Nessie, reconocedor óptico de texto en recortes de prensa escrita

Eliezer Talón Socorro

Tutor: Alexis Quesada Arencibia

Co-tutor: José C. Rodríguez Rodríguez

Universidad de Las Palmas de Gran Canaria — Septiembre 2009

A Josabet

Search for beauty, find your shore
Try to save them all, bleed no more
You have such oceans within
In the end, I will always love you

(Tuomas Huolopainen)

Contenidos

Índice de figuras

Tablas

Prefacio

Existe un gran interés de parte de muchas organizaciones y empresas por estar informadas en todo momento. Desde comienzos del siglo XIX el volumen de información disponible a través de los medios de comunicación ha ido creciendo, tanto que procesarla en su totalidad se ha convertido en una tarea muy compleja. Esto motivó el nacimiento de un nuevo mercado en torno a la selección y entrega personalizada de información: el *seguimiento de medios*. En pocas palabras, este modelo negocio se basa en un grupo de organizaciones que se suscriben a un servicio externo para disponer cada día sólo de la información de su interés, recopilada y clasificada tras filtrar el resto de noticias.

Las primeras empresas que comenzaron prestando este servicio recurrieron a la figura del documentalista, una persona que se sentaba a hojear páginas de periódico para recortar los artículos y clasificarlos en archivadores para su posterior entrega por correo o un medio similar. Pronto se dieron cuenta de que el modelo se quedaba obsoleto a medida que el número de medios a cubrir crecía, hasta que con el avance de la tecnología empezaron a implantarse sistemas automáticos de búsqueda, selección y entrega. El proceso manual fue evolucionando progresivamente con la introducción de escáneres, sistemas de visión por computador y almacenamiento en bases de datos. Hasta que actualmente las empresas de seguimiento de medios demandan sistemas automáticos de segmentación, reconocimiento y clasificación, que ofrezcan niveles de velocidad y fiabilidad cada vez mayores.

No resulta difícil entender que las técnicas de reconocimiento óptico de caracteres (OCR[1]), son una herramienta fundamental dentro de todo el proceso de automatización. A fin de cuentas es el mecanismo que permite traducir las imágenes en palabras, rescatando el contenido que verdaderamente interesa al cliente.

El *reconocimiento óptico de caracteres* es el resultado de aplicar de manera específica técnicas de clasificación de patrones, una disciplina que representa un eje fundamental dentro de la inteligencia artificial y la visión por computador. Identificar patrones diferenciando unos de otros es una tarea que el ser humano tiene perfectamente automatizada, a través de su compleja red sensorial, sus mecanismos de abstracción y su extraordinaria capacidad de razonamiento. Gracias a ellos hemos conseguido sobrevivir a lo largo de toda nuestra existencia. Dotar a una máquina de ese comportamiento constituye una tarea cuanto menos ambiciosa y compleja, pero en un dominio más reducido ciertos mecanismos sí pueden ser imitados.

Una de las aplicaciones más inmediatas de los OCR consiste en aumentar el rendimiento de procesos industriales, o evitar en mayor o menor medida la inversión de recursos humanos

[1] Acrónimo del término inglés "Optical Character Recognition"

en tareas que son tediosas, monótonas o muy sencillas de realizar. Por ejemplo, supervisar la impresión de la fecha de caducidad en una cadena de producción de latas de refresco se convertiría en un proceso muy lento si fuera llevado a cabo por un operario. Tampoco resulta rentable colocar a un vigilante a la entrada de un aparcamiento sólo para anotar las matrículas de los vehículos, cuando puede hacerlo un sistema controlado de cámaras automáticamente.

En este documento presentamos el Proyecto Fin de Carrera "Nessie, reconocedor óptico de caracteres en recortes de prensa escrita", desarrollado para la Facultad de Informática de la Universidad de Las Palmas de Gran Canaria (ULPGC). El objetivo principal de este trabajo consiste en el análisis y diseño de una biblioteca de funciones que realice operaciones básicas de reconocimiento de caracteres, como soporte a un sistema más complejo que pretende optimizar el funcionamiento de una empresa de seguimiento de medios. Los tutores del proyecto son Alexis Quesada Arencibia y José Carlos Rodríguez Rodríguez, ambos profesores del Departamento de Informática y Sistemas (DIS) y miembros del Instituto Universitario de Ciencias y Tecnologías Cibernéticas (IUCTC).

La idea de este proyecto nace como respuesta a una incipiente demanda del sector en Canarias, cuyo crecimiento es muy lento debido al alto coste que tienen otras soluciones presentes en el mercado. El software que se desarrolla actualmente está pensado para grandes compañías que cuentan con clientes en todo el mundo, procesando medios de comunicación de varios países. Por ello, una empresa en la que sus clientes sólo necesitan conocer un conjunto reducido de periódicos no tiene la solvencia necesaria para implantar un sistema a gran escala. El proyecto *Nessie* ha sido concebido con la idea de proveer una solución menos compleja y costosa, asequible para empresas cuyo radio de actuación es reducido.

Colaboración externa

Este proyecto se asienta dentro de un entorno multidisciplinar, en el que las fuentes de información son variadas y abundantes. Hay que poner en común la experiencia de periodistas, expertos en seguimiento de medios, ingenieros e imprentas, realizar un filtrado de la información y elaborar una base de conocimiento que permita responder en la medida de lo posible cualquier duda eventual.

Por suerte hemos contado con la colaboración de varias entidades para suplir la falta de información sobre aquellos aspectos que se escapan al ámbito de la Ingeniería Informática. A continuación citamos las dos más importantes:

- Marta Cantero Lleó, de la empresa Entrelíneas Canarias S.L., fue consultada como experta en seguimiento de medios. Facilitó los contactos con el resto de entidades y nos introdujo en el mundo del seguimiento de medios y la actividad periodística. También proveyó de conocimiento básico para consultar el resto de fuentes con el bagaje necesario, explicando en qué consiste el proceso de edición de un periódico y los factores que intervienen.

- Federico González Ramírez es licenciado en Periodismo y fue consultado como experto en prensa canaria. También nos explicó la influencia de los medios de comunicación en campos como la política y la economía y el interés de muchas entidades por suscribir un servicio de seguimiento de medios.

- Víctor Macías, de la Biblioteca General de la Universidad de Las Palmas de Gran Canaria (BULPGC), ha facilitado el suministro de ejemplares de periódicos utilizada como muestras

de prueba para el proyecto. La principal fuente de periódicos proviene de Jable, el servicio de archivo de prensa digital de la BULPGC.

Recursos adicionales

Junto con este documento se distribuye un CD-ROM, que contiene una copia de la memoria en formato digital y los siguientes recursos:

- `nessieocr.tgz`, archivo comprimido con el código fuente de la biblioteca desarrollada.
- `nessieocrDoc`, directorio con la documentación del código fuente en formato HTML.
- `tools`, directorio con las API necesarias para compilar NessieOcr.

Organización del documento

Después de todo el trabajo realizado en este proyecto, resulta complicado poner todas las partes sobre la mesa y decidir cómo presentarlo al lector. Nos ha parecido que la siguiente estructura es la que más facilita la comprensión de todo el contenido.

La primera parte de este documento recoge toda la información recopilada durante la fase de análisis de requisitos, desde que se gestó la idea del proyecto hasta que se decidió la funcionalidad del mismo, pasando por toda una labor de investigación sobre el dominio del problema.

Capítulo 1. Contexto tecnológico y comercial: Descripción del proyecto *Nessie*, el modelo de negocio del seguimiento de medios y el funcionamiento de la edición de prensa escrita.

Capítulo 2. Modelo de análisis: Formalización del modelo de dominio y el modelo de casos de uso. Descripción general de un sistema de clasificación de patrones y su aplicación al reconocimiento óptico de caracteres.

Capítulo 3. Preprocesamiento de imágenes: Análisis de las primera fase de un sistema de clasificación de patrones.

Capítulo 4. Clasificación de patrones: Análisis de las dos últimas fases de un sistema de clasificación de patrones.

En la segunda parte se expone el diseño de la arquitectura del software que permite la implementación de la biblioteca de funciones, junto con los resultados y conclusiones obtenidas tras realizar una serie de pruebas.

Capítulo 5. Arquitectura del software: Formalización del diseño del software y la interacción entre sus módulos.

Capítulo 6. Implementación y resultados: Presentación de algunos detalles de implementación relevantes y de los resultados de tiempo y precisión.

Capítulo 7. Conclusiones: Consideración final y planteamiento de líneas de trabajo futuras.

Agradecimientos

Quisiera agradecer a todos los que me han apoyado de una u otra manera para llegar hasta aquí. Este proyecto es la culminación del trabajo realizado, no sólo a lo largo de los últimos meses, sino durante los años previos estudiando Ingeniería Técnica en Informática de Sistemas y la propia Ingeniería Informática.

A mis padres, por haber confiado en mi trabajo y apoyar siempre todas las decisiones que tomaba. Por su paciencia en las temporadas en las que prácticamente sólo estaba en casa para dormir. Por su soporte económico para que todo saliera adelante.

A Josabet García, por apoyarme en todo momento. Por escuchar cuantas veces necesitaba contar las vicisitudes diarias. Por animarme con una sonrisa y ayudarme a ver que hay otra realidad más allá de la carrera. Por soportar que no siempre estuviera allí.

Al grupo PL: Adexe Rivera, Adrián Peñate, Álvaro Lorenzo, Daniel Medina y Guanchor Ojeda. Por haber creado un ambiente de trabajo y compañerismo insuperable. Por tantas partidas en red y almuerzos juntos, haciendo más llevaderas las largas tardes de estudio.

A todos los que han pasado por *Bio* durante mi estancia allí: Mario Martín, Beatriz Domínguez y Yessica Hernández. Por estar siempre dispuestos a dejar el proyecto a un lado, relajar las piernas y asaltar la máquina de café.

A mis tutores: José Carlos Rodríguez y Alexis Quesada. Por ofrecer siempre un punto de vista crítico y honesto. Por estar siempre dispuestos a hablar y a aportar soluciones. Por poner todo tipo de facilidades para el desarrollo del proyecto.

Al personal de la Biblioteca de Informática: María Eugenia Rúa-Figueroa, Mercedes Celada y Teresa Samper. Por tramitar todas mis peticiones de libros. Por su profesionalidad y atención. Por atenderme siempre con una sonrisa y perdonar algún día de retraso.

Contacto

Aún cuando hemos puesto todo nuestro empeño en que toda la información de este documento sea correcta, es posible que hayamos podido cometer algún fallo. Para comunicar cualquier error, así como sugerencias o correcciones no dude en ponerse en contacto con el autor o con los tutores del proyecto:

- Eliezer Talón Socorro, en `elitalon@gmail.com`.
- Alexis Quesada Arencibia, en `aquesada@dis.ulpgc.es`.
- José Carlos Rodríguez Rodríguez, en `jcarlos@ciber.ulpgc.es`.

Parte I

Análisis de requisitos

1 Contexto tecnológico y comercial

La idea de implementar una biblioteca de funciones para realizar tareas de reconocimiento de caracteres surge como respuesta a la demanda de la industria del seguimiento de medios en Canarias. NessieOcr es un módulo integrado en una aplicación más compleja junto con otros subsistemas, cuyo desarrollo se enmarca dentro del proyecto *Nessie*. Esto ha llevado a que gran parte del diseño de NessieOcr esté influido por los requisitos funcionales de la aplicación global.

Entender cómo funciona el modelo de negocio del seguimiento de medios y el proceso de edición de prensa escrita es fundamental a la hora de analizar qué necesidades debe satisfacer Nessie como solución global. Como consecuencia, tener claro los requisitos del sistema completo nos conducirán a obtener la funcionalidad que debe proveer NessieOcr en particular. A lo largo de este capítulo veremos los aspectos más importantes que rodean el desarrollo de esta biblioteca, en relación a su contexto comercial y tecnológico.

1.1. El proyecto Nessie

El seguimiento de medios es un sector relativamente nuevo en Canarias, al menos en lo que se refiere a su explotación comercial por parte de empresas regionales. El Instituto Universitario de Ciencias y Tecnologías Cibernéticas (IUCTC) tuvo conocimiento de la necesidad de estas empresas de automatizar sus procesos mediante el soporte informático adecuado. Sin embargo existe una barrera económica que frena cualquier posibilidad de expansión: el alto coste de las soluciones que ya existen en el mercado. Desde el IUCTC se propuso comenzar el proyecto "Nessie, a news media exhaustive surveillance software", cuyo coordinador general es Alexis Quesada Arencibia, profesor del Departamento de Informática y Sistemas (DIS) de la Universidad de Las Palmas de Gran Canaria (ULPGC) y secretario del IUCTC.

El procedimiento actual que se sigue en las empresas consiste en la introducción, lectura y clasificación manual de las noticias aparecidas en diferentes periódicos por parte de un grupo de documentalistas. El objetivo global del proyecto es desarrollar un software que permita la automatización de este proceso, centrado por el momento en la prensa escrita. Con la implantación de la herramienta se persigue reducir el tiempo que se invierte en todo el proceso, transformando el rol del usuario actual a un supervisor y eventualmente un corrector manual de los errores que cometa el programa. Con ello pretendemos obtener un producto con una funcionalidad similar a las soluciones actuales pero a un coste reducido.

El primer subsistema de Nessie está planteado como Proyecto Fin de Carrera por Guanchor Ojeda Hernández, y se titula "Nessie, segmentación automática recurrente adaptativa". De este proyecto son tutores Alexis Quesada Arencibia y David Freire Obregón. El objetivo principal es la identificación de bloques de texto en cada página del periódico y su agrupación en noticias, dando como resultado un conjunto de áreas en las que se subdivide la página.

El segundo subsistema corresponde al trabajo descrito en este documento, y se encarga de obtener el texto que se encuentra en cada una de las áreas identificadas por el primer subsistema. Además irá generando cierta meta-información sobre distintos parámetros como el tamaño de letra, recuento de caracteres, etc. Aún formando parte de Nessie, este subsistema quiere plantear su diseño de forma que en la medida de lo posible pueda ser acoplado a otros proyectos como un servicio adicional y no tenga que estar ligado exclusivamente a las otras dos partes.

Por último el tercer subsistema se plantea también como Proyecto Fin de Carrera por Víctor Álvarez Hernández, y se titula "Nessie, clasificador de noticias por temas". De este proyecto son tutores Alexis Quesada Arencibia y Carlos Javier Toledo Mediavilla. Su misión consiste en controlar el proceso de suministro de periódicos, realizar una clasificación temática de cada noticia a partir de un conjunto de palabras clave predefinido, y presentar los resultados al usuario en una interfaz web.

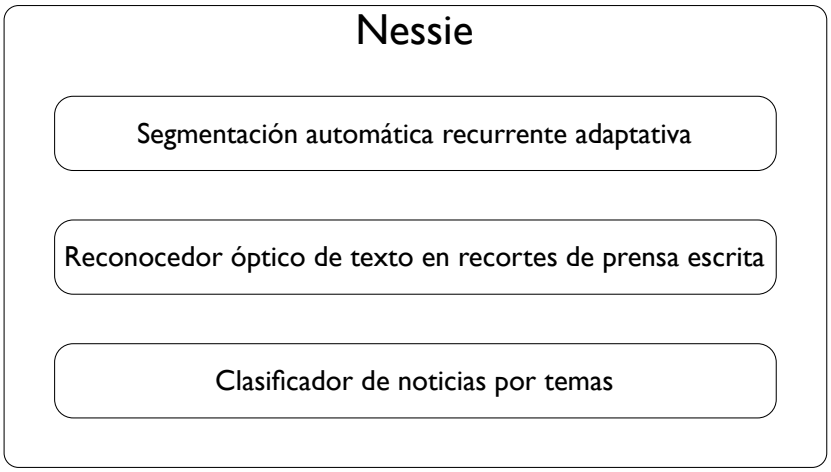


Figura 1.1: Descomposición en subsistemas del proyecto Nessie.

