenes se podían utilizar también para que los algoritmos aprendiesen y obtener así una clasificación mejor en el futuro. La aplicación, como es obvio, también era capaz de guardar los resultados obtenidos, tanto de la clasificación, como del aprendizaje.

En resumen, la aplicación era, a nivel funcional, totalmente operativa, ya que realizaba todas las funciones para las cuales había sido pensada en relación a los métodos de clasificación.

Pero faltaban aún algunos retoques, y alguna de las funcionalidades debían ser revisadas, sobre todo en el algoritmo de Lloyd, que al haber sido el primero, estaba ahora un poco olvidado.

Se añadió un apartado en la opción de los centros donde el usuario debía escoger el número de clases en las que se debía clasificar (por defecto dos).

La ventana final para el algoritmo de Lloyd se muestra a continuación:



Se muestra solamente la ventana de los centros porque es la única sobre la que se realizaron cambios prácticos. Ahora el usuario debe elegir primero los centros, con los distintos métodos que se le facilitan, y el número de clases en las que quiere que se clasifique la imagen seleccionada. Después de presionar el botón "*aplicar*" debe cambiar de pestaña a la de clasificar.

Si lo que quiere es utilizar la imagen para aprender basta con que vaya directamente a la pestaña aprender y utilice el menú de navegación para seleccionar la imagen que se utilizará para el aprendizaje.

Obviamente, la barra principal también sufrió algún cambio. Se añadieron botones para las nuevas funcionalidades y se cambió el aspecto gráfico de los que ya estaban implementados, añadiendo iconos a los botones para hacerlos lo más intuitivos posible.

El aspecto que presentaba la barra de herramientas en su versión final es la que se muestra a continuación:

Se observar que de un simple vistazo el usuario reconoce las funciones que la aplicación es capaz de realizar. Además el orden de ejecución que debe seguirse es también muy intuitivo. Las opciones que requieren de alguna operación previa permanecen desactivadas hasta que se realiza esa operación, como en el caso que se muestra en la figura, en que la única operación posible es la de apertura de una imagen. Cuando esta operación se realiza correctamente se activan las demás

opciones disponibles: guardar imagen, algoritmo de Lloyd, algoritmo de Fuzzy o algoritmo Bayesiano.

Y lo mismo ocurre cuando se selecciona alguno de los métodos proporcionados por la aplicación. Si alguno de ellos necesita alguna operación previa para clasificar, como pudiera ser un aprendizaje previo o la selección de algún centro o el número de clases, ésta estaría desactivada hasta que no se ejecutan dichas operaciones.

Obviamente todo este proceso intuitivo está además apoyado por un menú de ayuda y porque pequeñas ayudas emergentes que explican al usuario el por qué de esa situación, por ejemplo, la necesidad de aprender antes de clasificar por primera vez.

Problemática

Como se ha mencionado anteriormente, según se iba avanzando nos íbamos encontrando con nuevos problemas. La mayoría no eran muy serios, y eran ocasionados básicamente por el desconocimiento de algún método concreto, o por pequeños fallos fácilmente subsanables.

No obstante hubo uno en especial relacionado con el algoritmo Fuzzy, que generó mayores problemas

Este algoritmo clasificaba igual de bien que los demás, y en un tiempo más que razonable, para una imagen estándar de 600x800 la clasificación era prácticamente inmediata, de unos 2 segundos o incluso menos. El problema se encontraba en el aprendizaje. Y lo peor de todo no era eso, sino que además, el aprendizaje Fuzzy era necesario para calcular los centros para el algoritmo de Lloyd e influía en al aprendizaje del algoritmo bayesiano.

En un principio se vio que el tiempo necesario era demasiado, por lo que se concluyó que no era aceptable. Teóricamente eran necesarios más de veinte minutos para realizar un aprendizaje de una imagen de tamaño medio. Lo peor es que la mayoría de las veces que se intentó para abordar el problema el ordenador se quedaba bloqueado y había que cerrar la aplicación.

Era un mal momento para encontrar un error crítico, así que en una primera aproximación intentamos descifrar el código que producía tantísima espera, y vimos que este estaba basado en operaciones con matrices, que no sólo se replicaban varias veces, sino que además se anidaban, por lo que la complejidad se disparaba enormemente.

Decidimos que lo mejor era poner en conocimiento de tan funesto hallazgo al grupo de desarrollo de los algoritmos, que por fortuna ya conocían el problema, como era obvio, y estaban buscando una solución.

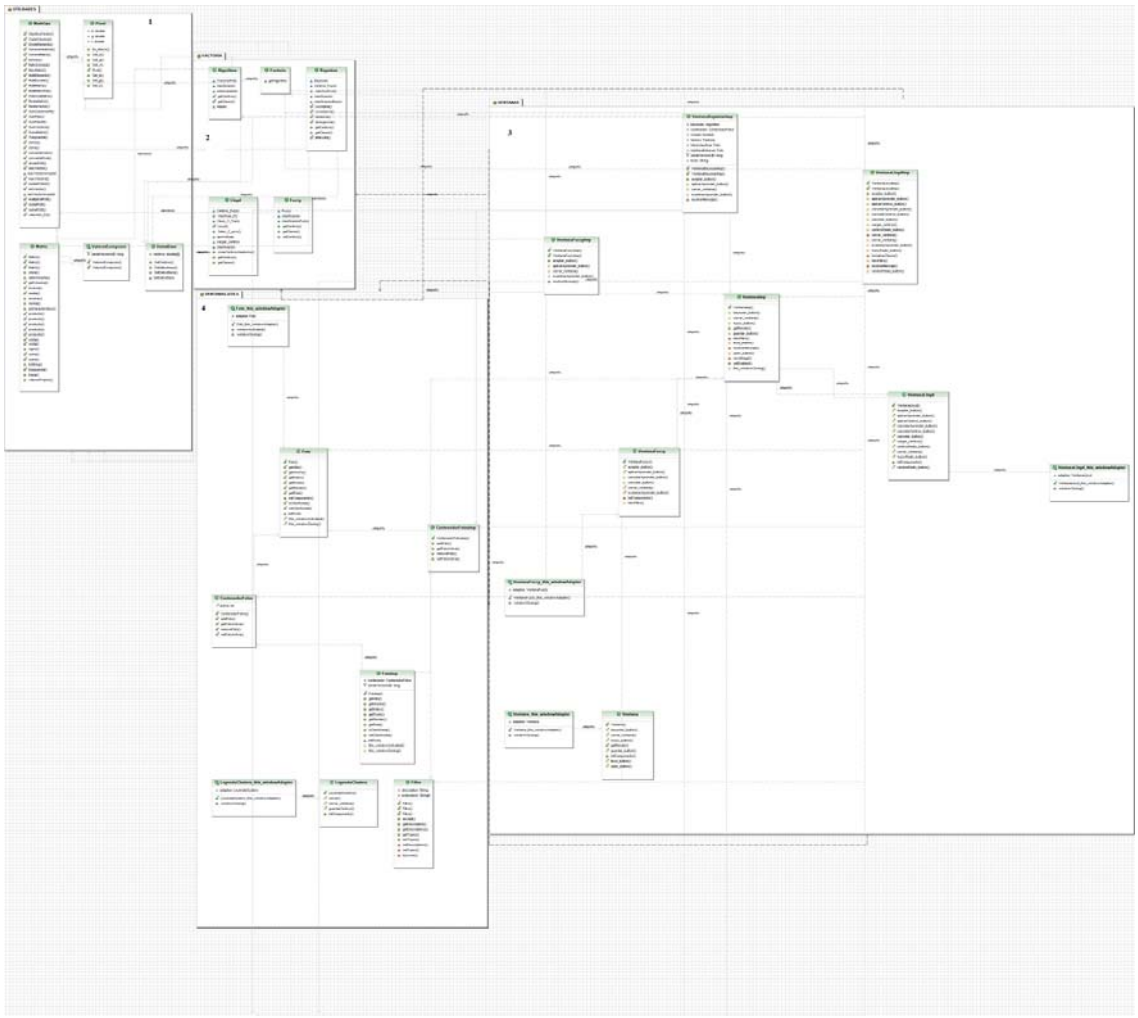
Por suerte ésta ni se hizo esperar mucho ni era muy difícil de implementar. Bastaba reducir el tamaño de la imagen que se utilizaba para aprender. El grupo de desarrollo, tras varias pruebas, concluyó que un tamaño para el que se tenía una respuesta aceptable era de 200x200.