1.2. El seguimiento de medios de comunicación

Hoy en día los medios de comunicación juegan un papel muy importante en la sociedad, ya no sólo por dar a conocer información sobre multitud de temáticas sino por su influencia sobre la opinión pública. Muchas veces han sido criticados por ponerse al servicio de grupos empresariales o como adalides de alguna ideología política. Otras veces han servido como mecanismo de denuncia de injusticias sociales o divulgación de conocimiento científico. Lo que está claro es que se ha generado un gran interés en conocer de manera exhaustiva la información que en ellos aparece.

Pero hacer un seguimiento regular es una tarea difícil y muchas veces imposible para profesionales y organizaciones. Estar permanentemente actualizado implica dedicar recursos materiales

y humanos que, conforme aumenta el número de publicaciones, emisoras de radio y canales de televisión, puede disparar los costes. Además, la inversión puede no resultar rentable si al cabo de pocos días un porcentaje alto de la información recogida pierde valor.

Esto ha llevado al rápido crecimiento de las empresas que ofrecen un servicio de seguimiento de medios. Este servicio consiste en recopilar y clasificar toda la información que aparece en los medios de comunicación, abarcando prensa, radio, televisión y recientemente también Internet, para poder recuperar noticias de manera selectiva. Por tanto actúan como respaldo a los clientes que deseen suscribirse a un servicio de entrega personalizado, realizando por ellos el trabajo que por falta de tiempo o infraestructura les resulta imposible. Algunas de las ventajas que obtiene el cliente son [19]:

- Conocer con rapidez las noticias de las que es objeto.
- Estar al corriente de las informaciones que por alguna razón le son de interés.
- Utilizar la información como valor estratégico en beneficio propio.
- Reducir el riesgo de error en la toma de decisiones.

1.2.1. Modelos de servicio

La industria del seguimiento de medios comenzó a finales del siglo XIX, teniendo como base la lectura y distribución de las noticias publicadas en los periódicos de la época. Este servicio era usado por instituciones gubernamentales e importantes compañías. Los métodos empleados eran bastante rudimentarios: los periódicos eran leídos y todas las noticias relevantes se identificaban. Los artículos se recortaban a mano y se enviaban por correo a los clientes indicando la fuente y la fecha de edición; a menudo la cabecera del periódico también iba incluida. La evolución de la tecnología llevó años más tarde a utilizar la fotocopiadora para aumentar la velocidad de producción y reducir costes. Finalmente la era digital revolucionó completamente la industria, permitiendo que la distribución tradicional se sustituyera por el correo electrónico, sitios webs y un mejor soporte para la información publicada de forma impresa, digital y a través de medios audiovisuales.

Para realizar las tareas de seguimiento de medios a menudo es necesario contar con delegaciones en diferentes países, en las que trabajan un número considerable de profesionales (documentalistas, traductores, técnicos, etc.) condicionados por los horarios de aparición de la información en los medios. Asumir el coste de mantener un departamento similar dentro de una organización hace que muchas veces los clientes prefieran contratar los servicios de empresas especializadas.

Los modelos de servicio que integran la industria se sitúan a medio camino entre los medios de comunicación y tecnología, dado que su actividad se sustenta en ambos elementos. A grandes rasgos se pueden dividir los servicios dependiendo de la naturaleza de la fuente en la que se basen. Así, por ejemplo, existen empresas dedicadas exclusivamente al seguimiento de prensa impresa, digital, emisoras de radio o cadenas de televisión. No obstante, en la actualidad es frecuente encontrar empresas que ofrecen cobertura tanto de prensa como de medios audiovisuales. En ocasiones también resulta útil dividir las empresas según su cobertura geográfica o su especialización. Sea como fuere, los dos grandes grupos del sector pueden agruparse en dos categorías:

Press clipping: Se trata de los servicios orientados a la prensa escrita. En la actualidad el seguimiento de medios impresos se continúa haciendo intelectualmente, porque aunque se cuenta con el apoyo de instrumentos tecnológicos de búsqueda, indexación y filtrado, todavía no ha sido posible substituir el trabajo llevado a cabo por profesionales. Documentalistas expertos revisan las publicaciones, seleccionan las noticias relevantes, las digitalizan, indexan, clasifican y elaboran un dossier que se remite al cliente por correo electrónico. Éste lo recibe por lo general a primera hora de la mañana. Sin embargo las empresas líderes del sector están desarrollando métodos cada vez mejores para automatizar la parte de la digitalización, donde el documentalista quedaría relegado a una labor de supervisión.

Broadcast monitors: A diferencia de la información aparecida en publicaciones impresas, que queda registrada de forma permanente y por tanto puede consultarse con relativa facilidad, las noticias difundidas a través de medios audiovisuales plantean serios problemas de accesibilidad, puesto que sólo pueden conocerse en el momento de su emisión. La actividad de las empresas de monitorización de emisoras de radio y cadenas de televisión tiene como objetivo paliar dicha problemática. El trabajo de los profesionales en este campo no se puede automatizar tan fácilmente, y son ellos los que deben realizar el seguimiento de medios locales, nacionales o internacionales y registrar, revisar y analizar las informaciones. En una fase posterior elaborar el producto final editando las pistas de audio o las grabaciones de vídeo. Este material puede complementarse con datos relativos a la fuente de la que se tomó la información.

El futuro de la industria depende de los servicios de valor añadido que acompañen al producto estándar [7], como técnicas de análisis especializado que permitan refinar la búsqueda en la información recopilada y directorios de noticias. Estos servicios marcan tendencia en el mercado y ofrecen al cliente una perspectiva que hace que se decante por una empresa u otra.

1.2.2. Perfil de los clientes

El abanico de clientes que suscriben el servicio prestado por las empresas de seguimiento de medios es amplio, reflejando la diversidad de necesidades informativas en la actualidad. Desde las propias editoriales de los periódicos hasta gabinetes de comunicación de partidos políticos, pasando por bibliotecas y empresas, cada uno solicita el seguimiento ya no sólo de la información referente a ellos mismos, sino de la competencia, cualquier tema relacionado con el sector de interés o incluso de temas concretos sin relación alguna. Podemos clasificar los clientes dentro de los siguientes perfiles [19]:

Periodistas: En su actividad cotidiana se ven obligados a analizar y seleccionar numerosos datos y documentos, y por ello les es muy útil contar con un servicio que les proporcione la información previamente filtrada y acorde a sus intereses. Algunos requieren información puntual sobre su ámbito de especialización, otros sobre temas en los que trabajan en un momento dado. En general, les interesa conocer el punto de vista de otros medios, mantenerse informados sobre eventos, convocatorias de ruedas de prensa, etc.

Gabinetes de prensa y comunicación: La tarea de velar por una imagen pública pasa por seguir con detalle lo que prensa, radio y televisión difunden, ya que solo así será posible contrarrestar los efectos negativos de determinadas informaciones o potenciar los beneficios que puedan derivarse de otras. Los gabinetes de prensa se encuentran entre los principales

clientes de las empresas de seguimiento. Tanto si son independientes como si pertenecen a entidades públicas o privadas, su función básica es velar por la buena imagen de organizaciones o personas que sean objeto de interés.

Empresas, instituciones y organismos: Además de interesarse por las noticias de las que son objeto también consideran necesario mantenerse al corriente de otras informaciones. Por ejemplo, en el caso de empresas, aquellas referidas a competidores, mercados, productos, etc. Utilizadas como valor estratégico, estas otras noticias pueden favorecer la toma de decisiones y ayudar a alcanzar puestos de liderazgo, adelantarse a las maniobras de la competencia, percibir oportunidades de negocio o descubrir mercados emergentes, etc.

Intranets, webs corporativas y portales de Internet: Los medios de comunicación son los perfectos aliados para proveer de contenido a los sitios en Internet. Sin embargo quienes deciden enriquecer sus intranets, sedes web o portales con noticias de actualidad suelen requerir información previamente filtrada y acorde a sus intereses.

1.2.3. Productos similares existentes

El modelo comercial más común de las empresas de seguimiento de medios no consiste en entregar el programa de forma aislada, sino establecer un contrato mediante el que la compañía pone a disposición del cliente un servidor remoto con la aplicación a cambio de una cuota fija o dependiente del volumen de trabajo. Existen varias alternativas líderes en el sector a nivel comercial, de las cuales mencionaremos algunos ejemplos:

- Burrelle's Luce, creada en el año 1888 en New York City como la "Burrelle's Press Clipping Bureau". Esta empresa norteamericana es el gran clásico de la industria y está considerada como una de las mayores y más importantes del sector. Su producto estrella es el "Media Monitoring", que es capaz de recopilar noticias de prensa escrita, blogs, portales de noticias en Internet y medios audiovisuales, entre otros. El servicio de entrega de los artículos es totalmente digital y se realiza tan pronto esté disponible. Para las fuentes audiovisuales está la posibilidad de obtener el original o su transcripción.

- Zissor, creada en 2001 por Jon Asakskogen y Ove Dirbal tras obtener los derechos sobre el código fuente de productos desarrollados por una empresa anterior. "Zissor Media System" fue el primer producto que sacaron, siendo la "British Media Monitoring Agency" su primer cliente. Desde entonces tanto periódicos como bibliotecas nacionales han usado la tecnología desarrollada por Zissor para clasificar y archivar en formato digital sus documentos. Más tarde estableció acuerdos con la empresa norteamericana "Burrelle's Luce".

- Olive Software fue fundada en el año 2000. Utilizan un software diseñado por un grupo propio de ingenieros para transformar documentos en ficheros XML. El modelo de servicio que ofrecen a sus clientes consiste en la conversión de documentos en sus propias instalaciones, donde el cliente sólo tiene que proporcionar la fuente original. La publicación de los resultados puede llevarse a cabo por parte del propio cliente o a través del servicio de "hosting" de la empresa. De toda su gama de productos el que más relación guarda con la documentación de prensa escrita es "Olive ActivePaper Archive".

- TNS Global está presente en ochenta países y cuenta con una experiencia en el mercado de sesenta años, abarcando el seguimiento de medios en prensa, radio y televisión. La

sede española, con el software "TNS Media Intelligence", es capaz de procesar más de 200 diarios de información nacional, regional y económicos, 17 canales nacionales o autonómicos y sus desconexiones regionales, 14 emisoras de radio y los principales medios online.

También existen otras aplicaciones que usan la misma tecnología, aunque no están especializados en la búsqueda en prensa. Por el contrario, pretenden ser muy genéricos e intentan reconocer cualquier tipo de documento. Como ejemplo tomemos los siguientes tres proyectos:

- Ocropus es un sistema completo de análisis de documentos y reconocimiento de texto, que más que constituir un sistema cerrado permite que se le acoplen distintos "plug-in" para cada fase del proceso, como puede ser el modelado de estadísticas del lenguaje o soporte para múltiples idiomas. Tiene la ventaja de que puede reconocer también textos escritos a mano.

- Clara OCR no es un OCR tan potente como el anterior, pero permite que el propio usuario ajuste los parámetros del reconocedor para adaptarlo a sus necesidades concretas. Además está pensado para ser utilizado de manera cooperativa con otros tipos de reconocedores, e incluye una interfaz gráfica para el sistema "X Window" y una interfaz web.

- Pandora Digital Archive System fue desarrollado por la "National Library of Australia" para obtener un archivo web que permitiera la administración de todos los documentos que almacenaban. Posteriormente ha sido adaptado y comercializado para otras entidades. La finalidad no es la de ofrecer un servicio de entrega de noticias sino almacenar documentos y permitir su consulta, por lo que se asemeja más a una hemeroteca digital que a una aplicación de "press-clipping".

El proyecto Laurin

El proyecto Laurin, que estuvo en activo desde mayo de 1998 hasta agosto del 2000, merece una especial mención aparte de los productos anteriores. Fue impulsado y coordinado por el "Departamento de lengua y literatura alemana" de la "Universidad de Innsbruck". Dicho departamento posee el "Innsbrucker Zeitungsarchiv", una de las mayores colecciones de recortes de prensa relacionados con literatura y opinión en Austria, Alemania y Suiza. Otras siete colecciones presentes en Italia, España, Noruega, Suecia y Finlandia colaboran también en el proyecto, recopilando artículos relacionados con temas políticos, económicos y culturales.

La arquitectura de LAURIN se compone de un conjunto de nodos conectados a través de Internet: un nodo por cada biblioteca colaboradora y un nodo central que recopila los datos de todos los nodos locales y sirve como interfaz para el usuario final. El nodo central contiene una base de datos relacional que almacena todo tipo de *meta-datos* sobre la información de los artículos (título, autor, palabras clave,…). Los nodos locales se encargan del proceso de press-clipping — escaneado, el análisis de composición, el reconocimiento de caracteres, indexado del contenido, etc.— pero sólo envían al nodo central toda la información del artículo bajo petición de un usuario.

Una característica interesante en LAURIN es que los recortes de prensa obtenidos en el flujo de trabajo están conectados con el "LAURIN Thesaurus" [1], que se almacena tanto en los nodos

[1] Un Thesaurus es un tipo de diccionario que contiene una lista de términos, en la que cada uno está acompañado de otros términos relacionados con él, un listado de sinónimos y antónimos, etc.

locales como en el central. Hay un envío constante de información desde los nodos locales hacia el nodo central, que se actualiza con nuevas entradas pendientes de revisar por el encargado del Thesaurus. Las entradas se relacionan por conceptos, que poseen una clave única común para aglutinar artículos de diferentes idiomas.

1.2.4. Regulación de la industria

Dada la importancia que fue adquiriendo el sector, han ido surgiendo diversas asociaciones que agrupan a las empresas de seguimiento y llevan a cabo actuaciones para la regulación del sector. De todas ellas cabe destacar la "Federation Internationale des Bureaux d'Extraits de Presse" (FIBEP) [7], que con más de medio siglo de historia agrupa a 90 miembros en más de 40 países. Actualmente representa el 80 % de las empresas de seguimiento de medios y realizan congresos cada 18 meses en distintas sedes del mundo. Esta asociación tiene entre sus objetivos mejorar las relaciones entre sus miembros, incluyendo el intercambio de conocimiento, valores y comportamiento ético. Al mismo tiempo pretende ser un foro donde todos los sectores de la industria puedan manifestar su visión. Otra organización importante en la misma línea es "The Internacional Association of Broadcast Monitors" (IABM) [16], que además pretende crear estándares de calidad para el sector, unificar las tendencias tecnológicas y salvaguardar el derecho público de acceso a la información.

Sin embargo con la internacionalización surgen problemas adicionales, dada la falta de armonización en la legislación relativa a propiedad intelectual. En Suecia, por ejemplo, no se permite enviar noticias por fax sino físicamente. En Alemania se admite el envío por fax pero no la distribución electrónica, ni tampoco hacer más de siete copias del mismo documento. En el Reino Unido, Francia o Suiza es obligatorio pagar un canon por cada página de periódico copiada, mientras que en España o Italia no existe este requisito [19].

La regulación de las actividades de la industria de seguimiento de medios, promovidas por las asociaciones comentadas, ha dado lugar a la formulación de códigos de conducta ética que intentan favorecer la actuación objetiva, neutral e independiente de las empresas del sector. Por ejemplo, enfatizan mucho la necesidad de independencia que deben tener las compañías de los medios que controlan, la fidelidad del contenido recopilado y el respeto a las leyes vigentes de propiedad intelectual u otros derechos. En el caso de medios audiovisuales también se propone que se omitan de la monitorización los programas de entretenimiento. Así se intenta compensar las diferencias en materia legislativa.

1.3. Funcionamiento de la prensa escrita

De todos los medios de comunicación actuales el periódico es uno de los formatos más importantes. Aparte de ser el de más temprana aparición, es el único que conserva un índice de volatilidad relativamente bajo con respecto a los demás. La información queda registrada de manera permanente y con facilidad en los archivos de cada editorial o en las hemerotecas. Actualmente también puede exportarse a otros formatos digitales como páginas web o archivos PDF.

Como vimos en la sección anterior el "press clipping" constituye toda una industria en sí mismo, y es importante conocer de qué manera se trabaja en un periódico de cara a abordar los objetivos de este proyecto. No vamos a entrar en detalle sobre cada aspecto del proceso de producción que requiere la publicación de un periódico, sino que nos centraremos en las partes que influyen a este proyecto: la edición de noticias y el diseño de las páginas.

1.3.1. El proceso de edición

Normalmente un periódico de información general está estructurado en secciones que engloban noticias de contenido común. Estas secciones suelen ser bastante genéricas y por ello internamente se dividen en temas. Por ejemplo, dentro de la sección "deportes" podemos encontrar temas como "fútbol", "baloncesto", "tenis", etc. Además en Canarias, por sus características territoriales, los periódicos suelen incluir secciones específicas de cada región. Sin embargo a veces es complicado decidir la colocación de una noticia, sobre todo si su contenido está claramente relacionado con dos temas.

Estrictamente hablando, el proceso de edición de un periódico se origina en la persona del director, que marca la línea editorial e incluso el enfoque que debe darse a la información. Por este motivo es muy habitual encontrarnos con periódicos asociados a una ideología concreta. Por debajo del director se encuentra el jefe de sección, que selecciona las noticias que van a ser publicadas y les asigna un espacio en página según su importancia. La información del espacio es utilizada por el diseñador gráfico para decidir la maquetación de cada página del periódico, generando una serie de plantillas que usará el redactor para escribir la noticia.

1.3.2. Pautas de diseño

Aunque la mayoría de los periódicos convencionales dividen cada página en cinco columnas, últimamente los periódicos están modificando los diseños centrándose en captar la atención del lector y facilitar la lectura de las noticias. De hecho, podemos encontrar que a veces las columnas no tienen ancho fijo o que se amoldan al contorno de las imágenes presentes en la página. En cualquier caso el redactor no tiene por qué preocuparse de esto, ya que todo viene predefinido en la plantilla que recibe del diseñador y sólo debe ajustar el tamaño de su noticia a las columnas que se le han asignado.

Sin embargo no es del todo cierto que la asignación de espacio para una noticia está supeditada a la importancia de la misma. La mayor parte de los ingresos de un periódico depende de la publicidad que aparece en las páginas, y es ésta la que marca el espacio restante que queda para la información. Por lo tanto el diseñador debe tener en cuenta el siguiente orden:

1. Reserva de páginas para cada sección.
2. Reserva de espacio para la publicidad.
3. Asignación de espacio para cada redactor/noticia dentro de la sección.

El contenido mínimo de una noticia lo constituye el título, la *data* (autor y localización) y el cuerpo del texto informativo. Junto a estos tres elementos básicos podemos encontrarnos otros muchos, que se utilizan para captar la atención del lector o añadir información al cuerpo principal. Aunque no son los únicos enumeramos a continuación alguno de los más importantes:

- Los *cintillos* son encabezados que se colocan normalmente en la parte superior de la página, indicando la sección a la que pertenece la noticia.
- Los *antetítulos y subtítulos* son pequeños fragmentos de texto situados en la parte superior e inferior del título respectivamente. Normalmente amplían el texto del título para formar en conjunto el titular de la noticia. Es bastante frecuente encontrarlo sobre todo en noticias cuyo título aparece en una tipografía de gran tamaño.

- Cuando se desea resaltar algunas partes de la noticia se hace uso de los *sumarios*, que no son más que extractos literales del cuerpo de la noticia incrustados en alguna de las columnas con una tipografía de tamaño mayor que la del cuerpo. Suele ser habitual para destacar declaraciones de alguna persona.
- Los *despieces* se utilizan para añadir información complementaria o relacionada con una noticia adyacente, pensado para que no interrumpa la continuidad de la lectura del artículo principal. La realidad es que muchas veces los despieces se encuentran incrustados en medio de dos columnas, acompañados de un pequeño encabezado y con la tipografía en cursiva para distinguirla del cuerpo del texto.

Por último nos encontramos con la separación entre noticias. A menudo se utilizan líneas verticales, especialmente cuando no existe relación de contenido entre ellas a pesar de estar dentro de la misma sección. También podemos encontrar líneas horizontales, aunque no es tan habitual porque normalmente el titular es suficiente como elemento separador. La separación de las columnas del cuerpo del texto se realiza con una banda de espacio en blanco, de un ancho significativo para conseguir el objetivo sin independizar visualmente una columna de otra.

1.3.3. Tipografías

Stanley Morison, creador de la fuente Times, definió la tipografía como el "arte de disponer correctamente el material de imprimir, de acuerdo con un propósito específico: el de colocar las letras, repartir el espacio y organizar los tipos con vistas a prestar al lector la máxima ayuda para la comprensión del texto" [28]. En el contexto actual esta definición de tipografía debe adaptarse a la realidad del proceso editorial: todos los periódicos se editan digitalmente a través de un software. Por tanto el diseñador sólo tiene que elegir la fuente, el estilo y el tamaño en cada situación, lo que se conoce como *elección de tipo*.

Tradicionalmente se ha jugado con los tamaños de letra en los titulares para identificar visualmente la importancia de las noticias, de forma que titulares con una fuente mayor implican una importancia mayor. El tamaño también se usa para destacar elementos del cuerpo del texto, como los sumarios o despieces. En contraste, el tamaño de la letra que forma el cuerpo del texto es uniforme para todas las noticias y es un parámetro estrechamente ligado a la legibilidad de la fuente.

El estilo de la fuente empleada es un recurso orientado sobre todo a destacar fragmentos de texto sobre otros en párrafos con letra de un mismo tamaño. Por ejemplo, se usa el estilo *bastardilla* cuando se quiere hacer énfasis en un término, distinguir el texto de un despiece en la noticia adyacente o citar palabras textuales de una declaración. Un uso similar, aunque más orientado a los encabezados, se hace del estilo *negrita* para igualmente resaltar términos o el autor y la localización de la noticia. Anteriormente también se utilizaba el estilo subrayado para destacar alguna palabra, pero ha quedado obsoleto.

La elección del estilo y el tamaño es la parte más sencilla del diseño de la tipografía de un periódico, ya que cumple unas funciones más específicas relacionadas con la importancia del contenido. Sin embargo a la hora de elegir una fuente hay que ser mucho más cuidadoso. Salvando los elementos circundantes (titular, sumarios,…), el texto principal de un artículo suele ser el más largo y en el que más tiempo invierte el lector. La facilidad que aporte una fuente para su lectura es un parámetro de vital importancia que debe tenerse en cuenta.

Una forma de clasificar las letras consiste en evaluar si tienen "serif" o no. Se conoce por "serif" (en español puede traducirse como gracia o remate) a las pequeñas líneas que se encuentran

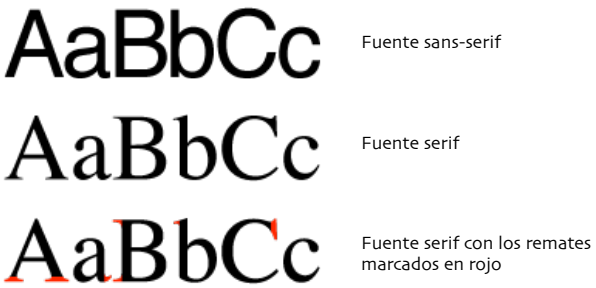


Figura 1.2: Diferencia entre una fuente "serif" y una "sans serif". Los remates crean el efecto visual de continuidad y facilitan la lectura.

en las terminaciones de las letras, principalmente en los trazos verticales o diagonales. Por el contrario las letras "sans serif" (conocidas como de palo seco) son aquellas que no llevan ningún tipo de terminación. En la figura 1.2 puede verse la diferencia entre ambos tipos de letra.

Las letras "serif" (llamadas también romanas) facilitan la lectura, ya que crean en el ojo la ilusión de una línea horizontal por la que se desplaza la vista al leer. Las "sans serif" (o de palo seco) son consideradas inadecuadas para un texto largo, ya que la lectura resulta incómoda al existir una tendencia visual a identificar este tipo de letras como una sucesión de palos verticales consecutivos. Por esta razón, las letras con "serif" se utilizan en los periódicos, revistas y libros, así como en publicaciones que contienen textos extensos, mientras que las otras son usadas en titulares, rótulos, anuncios y publicaciones con textos cortos.

1.4. La tecnología aplicada al seguimiento de medios

La aplicación del reconocimiento óptico de caracteres al seguimiento de medios no es una idea novedosa, al menos conceptualmente. A lo largo de la historia de la industria se han realizado aplicaciones muy específicas orientadas a obtener información que al principio estaban centradas en un solo tipo de periódico. El primer sistema comercial del que se tiene constancia fue instalado en el año 1955 para la revista estadounidense "Reader's Digest", que años más tarde lo cedió a la "Smithsonian Institution".

Con el paso de los años mejoró la tecnología software y hardware, el horizonte fue expandiéndose y comenzó a pensarse en aumentar la potencia de la herramienta, de manera que se pudieran tratar periódicos de distintas editoriales. Evidentemente este objetivo no tiene que ver únicamente con las prestaciones que ofrezca el reconocedor de texto, sino también con la capacidad de detectar las diferentes maquetaciones que aplica cada imprenta, lo que se conoce como *document image analysis*.

La detección de caracteres en fuentes tipográficas se considera un problema resuelto en la mayoría de sus aspectos, mientras que el reconocimiento de caracteres manuscritos presenta una mayor problemática. Aunque en ámbitos académicos se siguen realizando investigaciones para desarrollar nuevos métodos o plantear mejoras a los ya existentes, la realidad es que la industria se ha centrado más en la implementación de las técnicas cuyo éxito está contrastado, haciendo hincapié en aspectos como la rapidez en la consecución del resultado o el número de palabras reconocidas correctamente.

1.4.1. Ciclo de trabajo básico

Los reconocedores de texto actúan normalmente como un módulo dentro del proceso de análisis de documentos. En el caso más simple un analizador de documentos recibe una página en formato digital (JPEG, PNG, TIFF,...) y comienza a detectar fronteras que presumiblemente delimiten el contenido de una noticia. Cuando se estima que una zona de la página contiene un bloque de texto uniforme se recorta y se envía al reconocedor de texto. Éste realiza su labor identificando los caracteres y parámetros referentes al estilo empleado por la editorial, como el tamaño de la fuente o la forma. Al terminar devuelve al analizador el resultado para continuar con otras noticias. En la figura 1.3 tenemos una representación de un sistema como el descrito con todos los módulos conectados de manera secuencial.

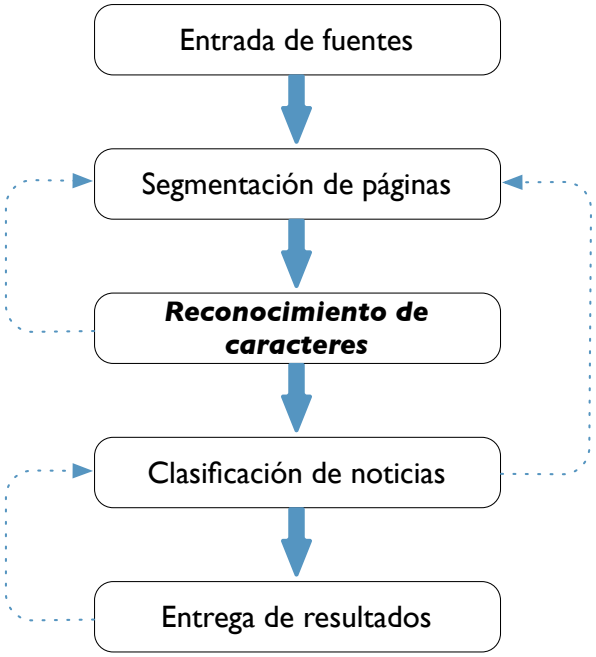


Figura 1.3: Etapas por las que atraviesa un sistema de seguimiento de medios automatizado. Las líneas discontinuas muestran los caminos de realimentación que permiten refinar los resultados.

La tecnología actual permite utilizar escáneres de gran resolución y velocidad para obtener las imágenes cuanto más pronto posible. Esto es crítico ya que en el mundo de la computación el cuello de botella generado por la entrada/salida ralentiza la ejecución de los programas y condiciona las mejoras que se hagan en otros aspectos. Por su parte, el analizador y el OCR pueden programarse para utilizar computación distribuida o paralela. Si se es capaz de repartir la carga de trabajo entre varios procesadores el número de páginas por unidad de tiempo que se analizan puede disminuir considerablemente. La compañía "Olive Software" que citamos en la sección 1.2.3 utiliza esta técnica en el "Online Microfilm Institute". En sus instalaciones disponen de la arquitectura "PipeX", que puede usar hasta 96 procesadores para repartir la carga y procesar páginas o incluso periódicos en paralelo.

Al ciclo de trabajo básico descrito anteriormente se le pueden añadir ciertas mejoras. Por ejemplo, el analizador puede hacer uso del contenido semántico del texto y de la posición de los bloques de texto dentro de la página, proporcionados por el OCR, para decidir cuáles per-

tenecen a la misma noticia. Si dos bloques adyacentes comparten un cierto número de palabras clave es posible que haya que unir ambos bloques y formar un único bloque. También se podría identificar qué bloques de texto son de algún tipo especial atendiendo a las propiedades detectadas por el reconocedor. Si un estilo es de un tamaño y una forma concreta se puede asumir que es un titular, un pie de foto o la firma del periodista. Uniendo ambas mejoras se podría llegar a asociar el titular a la noticia.

En una fase posterior todas las noticias obtenidas pasarían a un módulo de clasificación, que aplicando mecanismos de búsqueda de palabras clave o basándose en ontologías pueden categorizar las noticias. Desde una perspectiva más comercial se pueden seleccionar noticias concretas si se detectan ciertos patrones que concuerden con los parámetros de un cliente en particular. El resultado final se somete a la revisión del documentalista, que puede reorganizar los bloques asociados a una noticia si el proceso automático lo hizo de manera errónea.

2 Modelo de análisis

Hasta ahora sabemos que NessieOcr debe funcionar como un subsistema dentro de Nessie, prestando una serie de servicios a un conjunto de clientes. Estos clientes no son personas físicas, sino que consideramos un cliente como una entidad que hace uso de las facilidades de la biblioteca. Es importante tener claro qué tareas debe desempeñar Nessie para satisfacer las necesidades de tales clientes.

En este capítulo presentamos el análisis de requisitos de NessieOcr, de cara a formalizar tres aspectos: quién va a usar el software, qué requisitos necesita satisfacer y qué elementos intervienen en dichos requisitos. Como resultado obtenemos el *modelo de dominio*, descrito en la sección 2.1, y el *modelo de casos de uso*, descrito en la sección 2.4.

2.1. Modelo de dominio

El software de seguimiento de medios que propone el proyecto Nessie se compone de cuatro subsistemas principales: un cargador de periódicos, un segmentador de páginas, un reconocedor de texto y un clasificador de noticias (Figura 2.1). El modelo de dominio que describimos a continuación resume las relaciones entre los subsistemas y los elementos de los que es responsable cada uno:

- El cargador lee un periódico desde un soporte externo y crea una lista ordenada de páginas.
- El segmentador toma cada página y analiza su estructura, extrayendo un conjunto de recortes.
- El reconocedor procesa cada recorte y genera un texto, a partir de los caracteres que ha encontrado.
- El clasificador recibe todos los textos asociados a una noticia y construye un artículo.
- La gestión de todo el sistema la realiza un controlador, suministrando los periódicos al comienzo y enviando los artículos a su destino al final.