Como era obvio ese tamaño era demasiado pequeño, y por otro lado no era agradable haber trabajado para hallar la manera de no limitar el tamaño de las imágenes que se podían tratar, y finalmente tener que limitarlo por otro motivo.

Así que se decidió buscar la manera de realizar el aprendizaje sin restringir el tamaño de la imagen que el usuario podía clasificar. Finalmente dimos con una solución bastante satisfactoria.

Decidimos que la imagen original sería mapeada de forma transparente para el usuario, y que el algoritmo utilizaría ese mapeo de la imagen para realizar el aprendizaje. Para ello bastaba tomar el tamaño de la imagen y dividirlo entre el tamaño que el otro grupo había obtenido como aceptable. El resultado nos daría el factor de mapeo. Ahora bastaba pasar los *n-ésimos* píxeles del vector que contenía la imagen original a otro vector, que contendría la imagen mapeada y sería el parámetro que se le pasaría al aprendizaje del algoritmo Fuzzy.

4.3.1 Diagramas UML

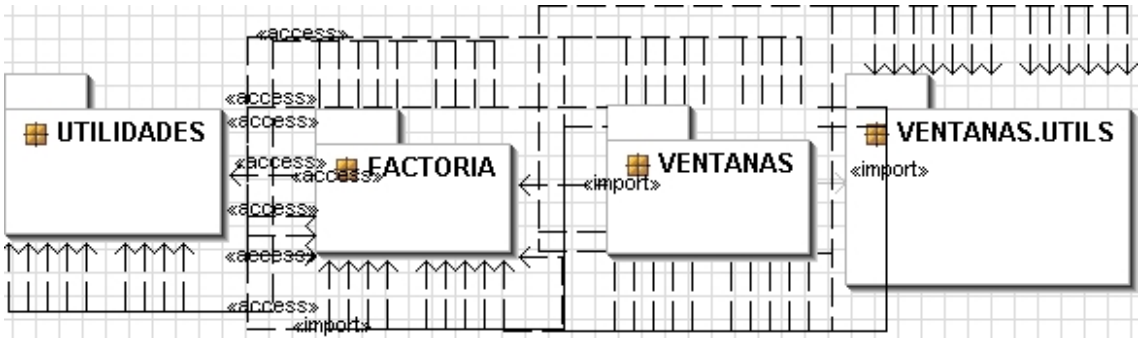


Explicación detallada

Debido al gran tamaño que presentan los diagramas UML se han representado solamente los métodos, constructores y las relaciones entre clases y paquetes, dejando ocultos los atributos que cada clase pudiera tener.

Además, como el diagrama general es demasiado amplio, para verlo en todo su contexto se ha tenido que reducir el tamaño, por lo que las clases apenas se ven. Por este motivo presentamos los diagramas esquemáticos de los cuatro paquetes que conforman el proyecto: 1) Utilidades, que contiene las clases y métodos para tratar los píxeles y las matrices asociadas a su manejo, 2) Factoría, que contiene las clases y métodos para el manejo de los algoritmos que se emplean en el proyecto, 3) Ventanas, que contiene las clases e implementaciones para manejar las ventanas de los distintos métodos y 4) Ventanas.Utils, que contiene las utilidades para el manejo de las imágenes, como por ejemplo las vistas previas, la leyenda de los clústeres, etc.

Se muestra a continuación la relación entre estos cuatro paquetes:

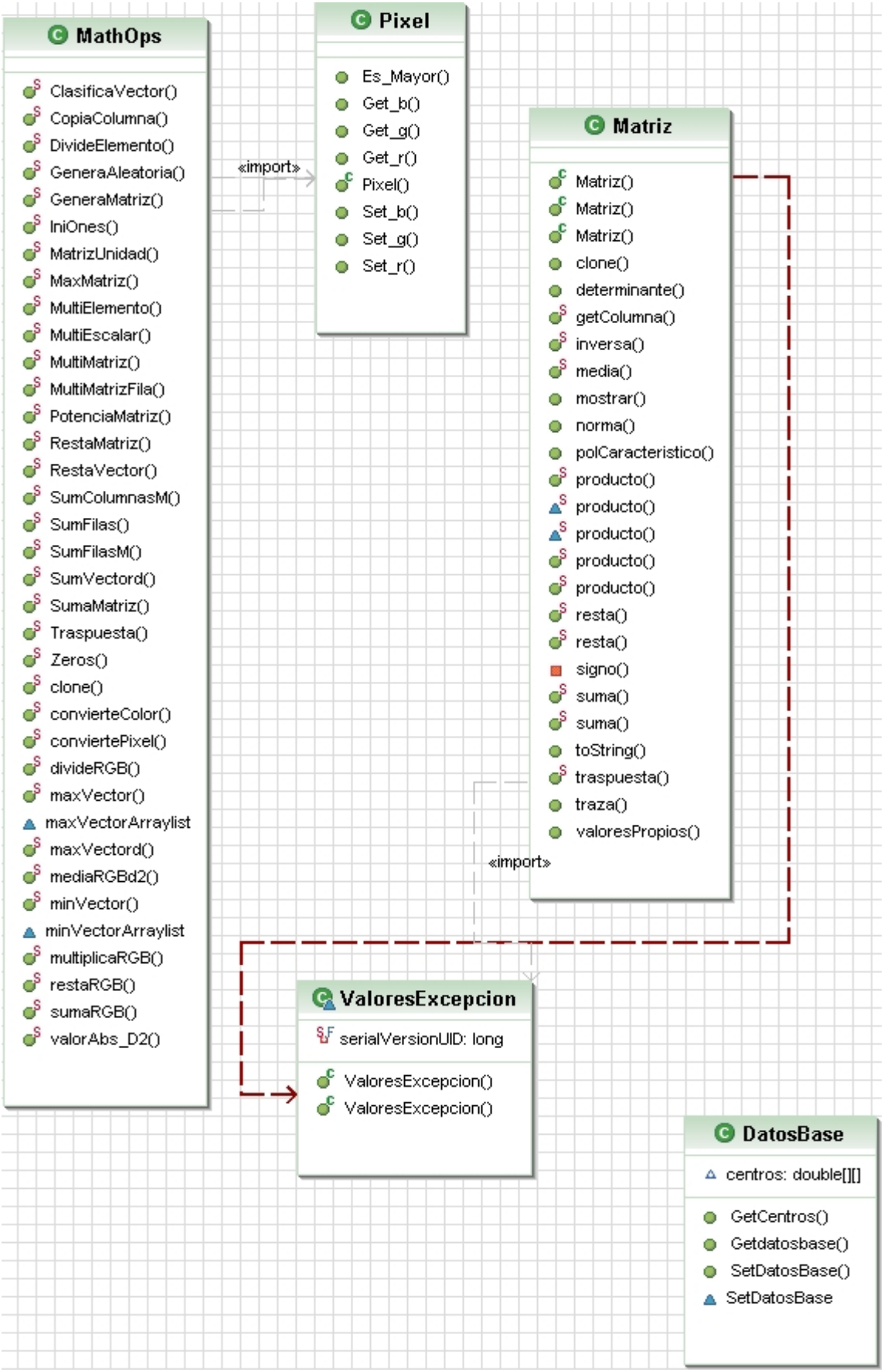


En el diagrama se muestran las importaciones que cada paquete hace de los restantes, a modo de resumen de cómo están representados en el diagrama general.