El dominio de NessieOcr (Figura 2.2) es bastante limitado. El controlador de la aplicación le entrega un recorte de prensa que ha generado el segmentador a partir de la página de un periódico, el reconocedor lo procesa, extrae los caracteres y construye un texto. Dicho texto se pasará

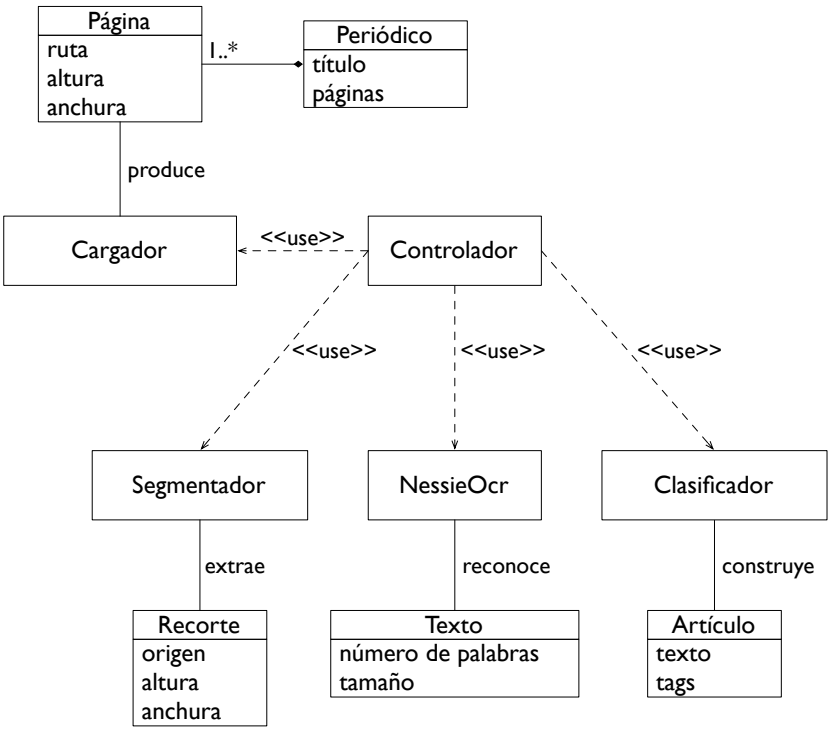


Figura 2.1: Modelo de dominio de Nessie.

como dato de entrada al clasificador. Como información adicional hay que generar una serie de resultados estadísticos para evaluar a posteriori el funcionamiento interno del subsistema.

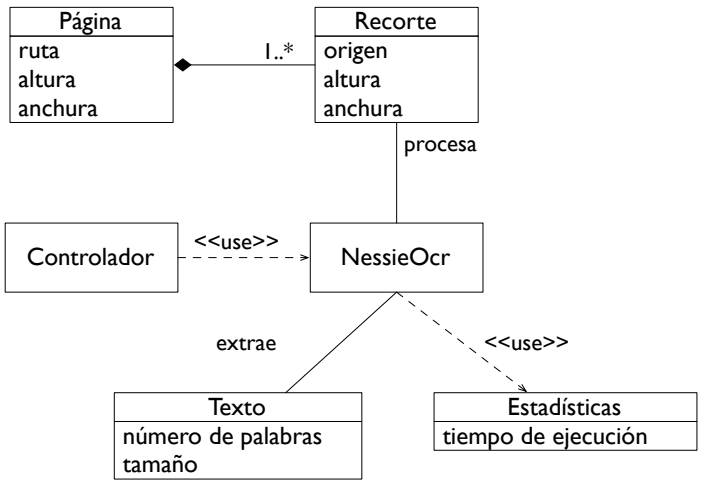


Figura 2.2: Modelo de dominio de NessieOcr.

2.2. Requisitos del producto

Uno de los requisitos impuestos por el proyecto Nessie está relacionado con el *tiempo de ejecución*. Todo el sistema debe procesar al menos una página por minuto para que la implantación del producto sea rentable. Este tiempo debe repartirse entre la carga de un periódico en la aplicación, la segmentación de cada página en recortes, la extracción del texto de cada recorte y la construcción de un artículo clasificado con una serie de etiquetas.

Puesto que la unidad de medida de este requisito es el número de páginas por minuto, podemos dejar fuera del cálculo la carga de periódicos. Esto nos deja con que el minuto de tiempo por página debe repartirse entre las tres tareas principales: segmentación, reconocimiento y clasificación. Por tanto, si realizamos un reparto equitativo NessieOcr sólo dispone de 20 segundos como máximo para realizar su trabajo. La ecuación (2.2.1) resume esta restricción, donde t_i representa el tiempo invertido en el recorte i-ésimo y R el número total de recortes de una página.

$$\sum_{i=1}^{R} t_i \leq 20s \tag{2.2.1}$$

El tiempo que se invierte en cada recorte es difícil de calcular, ya que no todos tienen las mismas dimensiones ni el mismo número de caracteres. La siguiente expresión es una aproximación que nos debe permitir establecer el tiempo máximo por recorte:

$$\textit{Tiempo medio por recorte} \leq \frac{20s}{\textit{Número medio de recortes por página}} \tag{2.2.2}$$

Esta expresión no es del todo realista, ya que si una página contiene sólo dos recortes, uno de ellos ocupando casi toda la página y el otro muy pequeño en relación a aquel, el tiempo medio por recorte será diez segundos. Este tiempo verifica efectivamente la ecuación anterior, pero es obvio que procesar el recorte mayor llevará más tiempo que procesar el recorte menor. Por tanto, la expresión nos vale para casos más frecuentes, en los que la distribución de una página contiene muchos más recortes.

El segundo requisito del producto está relacionado con la *portabilidad*. Debe garantizarse que el código fuente es compatible con el estándar POSIX, y totalmente portable como mínimo entre sistemas Linux y Mac OS X. Al mismo tiempo es deseable que siendo portable para Linux también lo sea para el resto de sistemas operativos derivados de UNIX.

2.3. Sistemas de reconocimiento de patrones

Un sistema de reconocimiento de patrones tiene como objetivo clasificar un conjunto de muestras obtenido de una señal de entrada en un conjunto de clases, donde cada clase representa un tipo de objeto distinto perteneciente a un dominio conocido. Estos sistemas normalmente tienen tres etapas fundamentales: preprocesamiento, extracción de características y clasificación.

Un reconocedor óptico de caracteres no es más que una implementación concreta de un sistema de este tipo. Las muestras son conjuntos de pixels pertenecientes a una imagen digital, y las clases son los distintos tipos de caracteres de un alfabeto dado. Por tanto, para definir la funcionalidad de NessieOcr es necesario entender este modelo de referencia.

A lo largo de esta sección vamos a describir la arquitectura general de un sistema de reconocimiento de patrones, que nos permitirá construir con precisión el modelo de casos de uso de NessieOcr.

2.3.1. Arquitectura general

Supongamos que una empresa de venta de marisco nos pide que automaticemos el proceso de selección de las capturas, separándolas en dos clases: moluscos de tipo A y moluscos de tipo B. Para ello colocamos una cámara sobre una cinta transportadora que trae los ejemplares y observamos las diferencias visuales entre ellos. A simple vista parece que el perímetro de la concha es una buena característica para distinguir un molusco de otro, por lo que desarrollamos un sistema de reconocimiento que captura imágenes de la cinta y extrae este dato de cada ejemplar. En base a los valores obtenidos construimos un modelo que clasifica cada molusco en su categoría, estableciendo una frontera de decisión.

Cuando ponemos en marcha la aplicación nos damos cuenta que hay ejemplares de moluscos A cuyo perímetro de concha se acerca bastante al de los moluscos B. A pesar de que en ejemplares adultos la clasificación se realiza correctamente, no existe una frontera clara cuando los ejemplares son crías. La conclusión es evidente: el perímetro de la concha no es una característica válida por sí sola. Necesitamos características adicionales para redefinir el modelo y comprobar los resultados de una nueva clasificación.

Este hipotético sistema de clasificación de moluscos contiene varias etapas que definen la arquitectura del sistema, y que se representa de forma esquemática en la figura 2.3. Entender el funcionamiento de cada etapa nos ayuda a diseñar el sistema completo.

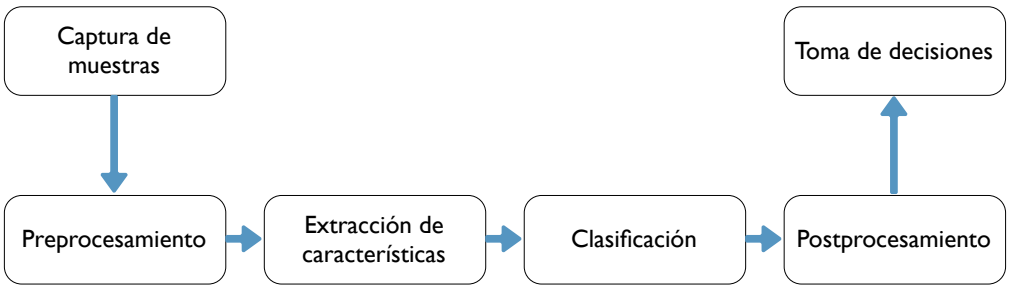


Figura 2.3: Arquitectura de un sistema genérico de reconocimiento de patrones.

Captura de muestras

La entrada a un sistema de reconocimiento es normalmente una especie de transductor, como una cámara, un escáner o un micrófono. El dispositivo convierte la señal de entrada (intensidad de la luz, cambio de presión, diferencia de voltaje,…) en una señal de datos digitalizada. La dificultad en este caso reside en superar las limitaciones de calidad del transductor — un ancho de banda bajo, una tasa de señal-ruido inaceptable, etc.

Preprocesamiento

En el ejemplo de los moluscos no tuvimos en cuenta que en cada captura podía aparecer más de un ejemplar. Además asumimos que podíamos distinguir perfectamente cada muestra de la cinta transportadora. Pero en la práctica es necesario aplicar una serie de transformaciones a la señal de entrada para poder aislar las muestras del fondo, del ruido o de otras muestras.

El preprocesamiento es una de las fases más importantes, ya que su eficacia nos permite asumir que las muestras que estamos procesando corresponden efectivamente a los objetos que

queremos clasificar. En caso contrario estaríamos buscando características de un objeto en una muestra de algo que no le corresponde. Las responsabilidades del preprocesamiento son diversas, pero quizás las más importantes son:

- Mejorar la calidad de la señal de entrada.
- Aislar las muestras de interés del resto de elementos.
- Eliminar la información redundante de cada muestra.
- Generar una lista de patrones válida para las siguientes etapas.

En el caso de las imágenes, la señal de entrada es una matriz cuyas celdas definen el valor de los pixels en algún espacio de color. Muchas veces se intenta evitar un preprocesamiento complejo invirtiendo en hardware de adquisición de alta resolución, pero si no es posible los algoritmos de este etapa ayudan a mejorar la imagen. Los métodos más comunes son:

- Transformaciones en los niveles de intensidad.
- Transformaciones geométricas.
- Análisis de vecindades.
- Restauración haciendo uso de información de la imagen completa.
- Localización de objetos mediante segmentación de bordes o regiones.

Extracción de características

Mediante la extracción de características seleccionamos el conjunto de propiedades que nos permite caracterizar los objetos que queremos clasificar. Al evaluar este conjunto para una muestra perteneciente a una categoría, los valores resultantes deben ser muy similares a los de otras muestras de la misma categoría y lo suficientemente alejados de los de otra.

Es importante procurar que las propiedades elegidas sean invariantes a transformaciones en la señal de entrada. Por ejemplo, si usamos imágenes como señal de entrada las propiedades no deben verse afectadas por rotaciones, translaciones o cambios de tamaño (una cría de mejillón sigue siendo un mejillón, no un berberecho).

Clasificación

El trabajo de un clasificador consiste en evaluar el conjunto de características de cada patrón para asignar la muestra de entrada a una clase. Cuanto más homogénea sea la información a procesar más difícil será asignar la clase a la que pertenece cada muestra. Sin embargo, tampoco es aconsejable ser demasiado estrictos al establecer la frontera entre clases, sino dejar un grado de libertad tal que el clasificador trabaje de una manera más general en base a probabilidades.

Para medir la calidad del clasificador se pueden utilizar diversas métricas. Un parámetro puede ser la acumulación del error cometido para obtener a posteriori la probabilidad de error total del sistema. Otra medida considera el costo que supone hacer una clasificación incorrecta.

Postprocesamiento

Por lo general un sistema de reconocimiento de patrones no existe por sí sólo, sino que se utiliza como base para tomar una acción concreta. En el ejemplo de los moluscos podría ser dividirlos en cestas diferentes para optimizar el proceso de empaquetado. Es en la etapa de post-procesado en la que se consideran los aspectos colaterales a la clasificación para conjuntamente tomar una decisión.

2.3.2. Clasificación de patrones

La etapa de clasificación de un reconocedor de patrones es el núcleo del sistema, asumiendo que somos capaces de localizar y extraer los patrones que representan los objetos de interés. El diseño de un clasificador conlleva normalmente la repetición de una serie de actividades. En esta sección vamos a describir con un poco más de detalle este ciclo de diseño, representado en la figura 2.4.

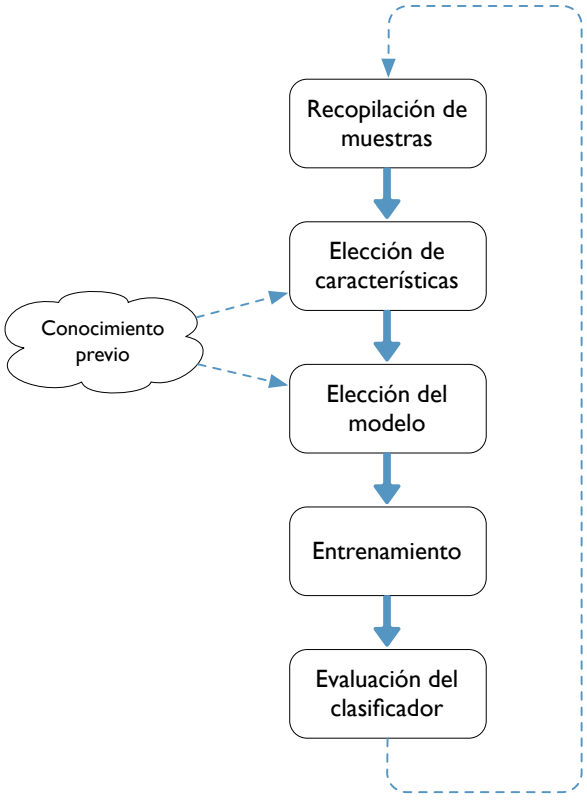


Figura 2.4: Ciclo de diseño de un clasificador de patrones. En cada iteración se pueden utilizar los resultados de la evaluación para modificar los parámetros de etapas anteriores, de forma que la clasificación sea cada vez más precisa.

Recopilación de muestras

Recopilar y seleccionar un conjunto de muestras representativo de los objetos que queremos clasificar no es una tarea compleja, pero puede llevarnos bastante tiempo. Cabe la posibilidad

de plantearnos realizar un estudio de viabilidad con un pequeño conjunto de ejemplos típicos, pero cuanto más ejemplares tengamos mejor rendimiento obtendremos en el entrenamiento del clasificador. De hecho, para poder aportar un grado importante de generalidad normalmente es necesario disponer de un grupo lo suficientemente grande y representativo de todas las clases.

Elección de características

La elección de un conjunto de características apropiado para distinguir ejemplares de diferentes categorías es uno de los pasos críticos en el ciclo de diseño. En la mayoría de los casos esta elección depende del conocimiento que hayamos obtenido tras realizar una observación empírica en el dominio del problema.

Sin embargo, a veces el conocimiento práctico debe ser respaldado por un conocimiento a priori. En el ejemplo de los moluscos elegimos el perímetro de la concha como propiedad discriminatoria porque al observar las imágenes capturadas por la cámara parecía que las diferencias eran bastante claras. Pero disponer de un conocimiento a priori de otras características de ambas especies hubiera ayudado a elegir otras propiedades más adecuadas, como el color de la concha o la geometría de su silueta.

La mejor opción es combinar ambas aproximaciones—obtener conocimiento a priori del dominio del problema y examinar un conjunto representativo de muestras—, para que al final las propiedades a seleccionar sean fáciles de extraer, invariantes a transformaciones, insensibles al ruido y útiles para distinguir unas clases de otras.

Elección del modelo

Las diferencias entre clases pueden ajustarse a diferentes modelos o descripciones, expresadas normalmente de manera matemática. El objetivo en la clasificación de patrones consiste en plantear hipótesis acerca de la clase de cada modelo, y observando los datos de entrada tras eliminar el ruido elegir qué modelo se ajusta mejor a cada categoría.

Es posible que tras una primera iteración del ciclo de diseño, el modelo elegido no se ajuste a las expectativas y pensemos en cambiar a otro. ¿Cómo sabemos cuando cambiar de modelo? ¿Cómo podemos estar seguros de qué modelo se ajusta mejor a nuestro vector de características? En la ejecución de esta etapa hay que ser especialmente cuidadosos, porque de lo contrario estaríamos abocados a entrar en una dinámica de prueba-error sin saber exactamente si el esfuerzo invertido se va a convertir en una mejora del rendimiento. Por suerte, hay una serie de principios que nos sirven de guía para saber en qué contextos inclinarnos por una clase de modelos con respecto a otra.

Entrenamiento del clasificador

Entrenar un clasificador implica, de manera general, usar un conjunto de muestras para determinar el rendimiento y la calidad en la decisión. No existe ningún método universal que permita diseñar un sistema de reconocimiento de patrones para cualquier dominio. Sin embargo, la experiencia adquirida a lo largo de las últimas décadas ha demostrado que el entrenamiento en base a muestras de ejemplo constituye una de las mejores herramientas.

Evaluación del clasificador

Esta etapa del diseño nos permite establecer un mecanismo por el que estimamos la calidad del clasificador. Básicamente el objetivo consiste en evaluar la tasa de error del sistema según los datos de entrada y en caso de no ser adecuada volver a revisar alguna de las etapas del ciclo. La evaluación del clasificador también permite obtener una medida del rendimiento del sistema, de forma que podamos plantear mejoras en alguno de los componentes.

Durante esta etapa es importante llegar a un compromiso entre eficiencia y calidad. Tal vez podamos tener un sistema que es muy poco eficiente en la fase de entrenamiento, pero que una vez se encuentre en proceso de producción la fiabilidad sea excelente. Por el contrario, es posible que en algunos entornos sea necesario admitir un clasificador un poco peor si el sistema debe cumplir ciertas restricciones de tiempo. En este caso habría que compensar los errores cometidos con algún tipo de postprocesamiento. En el caso de los reconocedores de texto, por ejemplo, es habitual hacer uso de diccionarios cuando una palabra no ha podido ser leída en su totalidad.

2.4. Modelo de casos de uso

El reconocedor de texto debe ofrecer un servicio que está perfectamente definido: procesar una imagen en un formato digital conocido para extraer el texto que contiene. También es fundamental que NessieOcr pueda ser entrenado para mejorar su tasa de acierto a la hora de reconocer caracteres. Además, debe ser capaz de generar una serie de resultados asociados a cada acción de reconocimiento, para disponer de información sobre cómo se ha llevado a cabo el reconocimiento. Por ejemplo, es muy interesante conocer los tiempos de ejecución o la tasa de acierto.

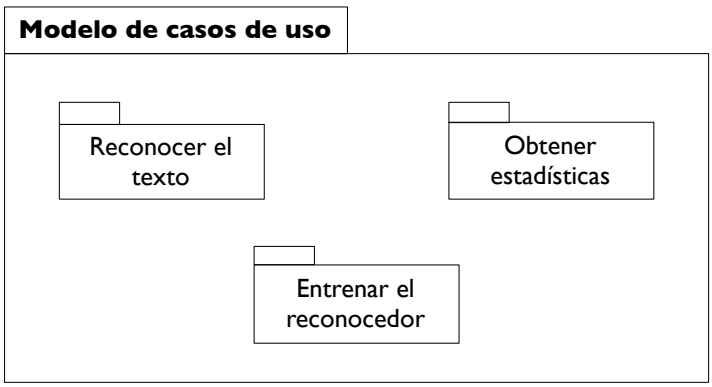


Figura 2.5: Modelo de casos de uso de NessieOcr.

El modelo de casos de uso (Figura 2.5) que describimos a continuación identifica los actores que intervienen (clientes) y las acciones que quieren llevar a cabo (casos de uso). Cada caso de uso se traducirá en la definición de los *servicios* de NessieOcr, y se describe a través de una explicación en lenguaje natural, un diagrama y una ficha con su flujo de sucesos en detalle. Las fichas pueden consultarse al final de este capítulo, en la sección 2.6.

2.4.1. Reconocer el texto

Reconocer el texto que se encuentra en un recorte de prensa es la función más importante del reconocedor. Para modelarla hemos decidido tomar como referencia el sistema de reconocimiento de patrones de la sección anterior, en el que los patrones representan los caracteres que componen el texto.

Para llevar a cabo el reconocimiento, NessieOcr necesita una imagen de entrada que representa el recorte de prensa. Esta imagen es sometida primero a una etapa de preprocesamiento, que se encarga de mejorar la calidad de la imagen, eliminar la información innecesaria o redundante y aislar las regiones de pixels que representan los caracteres del resto de elementos. Esto se consigue aplicando diferentes algoritmos de procesamiento de imágenes. A continuación es necesario identificar para cada región un conjunto de características que ayuden a distinguirla de las demás. El capítulo 3 está dedicado a analizar los métodos más apropiados para llevar a cabo estas acciones.

El conjunto de características obtenido se clasifica en base a algún criterio para obtener el carácter asociado a la región. En el capítulo 4 se describen los diferentes paradigmas de clasificación válidos para reconocer caracteres. El último paso consiste en construir el texto completo usando los caracteres reconocidos y someterlo a un postprocesamiento, según los requerimientos del subsistema que vaya a hacer uso de él. En este caso, el subsistema que usa el texto es el clasificador de noticias, que necesita únicamente las palabras, sin signos de puntuación u otros caracteres.

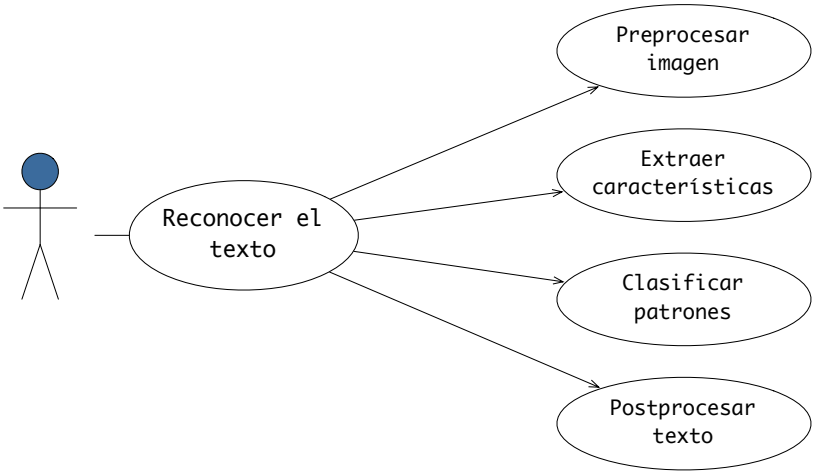


Figura 2.6: Caso de uso "Reconocer el texto".

El rendimiento de esta tarea es muy importante, ya que influye en el objetivo global de la aplicación de conseguir procesar una página en un minuto como máximo. Por tanto, los esfuerzos de optimización deben concentrarse en conseguir satisfacer la ecuación 2.2.2.

2.4.2. Entrenar el reconocedor

La clasificación de patrones de NessieOcr es un proceso que se entrena para mejorar los resultados obtenidos, especialmente la tasa de acierto. Esto no es sólo necesario en el momento inicial del lanzamiento del producto, sino que durante la vida del software puede ser necesario

enriquecer dicho comportamiento. Por ejemplo, un nuevo cliente puede solicitar incluir periódicos distintos a los que se pensó originalmente y en los que la fuente puede variar.

Para entrenar un reconocedor es necesario recopilar un conjunto de muestras de aprendizaje que correspondan a los nuevos patrones. Este conjunto debe pasar por todas las etapas de un reconocimiento normal, excepto por la de postprocesamiento. Por cada patrón reconocido, se compara el carácter detectado por el carácter que se esperaba. Si ambos valores coinciden no es necesario ningún ajuste. En caso contrario debe modificarse los parámetros del clasificador para corregir el error. Este proceso se conoce como aprendizaje supervisado.

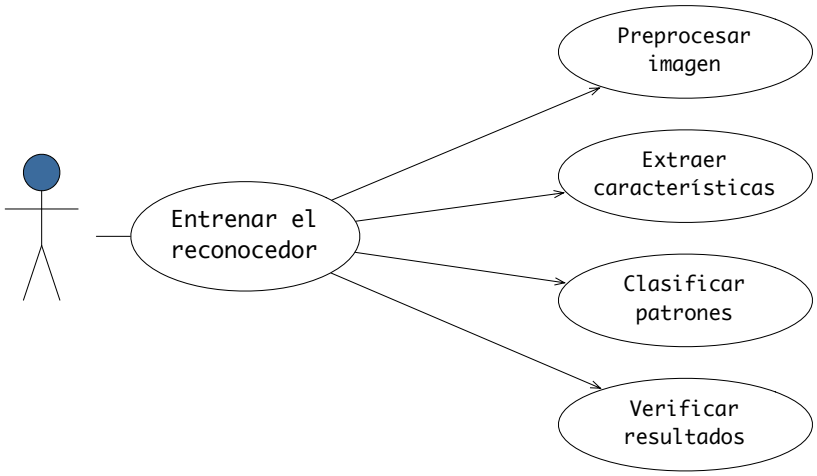


Figura 2.7: Caso de uso "Entrenar el reconocedor".

La importancia de esta tarea en cuanto a tiempo de ejecución es relativamente baja, ya que el entrenamiento es un procedimiento que se ejecuta de manera independiente al funcionamiento normal de NessieOcr en la aplicación global.

2.4.3. Obtener estadísticas

A partir de un texto reconocido se pueden obtener una serie de datos estadísticos. Los principales valores que debe proporcionar NessieOcr son los tiempos de ejecución de cada etapa interna al subsistema. Para un desarrollador también puede ser interesante obtener parámetros concretos relacionados con alguna de las acciones que se llevan a cabo en el caso de uso "Reconocer el texto", descrito en la sección 2.4.1.

El clasificador de noticias necesita saber ciertos aspectos de la naturaleza del texto reconocido, como el ancho y el alto promedio de los caracteres o la frecuencia de palabras. Hay otros valores que, aunque puede tener más sentido calcularlos en el ámbito del clasificador de noticias, puede significar una mejora en el rendimiento hacerlo dentro de NessieOcr. Estos últimos sólo se contemplarán a petición de dicho subsistema, quedando relegados por el momento.

2.5. Requisitos organizativos

Una de las motivaciones de este proyecto es realizar un software reutilizable, flexible y portable, junto con el deseo de emplear un lenguaje claro en el que sea sencillo descubrir el algoritmo.

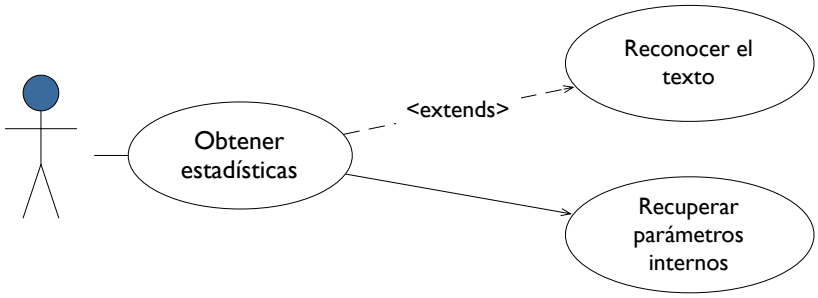


Figura 2.8: Caso de uso "Obtener estadísticas".

Si bien la naturaleza del producto parece indicar la aplicación de un diseño estructurado, creemos más apropiado emplear una metodología *orientada a objetos* tanto en el diseño como en la implementación. Para ello hemos adaptado a nuestras necesidades el "Proceso unificado de desarrollo de software" [17], eliminando todos aquellos aspectos y tareas que son aplicables a software más complejo o de otra naturaleza.

El resto de esta sección está dedicado a exponer los requisitos organizativos relativos a las herramientas necesarias para el desarrollo, justificando en cada caso el motivo de su elección.

2.5.1. Lenguaje de programación

Uno de los factores que influye de manera determinante para garantizar que el producto cumple con los requisitos de rendimiento fijados en la sección 2.2 es el lenguaje de programación. Tras evaluar las características y el rendimiento de varios lenguajes hemos determinado emplear *C++*.

C++ [36] es un lenguaje de programación de propósito general, definido como de medio nivel ya que su filosofía es una combinación de características entre la programación de alto y bajo nivel. Es un lenguaje compilado que genera código para el hardware de la máquina objeto, permitiendo programación estructurada, abstracción de datos, aplicación de paradigmas múltiples, programación orientada a objetos y programación genérica. El lenguaje se encuentra definido en el estándar ANSI/ISO C++ de 1998, que se compone de dos partes: el núcleo del lenguaje, heredado del lenguaje C, y la biblioteca estándar de C++. Esta última se compone de la "Standard Template Library" (STL) y una versión ligeramente modificada de la biblioteca estándar de C.

De todos los compiladores disponibles en el mercado hemos utilizado *GNU g++*, perteneciente a la familia de compiladores "GCC: The GNU Compilar Collection". El compilador g++ sigue las directrices del estándar ANSI/ISO C++ de 1998 y contiene además soporte experimental para muchas de las características del próximo estándar que está actualmente en vías de aprobación.

2.5.2. Manipulación de imágenes

Reconocer el texto presente en una imagen de la manera más óptima posible es una tarea compleja en sí misma. De qué manera se obtienen las imágenes a procesar y cómo se almacenan para su posterior manipulación deberían ser flujos de trabajo independientes de NessieOcr. Por ello hemos apostado por invertir la mayor parte del esfuerzo en diseñar NessieOcr de forma

transparente al concepto de imagen, pensando en un nivel de abstracción superior en términos de recortes de prensa.

Existe una gran variedad de proyectos que ofrecen bibliotecas orientadas a la manipulación de imágenes, cada una de ellas con unas características y una filosofía propias. Algunas están optimizadas para un formato de imagen específico, mientras que otras ofrecen simplemente la estructura de datos para cargar la imagen pero no métodos de manipulación adicionales. De todas las candidatas evaluadas finalmente se ha decidido utilizar *Magick++*, una API para interactuar con ImageMagick desde código escrito en C++ usando orientación a objetos.

ImageMagick

ImageMagick [14] es un conjunto de programas para crear, manipular y mostrar imágenes en forma de mapa de bits, soportando cerca de cien formatos de imágenes distintos. Su principal uso está orientado a realizar transformaciones en el contenido de las imágenes, como translación, rotación, recorte, escalado, ajuste de color, dibujo de líneas y polígonos, etc. Se distribuye bajo una licencia de software permisiva, aunque por definición ImageMagick es software libre, y su código es mantenido por una amplia comunidad de usuarios.

El software se compone principalmente de un conjunto de utilidades de línea de comandos para editar las imágenes, a diferencia de otros programas que disponen de una interfaz gráfica, como Photoshop o GIMP. Además de estas utilidades, ImageMagick proporciona una serie de APIs para muchos lenguajes de programación, entre las cuales destacan las interfaces para C/C++, Perl y PHP. Una de las características más usadas de ImageMagick es su capacidad para pasar una imagen de un formato a otro de forma precisa y eficiente. Además, ImageMagick también tiene implementados muchos métodos de procesamiento de clásicos, como ecualización, cuantización de colores, conversión entre espacios de color, umbralizado o eliminación de ruido.