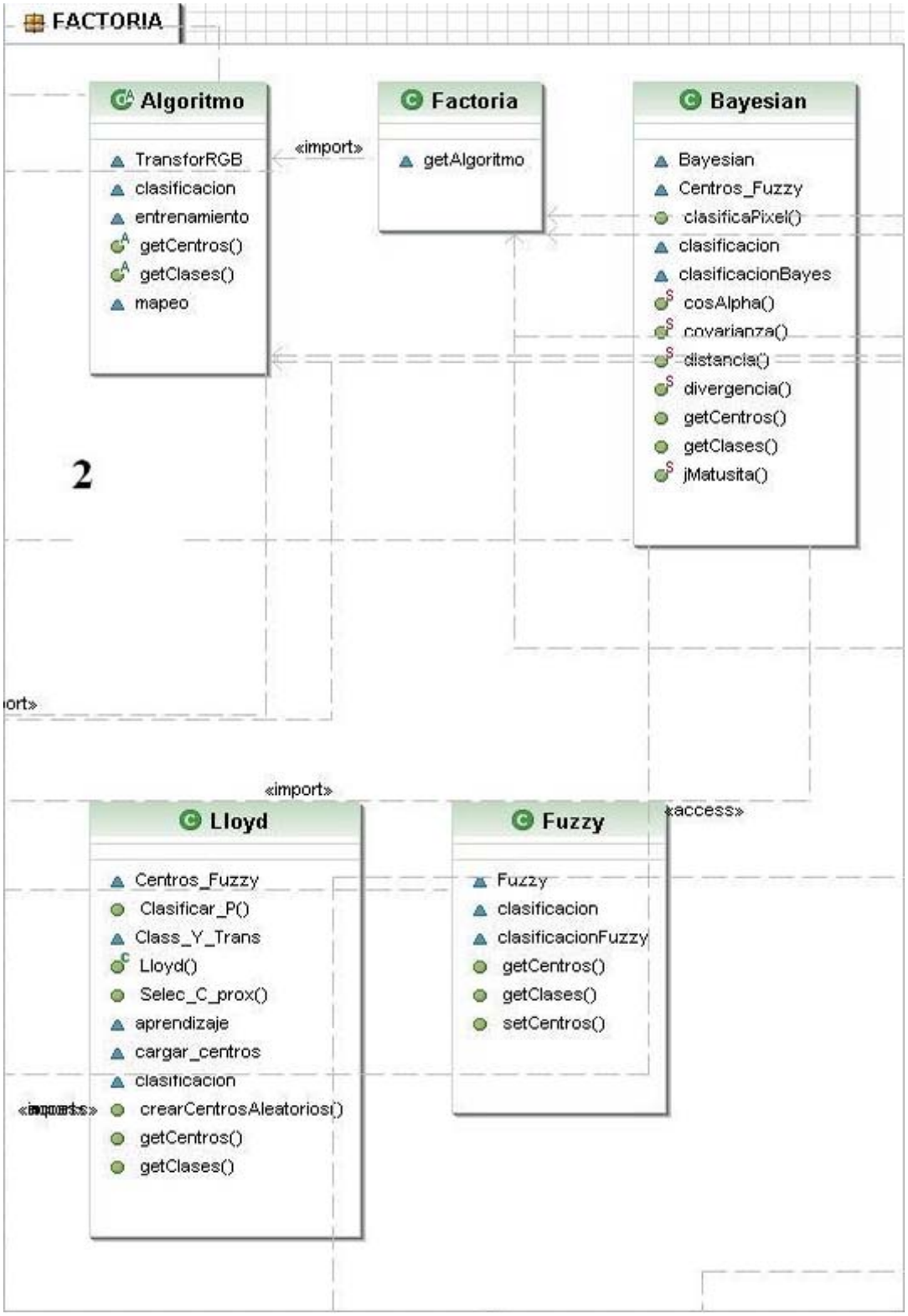
A continuación mostramos los diagramas específicos de cada paquete, mostrando como se ha hecho anteriormente, tan solo los métodos y dejando ocultos los atributos.

Las relaciones entre las clases vienen representadas por flechas direccionales que están marcadas con las palabras clave “import” y “access” dependiendo de cuáles sean dichas relaciones.

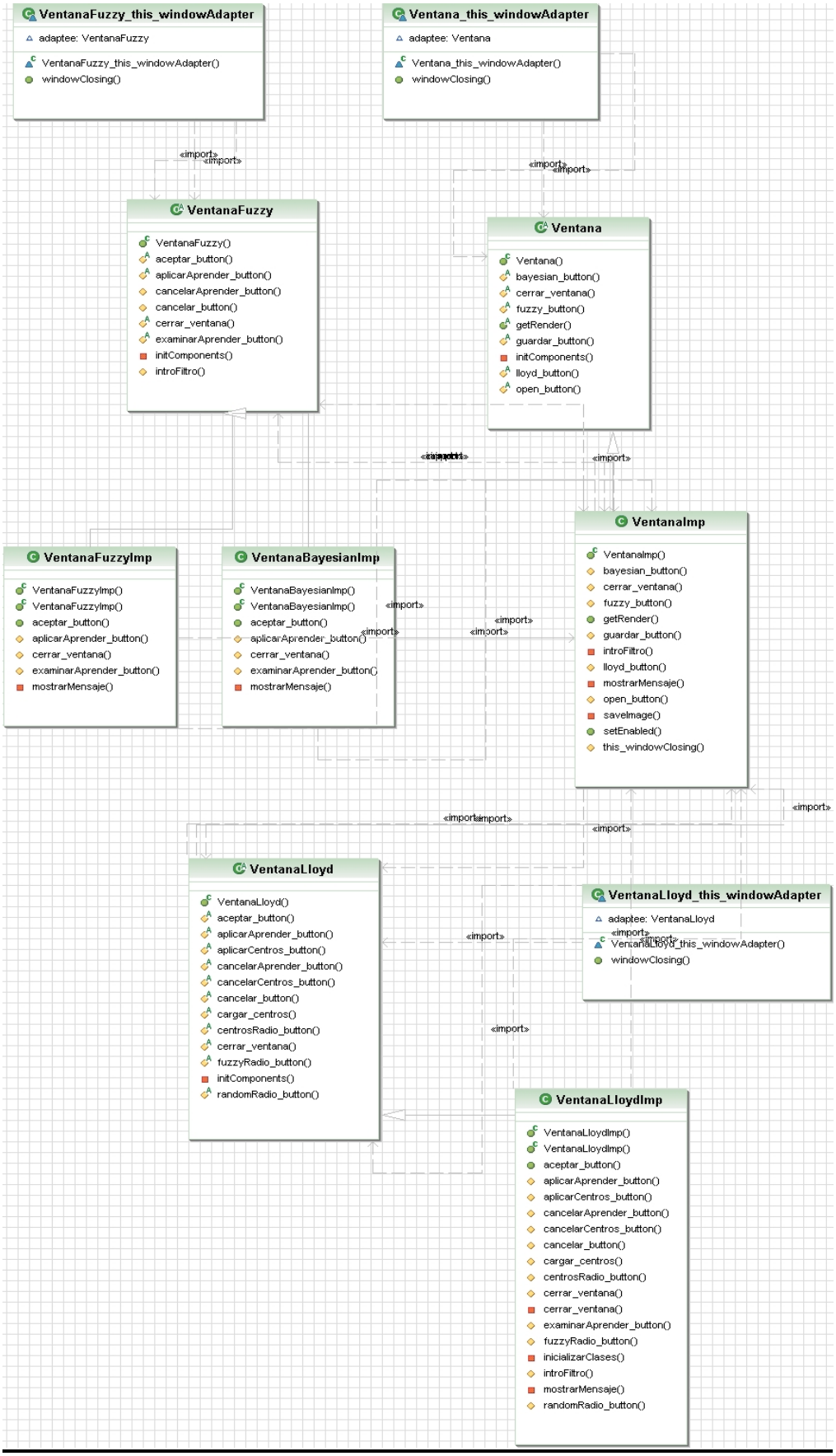
Paquete Utilidades:



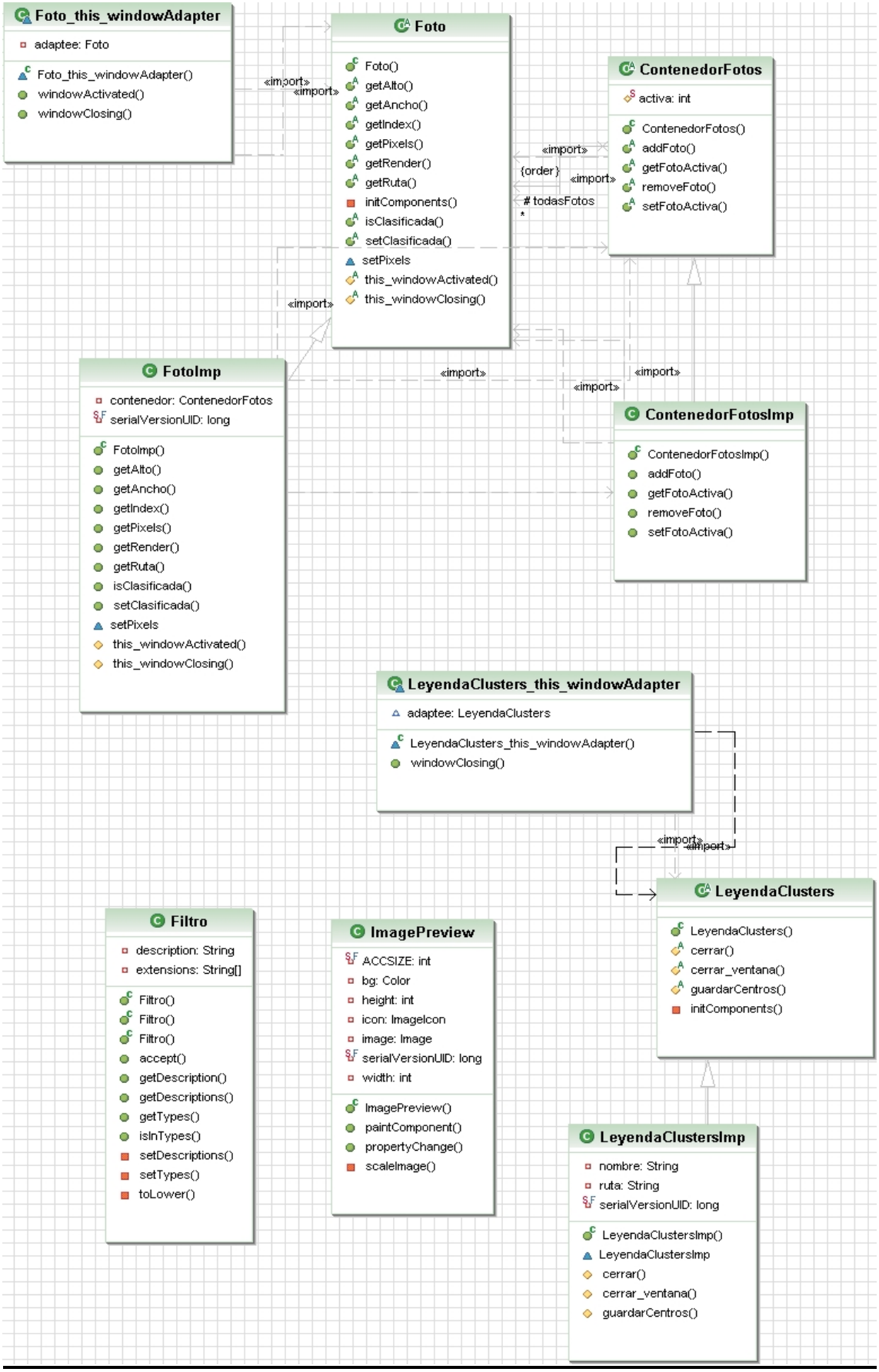
Paquete Factoría:



PaqueteVentanas :



Paquete Ventanas.utils :



5 Resultados

Una vez finalizadas todas las fases de desarrollo, se debía pasar a formalizar una valoración sobre la aplicación que había resultado. Para ello lo más importante era asegurar el correcto funcionamiento de todas y cada una de las funcionalidades que se ofrecían al usuario.

Con tal propósito se establecieron periodos de pruebas con sus respectivas correcciones, si se encontraban anomalías en el funcionamiento, o mejoras, si se consideraba que una funcionalidad no era lo suficientemente potente o intuitiva.

Como se ha expuesto anteriormente, cada grupo debía asegurar que su parte estaba totalmente libre de errores, y además, que la funcionalidad que se le requería estaba proporcionada dentro de unos parámetros de memoria y tiempo establecidos previamente como aceptables.

5.1 Pruebas de test

A continuación detallamos los tests que se llevaron a cabo después de cada fase o iteración; Como es lógico, la fase inicial no tenía pruebas asociadas puesto que estaba destinada prácticamente en su totalidad al diseño y documentación previa. Por otra parte, la corrección del código se comprueba inmediatamente durante la implementación.

Final de la fase de elaboración

Fecha: 1 –12 –2006

Al finalizar esta fase el grupo de interfaz tan sólo disponía de la parte concerniente al tratado de imágenes a nivel de usuario, esto es, abrir, guardar y mostrar una imagen en una ventana individual.

Como se puede comprobar en el apartado “Evolución por mejoras y necesidades” del capítulo 3, era un modelo muy simple, puramente funcional, y las pruebas que sobre él se hicieron fueron independientes, sin ninguna interacción con los algoritmos, que aún estaban en fase de diseño.

Aún así las pruebas no eran tan triviales como podía parecer. Como es habitual, debajo de la interfaz hay una funcionalidad de comunicación, y era ésta la que debía ser verificada.

Había que asegurarse que la forma que habíamos decidido para pasar una imagen era correcta y además, que la imagen que se pasaba era la que se pretendía y no otra.

Por lo tanto lo que se hizo fue un test “de suposición”, esto es, supusimos que al otro lado se encontraban ya implementados los algoritmos, de los que ya conocíamos los parámetros que necesitaban, y simulamos la interacción, comprobando que finalmente los datos llegaban correctamente y al lugar apropiado.

Final de la fase de construcción

Fecha: 20 – 1 –2007

Los objetivos propuestos para esta fase eran, entre otros, tener listo un primer prototipo sobre el que poder trabajar.