ImageMagick, junto con sus principales APIs, está disponible para la mayoría de los sistemas operativos, incluyendo Microsoft Windows, Linux, Mac OS X y otros sistemas basados en UNIX. Aparte de los archivos binarios, también se puede obtener el código fuente para compilarlo en plataformas específicas.

Magick++: accediendo a ImageMagick con C++

Magick++ [15] es una interfaz para acceder a las funciones de ImageMagick desde programas escritos en C++. Su diseño ofrece una arquitectura de clases que permite leer y escribir un gran número de formatos de imagen, así como aplicar un amplio abanico de métodos tradicionales de procesamiento de imágenes de manera análoga a las utilidades de línea de comandos de ImageMagick.

La clase principal en Magick++ es *Image*, que proporciona métodos para cargar imágenes desde un soporte físico, realizar modificaciones sobre ella y volcar el resultado sobre la propia imagen o en una imagen nueva. A través de un objeto Image se pueden aplicar prácticamente la totalidad de las operaciones que ofrece ImageMagick en su versión de línea de comandos.

Cuando sea necesario realizar operaciones de bajo nivel en la imagen, la clase *Pixels* proporciona un acceso eficiente a los pixels a través de la *Image Pixel Cache*, un marco rectangular o *vista* que se define sobre la imagen. El tamaño de la vista puede ir desde un pixel hasta la imagen entera. Magick++ sólo permite acceder directamente a los pixels que están dentro de la vista.

La clase *Color* es una clase base para representar los valores de los pixels de una imagen. A grandes rasgos, podemos verla como un contenedor para los valores del espacio de color RGB más un cuarto valor de transparencia (canal alfa). Normalmente la clase Color no se usa como tal, sino una de sus clases derivadas dependiendo del espacio de color con el que se trabaje.

2.5.3. Control de versiones

El control de versiones del código fuente nos va a permitir gestionar todos los cambios que se van sucediendo a medida que avanza el proceso de desarrollo. Aunque este proyecto tiene una envergadura limitada y dicho control puede realizarse de forma manual, creemos que disponer de herramientas que faciliten esta gestión hace más robusta la metodología de trabajo.

En este sentido se ha decidido emplear *Subversion*, un software libre bajo licencia Apache/BSD de sistema de control de versiones multiplataforma. Subversion ha sido diseñado específicamente para reemplazar al popular CVS y suplir muchas de sus deficiencias. Entre sus muchas características destacamos las siguientes:

- La creación de ramas y etiquetas en el árbol de directorios es una operación más eficiente, con un costo de complejidad $O(1)$ y no lineal ($O(n)$) como el caso de CVS.
- A la hora de reflejar los cambios en el contenido se envían sólo las diferencias en ambas direcciones, no el fichero completo.
- Puede integrarse con Apache, mediante el protocolo WebDAV, permitiendo utilizar todas las opciones que este servidor provee para autenticación.

2.5.4. Documentación

Toda la documentación de este proyecto ha sido generada con LaTeX, un sistema de composición tipográfica de alta calidad considerada como un estándar *de facto* para la comunicación y publicación de documentos científicos. No se trata de un procesador de textos típico "What You See Is What You Get" (WYSIWYG), sino que el usuario debe centrarse en el contenido y dejar que el procesador LaTeX se encargue del aspecto final.

El código fuente ha sido documentado con *Doxygen*. La información se extrae directamente de los ficheros fuente, lo que facilita tener la documentación actualizada y coherente con el código. Doxygen puede generar la salida en muchos formatos, entre los que destacamos HTML y LaTeX. En las etapas de análisis y diseño se ha empleado *Omnigraffle* para la elaboración de todo tipo de diagramas UML.

2.6. Casos de uso completos

Esta sección está pensada como referencia del modelo de casos de uso visto en la sección 2.4. En cada ficha se especifica todas las propiedades de cada caso de uso recogido en la fase de análisis de requisitos.

Nombre:	Reconocer el texto
Fecha:	23-07-2008
ID:	NO-1
Objetivo	Extraer el texto que se encuentra contenido en un recorte de prensa.
Actores	Controlador.
Precondiciones	La imagen de entrada debe estar codificada en un formato que pueda leerse por Magick++.
Flujo básico	1. El recorte de prensa se copia desde una imagen en una estructura interna. 2. El recorte de prensa se somete a un preprocesamiento que elimina información innecesaria y extrae las regiones de pixels que representan caracteres. 3. Por cada región se calcula un conjunto de características que identifican el carácter que representa la región y lo distinga del resto. 4. Cada conjunto de características se asocia a un carácter mediante un proceso de clasificación. 5. A partir de todos los caracteres reconocidos se construye un texto. 6. El texto se postprocesa según los requerimientos del subsistema que vaya a hacer uso de él.
Flujos alternativos	Si ocurre cualquier error se construye un texto vacío y se notifica el fallo ocurrido al controlador.
Postcondiciones	A partir del recorte de prensa se genera su texto correspondiente y un conjunto de estadísticas sobre la ejecución.
Rendimiento	El conjunto de recortes que pertenece a una página debe procesarse como mucho en veinte segundos.
Fiabilidad	En caso de error al procesar un recorte, debe ser posible recuperarse y proseguir con el siguiente.

Nombre:	Entrenar el reconocedor
Fecha:	15-09-2008
ID:	NO-2
Objetivo	Mejorar la tasa de acierto del reconocedor presentando recortes de prensa cuyo texto de salida se conoce a priori.
Actores	Controlador.
Precondiciones	El texto de referencia para comparar con el resultado de la clasificación debe coincidir exactamente con el contenido del recorte de entrada. Además, la imagen de entrada debe estar codificada en un formato que pueda leerse por Magick++.
Flujo básico	1. El recorte de prensa se copia desde una imagen en una estructura interna. 2. El recorte de prensa se somete a un preprocesamiento que elimina información innecesaria y extrae las regiones de pixels que representan caracteres. 3. Por cada región se calcula un conjunto de características que identifican el carácter que representa la región y lo distinga del resto. 4. Cada conjunto de características se asocia a un carácter mediante un proceso de clasificación. 5. A partir de todos los caracteres reconocidos se construye un texto. 6. El texto se generado se compara con un texto de referencia carácter a carácter. 7. Por cada carácter no reconocido se ajustan los parámetros de la etapa de clasificación.
Flujos alternativos	Si el carácter reconocido no coincide con el carácter esperado se descarta y se sigue con el siguiente.
Postcondiciones	La información del clasificador es actualizada para mejorar la tasa de acierto. Además se generan estadísticas sobre la ejecución.
Rendimiento	No existen restricciones.
Fiabilidad	En caso de error al procesar un recorte, debe ser posible recuperarse y proseguir con el siguiente entrenamiento.

Nombre:	Obtener estadísticas
Fecha:	23-07-2008
ID:	NO-3
Objetivo	Recuperar parámetros de interés relacionados con la ejecución del reconocimiento de texto.
Actores	Controlador.
Precondiciones	Debe haberse ejecutado al menos una de las etapas del caso de uso "Reconocer el texto" o "Entrenar el reconocedor".
Flujo básico	Al finalizar el caso de uso "Reconocer el texto" o "Entrenar el reconocedor", se recuperan los datos estadísticos asociados a cada una de sus etapas.
Flujos alternativos	No existen. Si no hay datos que recuperar no se realiza ninguna acción.
Postcondiciones	No existen.
Rendimiento	No existen restricciones.
Fiabilidad	En caso de error al recuperar algún parámetro es deseable que el resto sí se pueda obtener. En cualquier caso debe ser posible recuperarse del error y continuar.

3 Preprocesamiento de imágenes

El preprocesamiento es una de las tareas del caso de uso "Reconocer el texto" (Sección 2.4). Su objetivo es reducir el nivel de información presente en la imagen de entrada, de cara a eliminar datos innecesarios o redundantes. También se intenta mejorar la calidad de la imagen, ya sea eliminando el ruido o resaltando aquellas propiedades que sean de utilidad en etapas posteriores. Al final de la etapa tenemos que obtener una lista de patrones preparada para la siguiente etapa: la extracción de características.

A lo largo de este capítulo vamos a explicar el conjunto de algoritmos de procesamiento de imágenes que hemos seleccionado tras una serie de estudios previos para cumplir con los requisitos expuestos. Todos ellos se agrupan en tres bloques:

- Eliminación de ruido.
- Segmentación de regiones.
- Creación de patrones.

El primer bloque lo componen todos los algoritmos que limpian la imagen de ruido y eliminan información que no es de interés, mejorando la calidad de la imagen para facilitar el trabajo de los bloques siguientes. En la segmentación de regiones analizamos la imagen localizando los conjuntos de pixels conexos que forman cada objeto, extrayéndolos y ordenándolos en una lista aparte. En el último bloque utilizamos la lista de regiones para crear un conjunto de patrones que será la información a entregar a la siguiente etapa.

3.1. Propiedades de las imágenes

Antes de entrar en el análisis de los algoritmos vamos a definir las características de las imágenes con las que trabaja NessieOcr. Recordemos que las imágenes se cargan desde un soporte externo utilizando Magick++, la API descrita en los requisitos organizativos del capítulo 2.

3.1.1. Representaciones

Una imagen puede representarse de varias maneras, dependiendo de la estructura de datos en la que es cargada. Las tres más importantes son:

- Una *matriz*, donde cada celda representa un pixel y almacena su valor en un espacio de color dado. El número de filas y columnas de la matriz define el alto y el ancho de la imagen respectivamente.
- Un *histograma*, que define la distribución estadística de los valores de los pixels en un espacio de color.
- Un *función bidimensional de intensidad de luz* $f(x, y)$, donde x e y representan las coordenadas espaciales y el valor de f en un punto cualquiera representa un valor proporcional al valor de intensidad en ese punto. Considerar una imagen como una función permite realizar transformaciones sobre ella utilizando operadores matemáticos.

NessieOcr utiliza principalmente las dos primeras representaciones. La representación matricial se utiliza prácticamente en todos los algoritmos que procesan directamente la imagen, mientras que el histograma se emplea tan sólo en unos pocos métodos.

3.1.2. Espacio de color

Las imágenes de entrada al sistema son recortes de prensa. Salvo raras excepciones esto significa trabajar con un fondo de color uniforme sobre el que aparecen una serie de objetos, de un color cuyo contraste es bastante alto con respecto al fondo. El color del fondo suele ser blanco y el de los caracteres es principalmente negro. Sin embargo algunos tipos de titular muchas veces intercambian el diseño, usando de fondo algún color oscuro y colocando los caracteres en blanco, amarillo, etc.

3.1.3. Presencia de ruido

El ruido que aparece en los recortes de prensa degrada la imagen e incluye información que no es relevante. De los diferentes tipos de ruido que se conocen nos interesa el *ruido de impulsos*, que aparece como consecuencia de una mala calidad en el proceso de escaneado de periódicos. El ruido de impulsos hace que en una imagen aparezcan pixels más o menos aislados cuyo color no difiere del color usado para la tinta de los caracteres. Las imágenes que vengan en formato PDF cuyo contenido haya sido generado en formato vectorial, no a partir de una imagen previamente escaneada, no tienen este problema.

3.1.4. Relaciones de localidad entre pixels

Es interesante considerar algunas relaciones de localidad entre pixels. Para ello vamos a definir la imagen como una función $f(i, j)$, siendo i y j las coordenadas de un pixel p y el valor de la función en ese punto el color del pixel.

- La *vecindad de un pixel* viene determinada por los pixels inmediatamente adyacentes a él. Un pixel p de coordenadas (i, j) tiene cuatro vecinos $N_4(p)$ cuyas coordenadas son:

$$(i+1, j), (i-1, j), (i, j+1), (i, j-1) \tag{3.1.1}$$

  Ese mismo pixel tiene además cuatro vecinos en diagonal, cuyas coordenadas son:

$$(i+1, j+1), (i+1, j-1), (i-1, j+1), (i-1, j-1) \tag{3.1.2}$$

Uniendo todos los vecinos podemos obtener el conjunto $N_8(p)$ que nos da los ocho vecinos de un pixel; junto con él forman una matriz de dimensión $3x3$.

- La *conectividad entre pixels* establece los límites de los objetos en la imagen. Dos pixels están conectados si son adyacentes en algún sentido y si sus niveles de intensidad cumplen un criterio previamente definido. Sea V el conjunto de colores que establece el criterio de conectividad entre dos pixels. En una imagen dos pixels p y q estarán conexos si y sólo si sus colores pertenecen a V y además son vecinos.
- Una última relación de localidad viene dada por la *distancia entre pixels*. Una función D define una distancia si cumple la siguientes condiciones:

  1. $d(p,q) \geq 0$ ($d(p,q) = 0$ si y sólo si $p = q$).
  2. $d(p,q) = d(q,p)$
  3. $d(p,z) \leq d(p,q) + d(q,z)$

  Algunos ejemplos de distancia son la distancia *euclídea*, la distancia *Manhattan* o la distancia de *tablero de ajedrez*.

3.1.5. Relaciones de localidad entre caracteres

A nivel macroscópico también podemos definir una serie de relaciones de localidad, esta vez usando el espacio entre caracteres y palabras como referencia:

- El *espacio interpalabra* es aquel que existe entre dos palabras consecutivas dentro de una misma línea.
- El *espacio intercarácter* es aquel que existe entre dos caracteres consecutivos dentro de una misma palabra.
- El *espacio intracarácter* es aquel que no forma parte del trazo del carácter, lo que se considera en segmentación de imágenes como hoyos en la superficie del objeto. Por ejemplo, en la letra "O" el espacio intracarácter es la superficie encerrada por el contorno de la letra.

Cuando hablamos de espacio nos referimos al conjunto de pixels cuyo color es idéntico al fondo de la imagen. Muchas veces el proceso de escaneado de imágenes viola este espacio, y provoca la aparición de caracteres deformados, letras unidas, etc.

En el caso de los espacios interpalabra e intercarácter el conjunto de pixels que separa dos elementos quedaría definido por la distancia entre el pixel más a la derecha del primer elemento y el pixel más a la izquierda del segundo elemento.

3.2. Eliminación de ruido

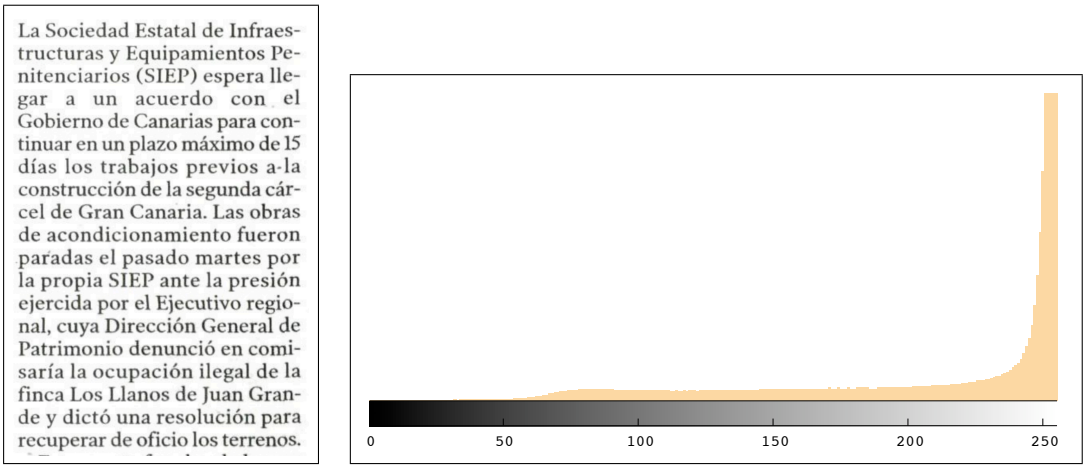
En una imagen de un recorte de prensa, el ruido representa todos aquellos pixels cuyo valor de intensidad coincide con el valor empleado para la tinta pero que no pertenecen a la forma de ningún objeto. Eliminar estos conjuntos de pixels es necesario para no confundirlos con otros caracteres y generar patrones ficticios que no existen en la imagen real.