Como el grupo que desarrollaba los algoritmos nos los suministraba secuencialmente, nosotros fuimos desarrollando la interfaz según las necesidades y funcionalidades de los distintos algoritmos de acuerdo el orden que nos indicaba el otro grupo.

Así por ejemplo el primer algoritmo que nos llegó fue el de Lloyd, que tenía tres funcionalidades: clasificaba una imagen según las texturas, aprendía de imágenes para mejorar la clasificación y además daba opciones sobre los centros que podría utilizar.

Nosotros desarrollamos la interfaz para dar soporte a esas funciones y cuando se completaba una se realizaban pruebas independientes.

En realidad la interfaz no debía cambiar mucho de un algoritmo a otro, puesto que sólo diferían en el número de parámetros, dado que su comportamiento era transparente desde nuestro punto de vista.

Cuando el prototipo estuvo listo para dar soporte a toda la funcionalidad del algoritmo de Lloyd se pasó a hacer las pruebas conjuntas.

En esta fase se detectaron algunos problemas concernientes al tamaño de las imágenes debidos a falta de memoria, pero fueron rápidamente subsanados.

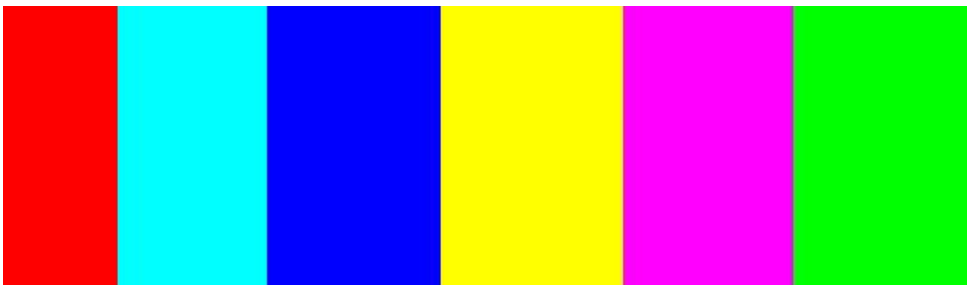
Hubo un problema que no fuimos capaces de resolver un problema que tenía su base en el funcionamiento interno del algoritmo y que por desgracia, era de difícil solución.

Cuando la aplicación debía clasificar en más de 4 clases había veces en las que se presentaba siempre una clase menos. Lo curioso era que en otras ocasiones la clasificación se realizaba perfectamente, con todas las clases.

La depuración de ese error nos llevó mucho tiempo y pocas soluciones. Al final, después de mucho deliberar, convinimos que el problema era de los centros. Había texturas para la que los centros eran los mismos, lo que quería decir que había dos clases que se solapaban en una única clase.

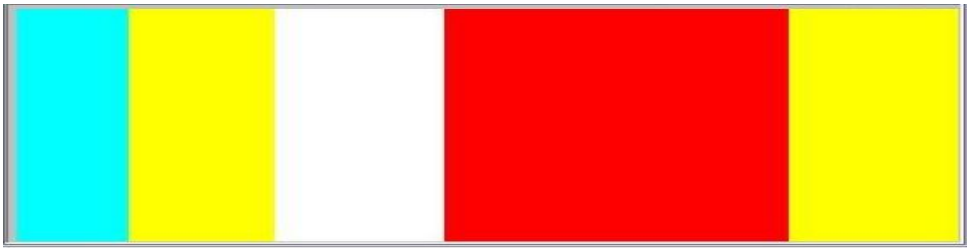
Para apoyar esta conjetura decidimos utilizar una imagen muy simple que contuviese un número de clases para el cual teníamos problemas, y que además éstas estuviesen claramente diferenciadas.

Un dibujo con bandas de 6 colores como el que se muestra:



Con esta prueba pudimos concluir que estábamos en lo cierto, puesto
que las clases que aparecían eran las que se especificaba. Además cuando los centros eran aleatorios y el algoritmo no había aprendido lo suficientemente bien, las seis clases se quedaban tan solo en 4; los colores más próximos concurrían en un único valor.

Esta situación se muestra a continuación:



Como puede observarse, las zonas más afines se han concentrado en torno a un mismo centro, haciendo imposible su reconocimiento.

Por lo tanto podíamos asegurar que el algoritmo funcionaba correctamente, y que ese error se debía solamente a la aleatoriedad.

Final de la fase de transición

Fecha: 1 –06 –2007

Debido a que esta fase era la que más carga práctica presentaba, los chequeos locales debían ser continuos. Para cada una de las necesidades o exigencias que se presentaban había que hacer un exhaustivo testeo.

Cuando todo estuvo listo y ensamblado se procedió a comprobar el buen funcionamiento de la aplicación en su conjunto, y para ello no había más que realizar las mismas pruebas que se habían realizado sobre el algoritmo de Lloyd para los demás algoritmos, y volver a realizar una batería de pruebas con las nuevas mejoras que se habían incorporado al algoritmo de Lloyd (primero en ser verificado).

Como se ha explicado detalladamente en la sección “problemática” del capítulo 4, nos encontramos con un gran problema de complejidad en el aprendizaje del método fuzzy. Y esto era un gran problema porque no solo afectaba al apartado del método fuzzy; los centros óptimos para el algoritmo de Lloyd se calculaban tras un aprendizaje del algoritmo de Fuzzy, por lo que nos vimos en la necesidad de volver a comprobar esa funcionalidad nueva, concluyendo que el tiempo empleado no era asumible.

La solución a ese problema se ha explicado anteriormente, y básicamente consistía en realizar un mapeo sobre la imagen original, para obtener una muestra aleatoria de menor tamaño que sirviese para realizar el aprendizaje.

5.2 Cumplimiento de objetivos

Establecimos a principio de curso que nuestra aplicación debería ser:

- Fiable
- Intuitiva
- Rápida
- Robusta
- Portable

Obviamente si queríamos dar por cumplidos todos los objetivos no bastaba con terminar una aplicación que realizase todas las tareas que se suponía debía realizar. Además debería hacerlo en un tiempo aceptable y con un margen de error muy pequeño. Otra de las condiciones fundamentales sería la tolerancia a fallos, esto es, que la aplicación en su conjunto no diese apenas problemas, ni de ejecución ni de consumo de recursos.

Podemos decir que hemos cumplido con estos objetivos que nos marcamos como primordiales:

- La aplicación es fiable, y así lo muestran los diversos test que sobre ella se han realizado con éxito

- Es muy intuitiva gracias a una interfaz agradable, concisa y fácil de utilizar. Además al usuario se le provee de ayudas emergentes y de explicaciones exhaustivas de cada algoritmo.

- La aplicación es lo bastante rápida como para que el usuario haga un uso normal de la misma sin que éste entorpezca otros trabajos que el usuario éste realizando en ese momento. Esto ha sido posible gracias a la atención que se ha prestado a la optimización de los recursos.

- La robustez se ha conseguido con un código bien organizado y modulado y haciendo uso de una factoría de algoritmos, que además aportan claridad.

- La portabilidad fue una de las cualidades que hicieron que nos decantásemos por utilizar el lenguaje JAVA, cuando aún teníamos algunas dudas sobre la plataforma de desarrollo.

A nivel de cualidades deseables para la aplicación se habían cumplido los objetivos, pero lo verdaderamente importante era si estos se habrían cumplido también a nivel de resultados, esto es, ¿realizaría la aplicación la función para la que había sido creada de manera satisfactoria?