Existen muchos métodos para eliminar ruido y mejorar la calidad de las imágenes. Sin embargo, hay que tener en cuenta las restricciones de tiempo a las que se enfrenta NessieOcr para

descartar aquellos cuya complejidad computacional es alta. Por ejemplo, realizar un promediado de muchas capturas del mismo recorte no es viable. Otros los descartamos simplemente porque no se ajustan a las propiedades de las imágenes de entrada. En este sentido, aplicar métodos de ecualización de histogramas no resulta útil porque la propia naturaleza de los recortes de prensa hace que los objetos ya tengan un alto contraste con respecto al fondo. Los algoritmos de eliminación de ruido que describimos a continuación se han elegido teniendo en cuenta todas estas consideraciones, y han sido incluidos en la implementación de NessieOcr.

3.2.1. Conversión a escala de grises

Manejar un espacio de color complejo implica procesar una mayor cantidad de información y consumir mayor cantidad de memoria para almacenarla. Un recorte de prensa se caracteriza por tener histogramas muy simples, con dos rangos de color perfectamente localizados. La figura 3.1 muestra un ejemplo de ello.



(a) Recorte de prensa **(b)** Histograma

Figura 3.1: Histograma típico de las imágenes que contienen recortes de prensa.

Un espacio de color muy común es el RGB. En esta representación tendríamos que trabajar con tres matrices de las dimensiones de la imagen, donde cada matriz almacena la información de un canal. Convertir la imagen original a un espacio de color que use como representación una escala de grises nos permite tener una única matriz de pixels. En ella, cada celda corresponde con el valor de gris de un pixel equivalente a la unión de los tres canales de una imagen RGB. Esto facilita la construcción de algoritmos para ejecutar acciones sobre la imagen, ahorra espacio en memoria y reduce el tiempo de ejecución. Para NessieOcr hemos decidido que, por defecto, el espacio de color todas las imágenes se convierta a escala de grises.

La conversión desde un espacio de color a escala de grises se realiza mediante una función conocida, que es una combinación lineal de los valores de intensidad de los canales de color de una imagen RGB. Para cada pixel p de coordenadas (i, j) sean F_r, F_g y F_b las funciones que devuelven el valor de color de cada canal, y B la función que devuelve el valor de intensidad en escala de grises. La ecuación (3.2.1) define la relación entre ambos espacios.

$$b(i,j) = 0.3f_r(i,j) + 0.59f_g(i,j) + 0.11f_b(i,j) \tag{3.2.1}$$

3.2.2. Promediado en base a filtros

Supongamos que el ruido ν en una imagen de L pixels es una variable aleatoria independiente sin ninguna correlación y cuyo valor medio es cero. Si obtenemos la misma imagen M veces podemos seleccionar el mismo pixel p de todas las imágenes. Una estimación del valor correcto z del pixel puede obtenerse promediando todos los valores de p, incluyendo aquellos que se corresponden con ruido:

$$z(i) = \frac{1}{M} \sum_{j=1}^{M} p_j(i), \quad \forall i = 1 \dots L \tag{3.2.2}$$

La etapa de preprocesamiento de NessieOcr sólo recibe una única imagen del recorte de prensa, así que no es factible realizar un promediado de estas características. Por el contrario, la técnica que presentamos a continuación utiliza la vecindad de un pixel para obtener los valores de referencia.

El *promediado en base a filtros* es una técnica de convolución que asigna un nuevo valor de color a cada pixel p mediante una combinación lineal de los valores de color de los pixels de su vecindad N_8. Un filtro o máscara se define como una matriz $3x3$ con una serie de valores asignados, que se hace coincidir sobre los pixels de la imagen que forman la vecindad $N_8(p)$, estando $p_{i,j}$ en la celda central y siendo i y j las coordenadas del pixel en la imagen:

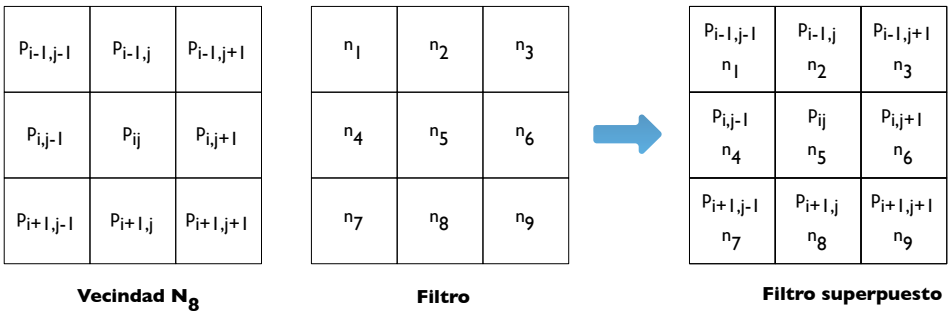


Figura 3.2: Superposición de un filtro de promediado sobre una vecindad N_8 de un pixel.

Si tomamos el filtro como un vector X y la vecindad $N_8(p)$ como un vector Y, el nuevo color z de un pixel $p_{i,j}$ se calcula realizando el producto interior de ambos vectores:

$$z(i) = X \cdot Y = X^T Y = \begin{bmatrix} x_1 & x_2 & \dots & x_8 \end{bmatrix} \begin{bmatrix} p_{i-1,j-1} \\ p_{i-1,j} \\ \vdots \\ p_{i+1,j+1} \end{bmatrix} \tag{3.2.3}$$

Para que la operación tenga el resultado esperado hay que respetar la correspondencia entre los valores del filtro y la vecindad cuando se superponen sobre la imagen, tal como muestra la figura 3.2. Las siguientes ecuaciones definen dos filtros muy usados en procesamiento de imágenes, donde L es el tamaño de la imagen:

$$z(i) = \frac{1}{9} \times \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \times N_8(p(i)), \quad \forall i = 1 \dots L \tag{3.2.4}$$

$$z(i) = \frac{1}{16} \times \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} \times N_8(p(i)), \quad \forall i = 1 \dots L \tag{3.2.5}$$

Con esta técnica se consigue difuminar el ruido de impulsos para separar el nivel de gris de los pixels de ruido del nivel de la tinta de los caracteres. Otra consecuencia es que los bordes de los caracteres también se difuminan, eliminando detalles irrelevantes. El método debe ejecutarse un número limitado de veces, puesto que de otra forma pueden llegar a desaparecer características importantes de los objetos. En la figura 3.3 podemos ver un ejemplo de una imagen filtrada según este algoritmo.

(a) Imagen original **(b)** Imagen filtrada

Figura 3.3: Promediado en base a filtros aplicado a un recorte de prensa.

3.2.3. Binarización del espacio de color

Una de las propiedades de los recortes de prensa es el alto contraste entre el color del fondo y el color de los caracteres. La técnica de binarización usa este conocimiento para establecer un umbral que separe los objetos y el fondo, de forma que tras su aplicación la imagen sólo tenga dos colores. Este método también recibe el nombre de umbralizado.

La definición formal de la binarización consiste en la transformación de una imagen de entrada F en una imagen de salida G, con dos únicos niveles de intensidad establecidos a partir de un umbral ϕ. Los pixels de la imagen de salida cuyo nivel de gris sea 0 representan el fondo, mientras que los pixels con valor 1 representan los objetos:

$$g(i,j) = \begin{cases} 0 & \forall\, f(i,j) \geq \phi \\ 1 & \forall\, f(i,j) < \phi \end{cases} \tag{3.2.6}$$

La correcta elección del valor del umbral ϕ es determinante para que no se eliminen detalles relevantes de los objetos. Esta decisión puede tomarse de manera empírica, observando las intensidades más frecuentes en los objetos de una muestra suficiente de imágenes, o usando

algún método de detección automática del umbral. La mayoría de los métodos de detección automática del umbral se basan en el estudio del histograma de la imagen.

El *método de Otsu* [32] es un algoritmo de detección automática del umbral de una imagen, que se considera teóricamente óptimo para esa imagen. Este método es muy utilizado en el campo del procesamiento de imágenes, donde ha demostrado ser el más preciso y eficiente. La estrategia propuesta por Nobuyuki Otsu maximiza la probabilidad de que al elegir un cierto umbral la imagen quede perfectamente dividida entre los objetos y el fondo. O visto de otra manera, lo que se busca es minimizar la probabilidad de clasificar un pixel del fondo como perteneciente a un objeto o viceversa.

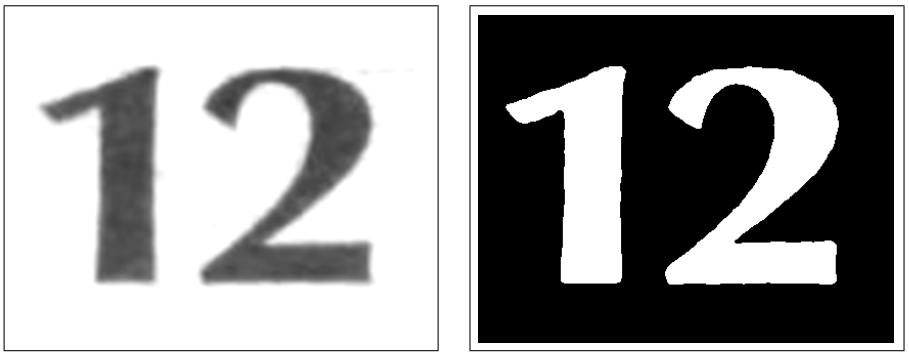
El procedimiento es muy simple, utilizando únicamente la información del histograma de la imagen para realizar cálculos estadísticos. Usando análisis de discriminantes, se demuestra fácilmente que el valor de la varianza en una misma clase (σ_W^2), entre diferentes clases (σ_B^2) o la varianza total (σ_T^2) alcanza su máximo valor en el valor del umbral que estamos buscando. El umbral óptimo ϕ^* puede determinarse maximizando una de las siguientes funciones con todos los posibles umbrales:

$$\lambda = \frac{\sigma_B^2}{\sigma_W^2}, \quad \eta = \frac{\sigma_B^2}{\sigma_T^2}, \quad \kappa = \frac{\sigma_T^2}{\sigma_W^2} \tag{3.2.7}$$

Tomando η, por ejemplo, el umbral óptimo ϕ^* quedaría definido por la ecuación(3.2.8), en la que L es el valor máximo que puede tomar el umbral:

$$\phi^* = \arg \max_{t \in [0, L-1]} (\eta), \tag{3.2.8}$$

En la figura 3.4 podemos ver un ejemplo de una imagen binarizada usando el algoritmo de Otsu para determinar el umbral óptimo.



(a) Imagen original **(b)** Imagen binarizada

Figura 3.4: Binarización aplicada a un recorte de prensa.

3.2.4. Filtros en base a plantillas

A pesar de que la mayoría del ruido de la imagen se elimina con el filtrado y la binarización, es posible que en los bordes de los objetos aún hayan artefactos que deformen su silueta. Incluso puede darse el caso de que algún pixel de ruido se encuentra aislado en el fondo. La aplicación de

filtros en base a plantillas es una última operación de eliminación de ruido que intenta solucionar estos casos.

Partimos de una imagen binarizada, donde el color del fondo está representado por un 0 y el color de los objetos está representado por un 1. La aplicación de las siguientes plantillas ayuda a suavizar los bordes y quitar el ruido remanente:

$$\begin{vmatrix} = & = & = \\ = & T & = \\ x & x & x \end{vmatrix}, \begin{vmatrix} x & = & = \\ x & T & = \\ x & = & = \end{vmatrix}, \begin{vmatrix} x & x & x \\ = & T & = \\ = & = & = \end{vmatrix}, \begin{vmatrix} = & = & x \\ = & T & x \\ = & = & x \end{vmatrix} \tag{3.2.9}$$

Estas plantillas son aplicadas sobre la imagen, repitiendo el proceso hasta que no haya ningún cambio. El proceso comienza en la esquina inferior derecha de la imagen, y continua hacia arriba de derecha a izquierda hasta terminar en la esquina superior izquierda. El pixel que cae en el centro de la plantilla es el objetivo T. Los pixels que caen sobre las celdas marcadas con una x se ignoran. Si todos los pixels que caen sobre las celdas marcadas con un $=$ tienen el mismo valor, entonces al pixel T se le asigna ese valor. Si no se cumple esta condición, el pixel se queda como está.

Los resultados de este algoritmo son muy difíciles de apreciar a simple vista, excepto para algunos recortes que proceden de un escaneado muy degradado. En el ejemplo de la figura 3.5 podemos ver que sobre todo los trazos rectos de los caracteres son los que más se benefician.

(a) Imagen binarizada **(b)** Imagen filtrada

Figura 3.5: Eliminación de ruido en base a plantillas aplicadas sobre un recorte de prensa.

3.3. Segmentación de regiones

La segmentación es la tarea de la etapa de preprocesamiento que se encarga de extraer los objetos de interés —los caracteres— que se encuentran en la imagen. El objetivo es generar una lista de regiones de pixels, que sirva para construir a continuación los patrones destinados a ser clasificados.

Una segmentación completa de una imagen R se compone de un conjunto de regiones R_i que comparten una cierta propiedad. En nuestro caso, dicha propiedad estipula que todos los pixels de una región R_i tienen el mismo valor y están conectados. Formalmente, la segmentación se define como:

$$R = \bigcup_{i=1}^{S} R_i, \quad R_i \cap R_j = \emptyset, \quad i \neq j \tag{3.3.1}$$

La manera en que los pixels de una imagen son asignados a su región correspondiente son variados. A continuación analizamos uno de ellos, el *crecimiento de regiones por agregación de pixels*, que permite construir la lista de regiones recorriendo la imagen una sola vez. Esta técnica ha sido la elegida para la implementación de NessieOcr.

3.3.1. Crecimiento de regiones por agregación de pixels

El crecimiento de regiones por agregación permite almacenar en una estructura de datos todas las regiones de una imagen. El método se basa en agregar los pixels vecinos de un pixel generador o semilla s, si éstos cumplen un predicado P. Por tanto, podemos ampliar la definición formal de segmentación de una imagen R de la ecuación (3.3.1) con las siguientes condiciones:

1. $\bigcup_{i=1}^{n} R_i = R$
2. R_i es una región conexa, $i = 1, 2, \ldots, n$
3. $R_i \cap R_j = \emptyset$, $\forall i, j$, $i \neq j$
4. $P(R_i) =$ VERDADERO, $\forall i = 1, 2, \ldots, n$
5. $P(R_i \bigcup R_j) =$ FALSO, $i \neq j$

La primera condición establece que la segmentación debe ser completa; cada pixel debe estar en una región. La segunda condición requiere que los puntos de una región deben ser conexos (véase la conectividad en la sección 3.1.4). La tercera condición obliga a que las regiones deben ser disjuntas. La cuarta condición define el predicado que debe satisfacer los pixels de una región segmentada. Finalmente, la quinta condición indica que las regiones R_i y R_j son diferentes en el sentido del predicado P.

El funcionamiento del algoritmo es muy intuitivo. Partimos de una *matriz de visitados*: una estructura de valores lógicos del mismo tamaño que la imagen inicializada al valor **falso**. Luego se construye un *conjunto de semillas* tomando todos los pixels cuyo color sea igual al color de los objetos de interés, suponiendo que la imagen está libre de ruido. A partir de cada semilla se exploran sus vecinos, verificando que cumplen el predicado P. Cada pixel que se incluye en la región se elimina del conjunto de semillas, mientras que cualquier pixel que se visite se marca como **verdadero** en la matriz de visitados. Al final del proceso todos los pixels de la imagen deben haber sido visitados, y todos los pixels de los objetos deben estar incluidos en una única región.