La única manera que tenemos de contestar a esta pregunta es mostrando los resultados que se obtienen tomando una imagen al azar; Por lo tanto nos pareció una buena idea realizar esta comprobación con una imagen aérea de un terreno irregular con diversas texturas.

Realizaremos la siguiente prueba. Utilizaremos la imagen que se muestra a continuación, y la someteremos a todos los algoritmos con una precisión de dos

clases. Después, y para ver tanto el contraste como el grado de precisión que se obtiene en la clasificación, utilizaremos ocho clases.

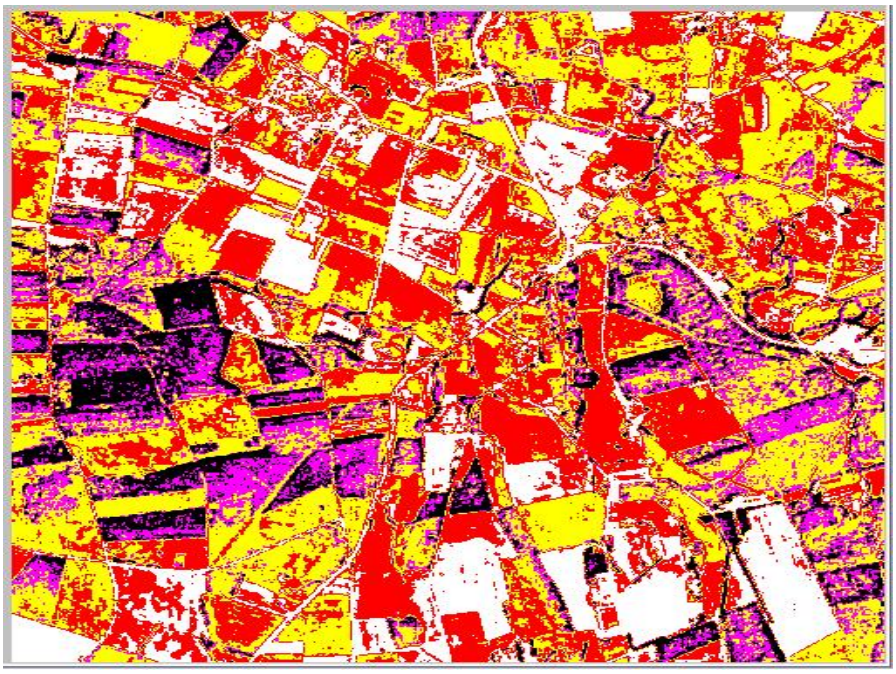
Como se puede comprobar, la imagen es verdaderamente irregular, y presenta muchas zonas con texturas mixtas y variadas.

A continuación se muestra el resultado obtenido tras aplicar el algoritmo con dos clases:

Se puede apreciar la diferenciación que ha resultado de la clasificación. Esta clasificación es correcta, incluso se podría decir que precisa, pero el rango de clasificación es muy bajo. Como sólo se dispone de dos valores, todo lo que se parece acaba siendo de la misma clase. Si la imagen es muy heterogénea y no se requiere un grado alto de diferenciación, esta podría ser una solución válida.

No obstante esto no suele ser lo habitual.

El resultado de aplicar el algoritmo con cinco clases a la misma imagen original es el siguiente:



Como se puede observar, el resultado obtenido es mucho más preciso, llegándose a distinguir clases que en la imagen original costaba mucho diferenciar, o que simplemente no era posible diferenciar.

Es posible realizar la clasificación hasta con ocho clases distintas.

Por lo tanto, podemos concluir que el aspecto funcional, se ha cumplido con lo exigido en la especificación previa. No obstante, daremos una muestra de una funcionalidad que se encuentra un poco más escondida, en segundo plano: el aprendizaje.

Mostraremos con un ejemplo cómo nuestra aplicación es capaz de aprender y de realizar una clasificación cada vez mejor. Para esto realizaremos una clasificación con los centros escogidos de forma aleatoria, con suficientes clases como para que se distinga perfectamente la texturación. Y después se realizará una clasificación idéntica para la misma imagen, pero esta vez con centros calculados tras realizar el aprendizaje.

Imagen sin aprendizaje

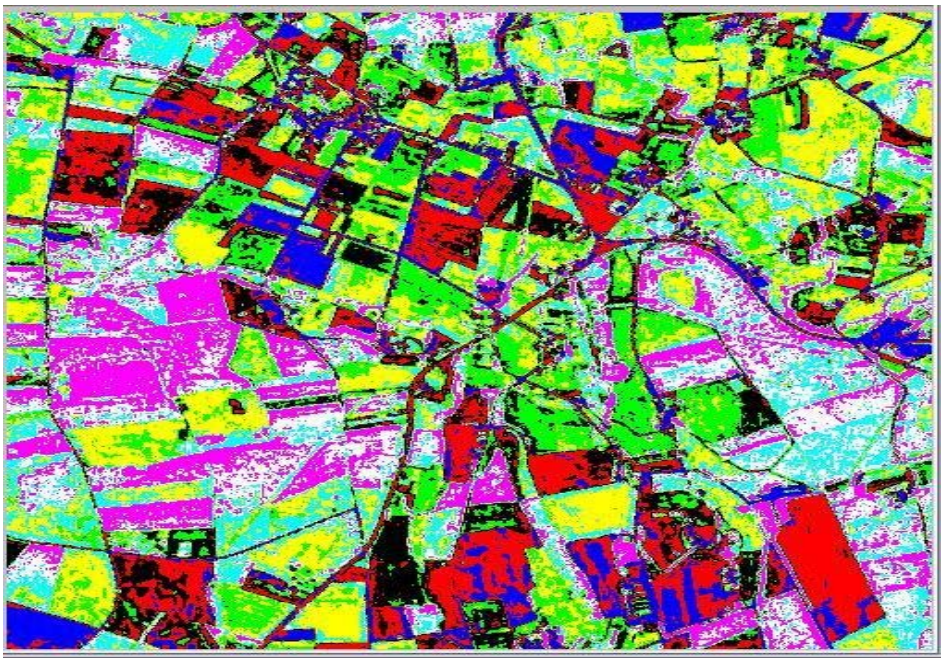
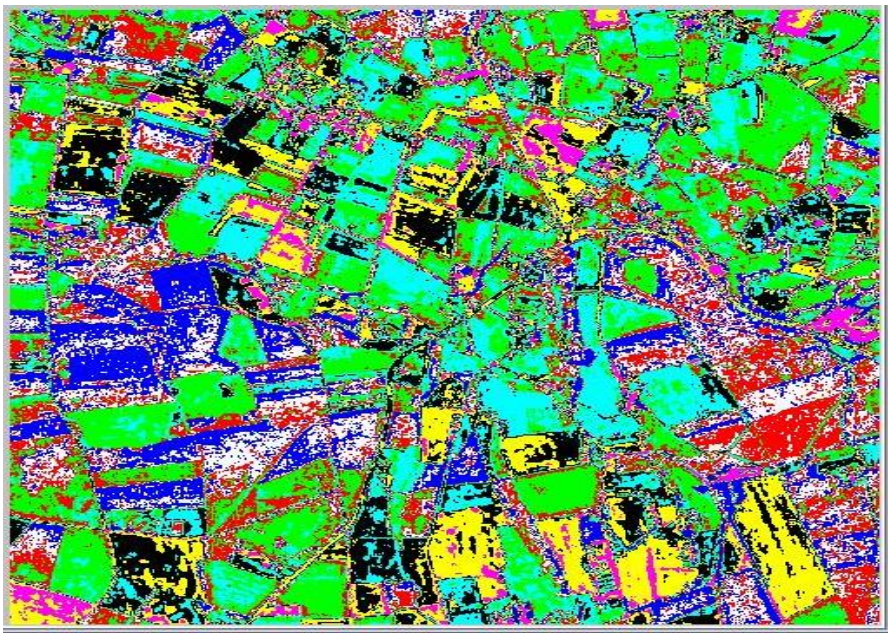


Imagen con aprendizaje



Se puede observar fácilmente si uno presta un poco de atención, que la precisión con la que se realiza la clasificación es mucho mayor después de haber realizado el aprendizaje. Es de notar también que cuantas más clases haya, mayor rendimiento se le sacará al aprendizaje, puesto que si nos limitamos a clasificar en blancos y negros, habrá mucha información que se perderá.

Hemos visto las diferencias que se muestran en la clasificación según se tomen más o menos clases, y también las mejoras que se presentan cuando se aplica el aprendizaje antes de clasificar.

Pero todo esto lo hemos visto tan sólo con el algoritmo de Lloyd, por ser el más versátil, puesto que es el único que, además de clasificar, permite elegir entre realizar una clasificación con centros elegidos al azar, o realizarla con centros obtenidos de un aprendizaje previo.

Pero aun siendo el más versátil no es el único, y por supuesto no tiene por qué ser el mejor. Por lo tanto lo que nos disponemos a mostrar a continuación es una comparativa práctica de cómo los diferentes algoritmos hacen la clasificación de una misma foto original. Para ello, y para que los tres algoritmos estén en igualdad de condiciones, vamos a realizar el aprendizaje con los mismos parámetros comunes en los algoritmos de Fuzzy y Bayesiano, y en el de Lloyd utilizaremos centros obtenidos a partir de un aprendizaje basado en la lógica de Fuzzy. Todos ellos hará una texturación con seis clases diferentes.

Como en las ocasiones precedentes utilizaremos una imagen de un terreno heterogéneo tomada desde el aire.

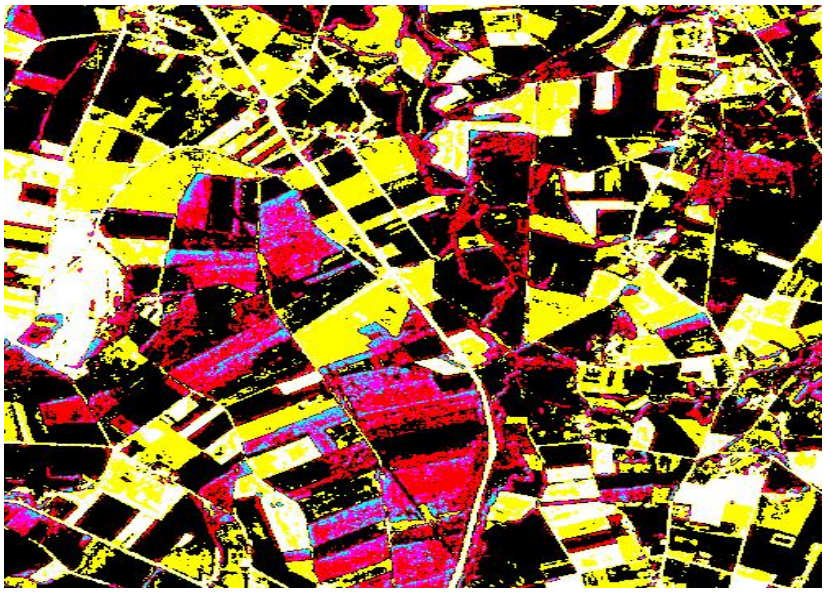
La imagen original es la siguiente:

Se pueden apreciar, además de las irregularidades en cuanto a zonas de color, zonas con distinta oscuridad dentro de un mismo color, colores que se entremezclan y zonas donde un color predominante contiene pequeñas zonas, a veces en forma de salpicaduras, de otro color menos intenso o viceversa.

En la imagen original es realmente costoso distinguirlo.

Veamos el resultado de las distintas clasificaciones:

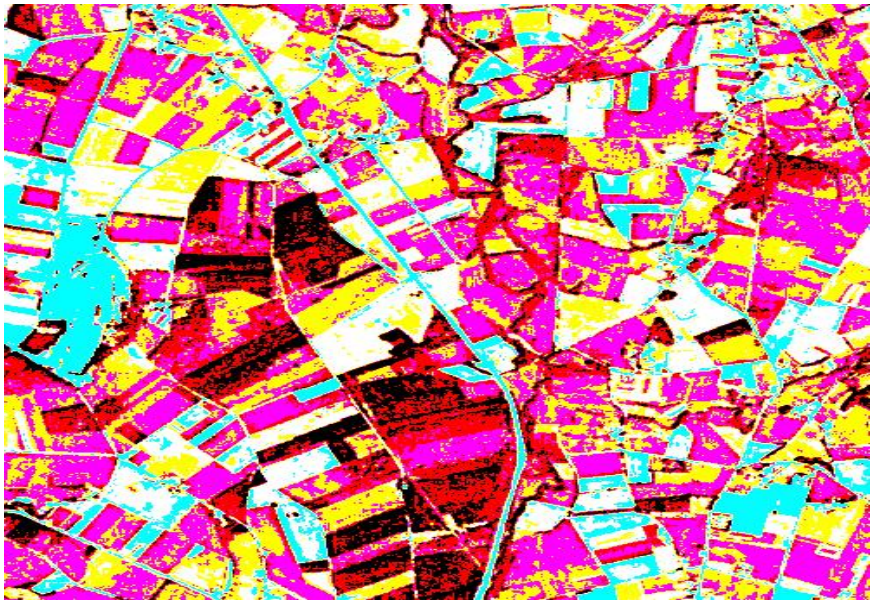
Clasificación mediante el algoritmo de **Lloyd:**



Se puede apreciar con bastante mayor claridad lo que se comentaba anteriormente. Zonas mixtas que en la imagen original eran muy difíciles de diferenciar.

Como se ve es una clasificación muy completa y precisa, y el único inconveniente que se le podría recriminar es que las zonas próximas al blanco en la imagen original pasen a ser también blancas en la imagen clasificada.

Clasificación mediante el algoritmo de **Bayes:**



Es difícil comparar estos resultados en tanto en cuanto los colores mostrados para las distintas clases difieren entre si. Pero si hacemos un pequeño esfuerzo de

visión y comparamos zonas pequeñas en ambas imágenes, veremos que la clasificación es básicamente la misma, y que en verdad lo único que cambia es la apreciación debido a los distintos colores que crean distintos contrastes al ojo humano; es más fácil, por ejemplo, distinguir el blanco sobre negro que el azul sobre amarillo.

Clasificación mediante el algoritmo de **Fuzzy:**

Teóricamente el algoritmo de Fuzzy es el más preciso de los tres, y no en vano el algoritmo de Lloyd utiliza el aprendizaje de Fuzzy para calcular los centros de manera optima.

Parece que en la práctica se cumple también que el algoritmo de Fuzzy realice una clasificación más precisa y con más detalles.

Esto puede plantear un problema por lo que decíamos antes, al ojo humano le resultará bastante más difícil examinar los resultados del algoritmo de Fuzzy por la sensación caótica que transmite tanto detalle de las diferentes texturas.

A nivel de la interfaz gráfica la elección de uno u otro algoritmo es irrelevante, ya que el comportamiento para todos ellos es exactamente el mismo. Todos toman prácticamente los mismos parámetros. Salvo tal vez el algoritmo de Lloyd, que proporciona al usuario la posibilidad de manipular los centros, los demás no presentan ninguna diferencia en cuanto a la funcionalidad o a la manera en la que esta se presenta.

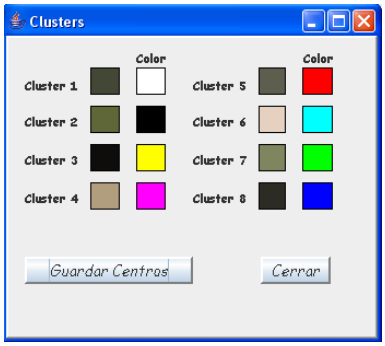
El manejo intuitivo que se propuso desde un principio ha sido claramente alcanzado para los tres algoritmos, así que la ejecución de uno frente al resto no presenta diferencias de cara al manejo por parte del usuario.

Las imágenes se obtienen del mismo modo en los tres algoritmos y se representan de igual forma. Así mismo, tanto la presentación de los resultados como el mecanismo para salvarlos es idéntico. La interfaz para los algoritmos de Fuzzy y Bayesiano no presentan diferencias, siendo la anteriormente citada de los centros, la única que presenta el método de Lloyd. Por lo tanto, a nivel de interfaz, la elección es totalmente indiferente.

Requisitos de última hora

Debido a la buena coordinación que hubo entre los dos grupos la planificación se cumplió estrictamente, por lo que antes de la entrega final tuvimos tiempo para satisfacer un último requisito formulado por el profesor director del proyecto.

Este consistía en mostrar la correspondencia de clases (clústeres) después de la clasificación, de modo que una vez texturada una imagen original, junto con la imagen clasificada apareciese una pequeña ventana que mostrase dicha correspondencia, tal y como se muestra a continuación:



6 Conclusiones y perspectivas de futuro

Una vez finalizado el proyecto es momento de sacar conclusiones.

A nivel de proyecto el resultado final ha sido bastante satisfactorio. Los algoritmos se ejecutan en un tiempo aceptable y con resultados muy satisfactorios, llegando casi a un nivel profesional.

Además, la aplicación resultante es de muy fácil uso gracias a una interfaz gráfica muy intuitiva y amigable.

No obstante, la aplicación es susceptible de mejoras, por lo que el proyecto puede ser mejorado y ampliado en el futuro.

A nivel de funcionalidad se podrían plantear varias mejoras, como por ejemplo, la posibilidad de utilizar otros algoritmos para la clasificación de texturas, o ampliar las funciones de la aplicación, de tal forma que el proyecto fuese algo más que un clasificador. Se podría incluso pensar en un generador de imágenes.

A nivel de la interfaz ésta se podría mejorar estéticamente, dotándola de una apariencia aún más profesional.

Pero la interfaz no es sólo estética, también es funcional. Por este motivo también es susceptible de mejoras funcionales, como podrían ser una barra de progreso en las esperas, que no obstante sería complicado debido a la sincronización, un mecanismo de zoom para ver las imágenes a mayor o menor tamaño, un sistema de comparación entre diversas imágenes, como por ejemplo entre dos clasificaciones de la misma imagen mediante dos métodos distintos o mediante el mismo método con distintos parámetros, selección de la correspondencia de clusters, posibilidad de elegir clusters principales, etc.

7 Bibliografía

- Documentación de la librería JAI:

Introducción
http://java.sun.com/products/java-media/jai/forDevelopers/jai1_0_1guide-unc/Introduction.doc.html

http://www.inf.ufrgs.br/~revista/docs/rita11/rita_v11_n1_p93a124.pdf

http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/index.html

http://java.sun.com/products/java-media/jai/forDevelopers/jai-imageio-1_0_01-fcs-docs/index.html

- Documentación de las librerías de JAVA

API
http://java.sun.com/j2se/1.5/docs/api/

API para desarrolladores
http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/index.html

Clase Components
http://java.sun.com/docs/books/tutorial/uiswing/components/

Clase Image IO
http://java.sun.com/j2se/1.4.2/docs/guide/imageio/spec/imageio_guideTOC.fm.html

Barras de progreso
http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/parte14/cap14-11.html

Menus
http://java.sun.com/docs/books/tutorial/uiswing/components/menu.html

Apéndice I Breve manual de usuario

Punto de partida

En la barra principal de la aplicación aparecen 5 funcionalidades. Al principio de cada ejecución se mostrará operativa sólo la opción de abrir.

Haga clic en el botón de abrir (icono con una carpeta)

Una vez presionado el botón aparecerá un diálogo de apertura. Utilizando el explorador seleccione la imagen que desee tratar. Puede ayudarse de la vista previa que se muestra en el recuadro de la derecha. Después de seleccionar la imagen presione “Aceptar”.

Se abrirá una nueva ventana que contendrá la imagen seleccionada.

Eligiendo el método

Una vez se haya abierto la imagen que se quiere tratar (si se abrieron varias se tratará la última en la que se hizo “clic”) se habilitarán el resto de las funcionalidades de la barra principal. Los métodos disponibles están marcados con iconos del mismo nombre. Seleccione el que desea emplear.

Método Lloyd

Si selecciona el método Lloyd se abrirá ante usted una ventana que contiene tres pestañas. En la primera, etiquetada como clasificar, encontrará una breve explicación.

Lo primero que debe hacer es decidir sobre los centros que empleará. Tiene tres opciones: escogerlos de forma aleatoria (más rápido), utilizar el algoritmo de Fuzzy (más preciso) o cargar centros que se hayan calculado previamente.

Una vez seleccionados los centros, y en la misma pestaña, deberá indicar el número de clases (texturas que se verán de distinto color).

Estos datos son necesarios tanto para clasificar como para el aprendizaje, que se realizará en la pestaña marcada como “aprender”. En ella encontraremos 3 valores, que están asignados por defecto. Si tiene alguna duda sobre los parámetros consulte la ayuda de la pestaña “clasificar”.

Cuando se han introducido todos los datos tan sólo falta ir a la pestaña “clasificar” y presionar el botón “aceptar”. Después de una corta espera aparecerá la imagen resultante de la clasificación y una leyenda con la correspondencia de clases.

Método Fuzzy

Si selecciona la opción del método Fuzzy se abrirá una ventana que tan sólo contendrá dos pestañas, una marcada como “aprender”, en la que se deberán introducir los datos requeridos (están asignados por defecto) y la imagen que se desea utilizar como modelo, y otra marcada como “clasificar”. En esta última encontrará una breve explicación sobre los parámetros de aprendizaje. Después de

realizar el aprendizaje tan sólo deberá presionar aceptar en la pestaña “clasificar” y el resultado se verá en una nueva ventana acompañado de una leyenda con la correspondencia de clases.

Método Bayesiano

Si selecciona la opción del método Bayesiano se abrirá una ventana que tan sólo contendrá dos pestañas, una marcada como “aprender”, en la que se deberán introducir los datos requeridos (están asignados por defecto) y la imagen que se desea utilizar como modelo, y otra marcada como “clasificar”. En esta última encontrará una breve explicación sobre los parámetros de aprendizaje. Después de realizar el aprendizaje tan sólo deberá presionar aceptar en la pestaña “clasificar” y el resultado se verá en una nueva ventana acompañado de una leyenda con la correspondencia de clases. (Análogo al método Fuzzy)

Guardando los resultados

Una vez haya obtenido la nueva ventana con la imagen clasificada podrá guardarla haciendo uso del botón de guardar (marcado con un icono en el que se ve un disco). Se abrirá ante usted un diálogo de guardar. Utilice el explorador para seleccionar la ubicación que más le convenga.

Apéndice II Contenido del CD

- Código del proyecto
- Memoria
- Imágenes de muestra