Con todas las regiones obtenidas se construye una lista, en la que cada nodo contiene una región y varios parámetros adicionales. En la figura 3.6 podemos ver el significado de estos parámetros en una región de ejemplo. El primer parámetro lo constituye las coordenadas absolutas del baricentro B de la región con respecto a la imagen global. También se guardan las coordenadas de cuatro puntos de localización P_N, P_S, P_E y P_O, que coinciden con los pixels que se encuentran más al norte, al sur, al este y al oeste respectivamente. Y por último se almacenan

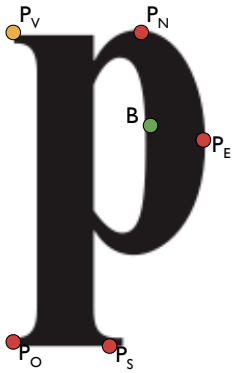


Figura 3.6: Puntos de localización de una región segmentada.

las coordenadas de un hipotético pixel P_V que se encontraría en la esquina superior izquierda de un marco virtual que encierre la región.

El último paso de este algoritmo consiste en ordenar la lista de regiones, de forma que recorrer la lista sea equivalente a leer el texto de arriba a abajo y de izquierda a derecha. Al mismo tiempo se obtiene el índice de los nodos entre los que es necesario insertar un espacio en blanco cuando se construya el texto final. Para ello podemos utilizar una *ordenación por inserción*, que es apropiada para listas de pequeño tamaño.

Existe un problema relacionado con este algoritmo, que complica este procedimiento obligando a realizar tareas adicionales. Si la imagen de entrada ha sido escaneada en una resolución baja aparece un fenómeno de *unión de caracteres*, en el que dos o más caracteres consecutivos se unen por alguna zona de sus siluetas. La figura 3.7 muestra un ejemplo de esta situación.

Figura 3.7: Fenómeno de unión entre caracteres independientes en una imagen de baja resolución.

Podemos optar por aumentar la resolución de las imágenes de entrada o separar automáticamente los caracteres. En este último caso necesitamos un mecanismo que detecte si efectivamente una región identificada como un objeto representa en realidad la unión de dos objetos independientes. Luego debemos decidir en qué punto cortamos la región para obtener los dos objetos por separado. En nuestro caso hemos decidido aumentar la resolución de las imágenes, ya que implementar un mecanismo de corte y vuelta a atrás en caso de error puede tener consecuencias negativas sobre la modularidad del diseño o el tiempo total de cómputo.

3.3.2. Fusión de caracteres compuestos

Un fenómeno que ocurre al extraer los objetos de la imagen usando el algoritmo de agregación de pixels es la separación de los caracteres compuestos por dos o más formas. Las vocales

acentuadas, la letra "i" o el símbolo "%" pertenecen a este grupo de caracteres. Para unir dos o más regiones que pertenezcan a un mismo carácter tenemos que explotar las propiedades de localidad espacial entre regiones.

Las regiones que forman un carácter compuesto tienen la propiedad de estar muy cercanas entre sí. De hecho, todos los acentos se encuentra siempre encima de su vocal correspondiente. Por tanto, podemos realizar una búsqueda de aquellas regiones que compartan un cierto número de pixels con el mismo valor de coordenadas en el eje de abscisas. En la figura 3.8 se muestra un ejemplo de esta situación. El rango delimitado por las líneas que salen del acento define la zona de pixels con coordenadas comunes.

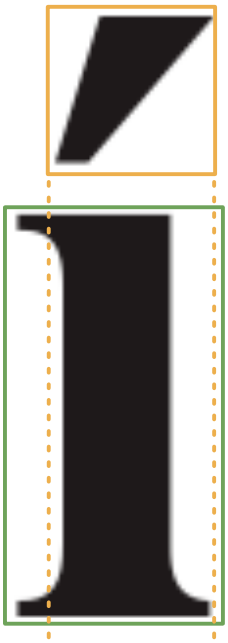


Figura 3.8: Ejemplo de dos regiones que forman un único carácter. Las líneas que salen del acento permiten definir una zona en la vocal donde hay pixels con coordenadas en el eje de abscisas en común.

Un problema colateral consiste en establecer un criterio por el que la búsqueda de regiones se realice sólo con los caracteres de una misma línea. Esto asegura que no se comparan caracteres que están uno encima de otro pero en diferentes líneas. Todos estos aspectos han sido tenidos en cuenta para implementar la fusión de caracteres compuestos en NessieOcr.

3.4. Creación de patrones

Las regiones de pixels que se extraen en la fase de segmentación presentan características dispares. Sus tamaños son distintos, ya que lógicamente el número de pixels de tinta necesarios para imprimir un carácter varía de uno a otro, y cada región contiene información redundante sobre el contorno, ya que una letra siempre es la misma independientemente del estilo de la tipografía.

Mediante la creación de patrones pretendemos obtener una representación homogénea de las regiones extraídas en la segmentación de la imagen. Este conjunto de patrones será la entrada de la siguiente tarea: la extracción de características. Con la normalización de regiones hacemos que todas las regiones sean de un tamaño homogéneo, mientras que la esqueletización elimina la información sobrante acerca del contorno de cada objeto.

3.4.1. Normalización de regiones

La normalización de regiones está considerada como un paso fundamental en el preprocesamiento. Esta operación toma cada región y la ajusta al tamaño de un plano estándar, de forma que al final todas las regiones tienen las mismas dimensiones. El tamaño de este plano normalmente tiene dimensiones entre 32×32 y 64×64. Lo que buscamos con esta transformación es que la variación entre muestras de una misma clase (un mismo carácter en diferentes tipografías) sea mínima, y así mejorar la tasa de acierto en la etapa de clasificación.

Método	**Función**
Relación de aspecto constante	$R_2 = 1$
Relación de aspecto conservadora	$R_2 = R_1$
Raíz cuadrada de la relación de aspecto original	$R_2 = \sqrt{R_1}$
Raíz cúbica de la relación de aspecto original	$R_2 = \sqrt[3]{R_1}$
Seno de la relación de aspecto original	$R_2 = \sqrt{\sin \frac{\pi}{2} R_1}$

Tabla 3.1: Funciones de transformación de la relación aspecto.

Existen básicamente dos métodos de normalización: los lineales y los no lineales. De todos ellos quizá el más conocido sea la normalización adaptativa de la relación de aspecto ("aspect ratio adaptive normalization", ARAN). En ella, la relación de aspecto de la región original R_1 se transforma en una nueva relación de aspecto R_2, usando un función asociativa predefinida (Tabla 3.1). La relación de aspecto se define según las siguientes ecuaciones, donde W_1 y H_1 representan el ancho y el alto de la región, y W_2 y H_2 el ancho y el alto del plano estándar:

$$R_1 = \frac{\min(W_1, H_1)}{\max(W_1, H_1)} \tag{3.4.1}$$

$$R_2 = \frac{\min(W_2, H_2)}{\max(W_2, H_2)} \tag{3.4.2}$$

La posición de cada pixel en la región original se ubica en una nueva posición en el plano estándar, usando una función de conversión de coordenadas. Varios ejemplos de dichas funciones se encuentran en la tabla 3.2, donde α y β son las relaciones de escalado según las ecuaciones

$$\begin{aligned} \alpha' &= W_2/W_1, \\ \beta' &= H_2/H_1 \end{aligned} \tag{3.4.3}$$

Una vez que todos los pixels de la región han sido llevados al plano estándar, es necesario aplicar un proceso de interpolación si la conectividad entre dos o más pixels ha desaparecido al ubicarlos en sus nuevas posiciones. Magick++ dispone de dos métodos de normalización

Método	Función
Lineal	$x' = \alpha x$ $y' = \beta y$
En base a momentos	$x' = \alpha(x - x_c) + x'_c$ $y' = \beta(y - y_c) + y'_c$
No lineal	$x' = W_2 h_x(x)$ $y' = H_2 h_y(y)$

Tabla 3.2: Funciones de conversión de coordenadas para normalizar una región. Los valores de α y β se definen en las ecuaciones (3.4.3).

que aplica una discretización de coordenadas para rellenar los pixels de fondo que pudieran aparecer entre dos o más pixels anteriormente conectados. Estos métodos son *Image::sample()* e *Image::scale()*, y han sido utilizados en NessieOcr para implementar esta técnica.

3.4.2. Esqueletización

El concepto de esqueleto fue introducido por Manuel Blum como resultado de su investigación sobre la transformada del eje medio (MAT). Esta transformación determina los puntos del contorno más cercanos a cualquier punto de un objeto. Un punto interior de un objeto pertenece al esqueleto si está conectado con al menos dos puntos del contorno. Encontrar el esqueleto de un carácter es necesario para eliminar información redundante sobre su forma. Por ejemplo, en una tipografía en negrita el trazo de cada carácter es más grueso, pero el esqueleto fundamental de la letra es el mismo que si no fuera negrita.

Hay diferentes aproximaciones para encontrar el esqueleto de un objeto. Una de ellas es el *adelgazamiento* del objeto, un proceso por el que vamos eliminando de un objeto tantos puntos como podamos sin afectar a su forma general.

NessieOcr implementa el *algoritmo de Zhang-Suen* para esqueletizar las regiones. Este algoritmo de adelgazamiento tiene dos pasos, que se aplican sucesivamente a cada región. En cada iteración los pixels del contorno de la región susceptibles de ser eliminados se marcan. Se considera que un pixel pertenece al contorno de la región si pertenece a la región y tiene al menos un pixel del fondo en su vecindad N_8. Los pixels de la vecindad N_8 se nombran según el siguiente esquema:

$$\begin{pmatrix} P_9 & P_2 & P_3 \\ P_8 & P_1 & P_4 \\ P_7 & P_6 & P_5 \end{pmatrix} \tag{3.4.4}$$

Para esta matriz definimos dos funciones $A(P_1)$ y $B(P_1)$, que tienen los siguientes significados:

- $A(P_1) =$ número de transiciones de 0 a 1 en la secuencia ordenada $P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_2$.
- $B(P_1) = P_2 + P_3 + P_4 + P_5 + P_6 + P_7 + P_8 + P_9$.

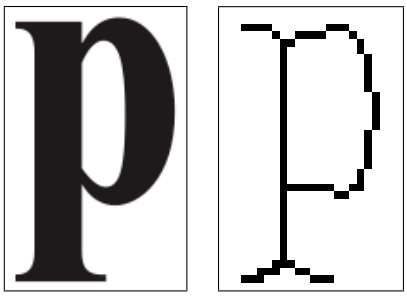
El primer paso del algoritmo marca un pixel P_1 del contorno si se cumplen cada una de las siguientes condiciones:

- *Condición 1:* $2 \leq B(P_1) \leq 6$.
- *Condición 2:* $A(P_1) = 1$.
- *Condición 3:* $P_2 \cdot P_4 \cdot P_6 = 0$.
- *Condición 4:* $P_4 \cdot P_6 \cdot P_8 = 0$.

El segundo paso del algoritmo marca un pixel P_1 del contorno si se cumplen cada una de las siguientes condiciones:

- *Condición 1:* $2 \leq B(P_1) \leq 6$.
- *Condición 2:* $A(P_1) = 1$.
- *Condición 3:* $P_2 \cdot P_4 \cdot P_8 = 0$.
- *Condición 4:* $P_2 \cdot P_6 \cdot P_8 = 0$.

El primer paso del algoritmo se aplica a todos los pixels de la región, aunque ningún pixel es eliminado hasta que no se termine de evaluar todo el conjunto de pixels del objeto. Esto se realiza así para evitar que cambios locales afecten a evaluaciones posteriores. A continuación se aplica el segundo siguiendo el mismo procedimiento, eliminado los pixels marcados. El proceso se ejecuta de manera consecutiva hasta detectar que no se ha producido ningún cambio. La región normalizada resultante constituye el esqueleto del objeto. En la figura 3.9 podemos apreciar un ejemplo del resultado de una esqueletización.



(a) Patrón original **(b)** Esqueleto

Figura 3.9: Resultado de la esqueletización aplicada a un carácter.

3.4.3. Corrección de inclinaciones

La inclinación que se encuentra normalmente en las tipografías itálica o cursiva se denomina pendiente. La corrección de tales inclinaciones es importante para garantizar una generación de patrones homogénea. Sin embargo, tiene más sentido cuando estamos tratando con textos escritos a mano, o textos que a priori sabemos que tienen una alta presencia de caracteres en cursiva.

Como sabemos, NessieOcr trabaja con caracteres impresos, en los que la aparición de caracteres en cursiva es despreciable. Por tanto, no es rentable el esfuerzo de diseñar e implementar un método de corrección de cursiva, sino confiar en la generalización de la clasificación.

Estos algoritmos se basan en la detección del ángulo que es necesario rotar los objetos, para luego aplicar una transformación conocida. La detección puede ser tan sencilla como forzar la rotación consecutiva del objeto un número de grados predefinido. En cada paso calculamos y almacenamos cuántos pixels tiene la columna con el mayor número de pixels pertenecientes al objeto en esa rotación. Tras terminar asumimos que, dada la naturaleza de los caracteres que estamos tratando, la rotación que otorgó la columna con el mayor número de pixels determina el ángulo que hay que rotar la región normalizada original. La transformación que hay que aplicar viene dada por la ecuación (3.4.5).

$$\begin{aligned} x' &= x - y \cdot \tan\theta, \\ y' &= y \end{aligned} \tag{3.4.5}$$

4 Clasificación de patrones

La finalidad última de un sistema de reconocimiento de patrones es obtener la clase a la que pertenece cada objeto extraído de una señal de entrada. Desde el punto de vista de NessieOcr un objeto es un carácter que está representado por un conjunto de pixels de tamaño fijo, al que denominamos patrón. Una clase es una etiqueta que identifica todas las formas de un carácter de un alfabeto dado, independientemente de la tipografía empleada. La figura 4.1 ilustra este principio, en el que diferentes formas —patrones— pertenecen a la misma clase: la letra "a".

Figura 4.1: Conjunto de formas distintas que pertenecen a la misma clase.

La clasificación de patrones es la etapa encargada de asociar un objeto a una clase. Durante el desarrollo histórico de la visión por computador se han estudiado y definido diferentes paradigmas de clasificación: métodos estadísticos, redes neuronales, métodos estructurales, etc. Para diseñar el clasificador de NessieOcr hemos optado por centrarnos en los *métodos estadísticos*, dado que tienen un fundamento teórico sólido, son relativamente simples de implementar y pueden aplicarse a una gran variedad de problemas. En este paradigma, la clasificación se lleva a cabo examinando un conjunto de características del objeto incógnita y comparándolo con las características que definen los objetos de cada clase conocida.

En este capítulo vamos a describir los fundamentos teóricos de los métodos de clasificación pensados para NessieOcr, así como las características elegidas para representar cada región de pixels que forma cada patrón.

4.1. Extracción de características

La elección de una buena representación del conocimiento es fundamental para obtener buenos resultados en la clasificación. Existe un principio general a aplicar cuando aparecen dificultades en la clasificación de patrones: "Antes que intentar resolver el problema a través de intrincados artificios matemáticos, es preferible buscar mejores características". Por tanto,

el esfuerzo de diseño debe centrarse en estudiar qué tipo de propiedades son más adecuadas para describir los caracteres adecuadamente. En este sentido es importante tener en cuenta los siguientes postulados propuestos por Niemann [31]:

- Representatividad: La construcción de un sistema de clasificación requiere de una muestra representativa de los objetos que podemos encontrar.
- Discriminabilidad: Siempre debe ser posible encontrar un conjunto de características que discriminen un objeto de otro en algún tipo de espacio.
- Compactibilidad: Las características de las formas de una categoría ocupan un dominio compacto de la representación, estando los dominios lo suficientemente separados

NessieOcr puede recibir como entrada recortes de prensa obtenidos a partir de periódicos escaneados, por lo que las propiedades a elegir deben ser invariantes, como mínimo, a la translación, la rotación y el escalado. También hemos de tener en cuenta la velocidad de la aplicación, por lo que no es deseable aplicar transformaciones complicadas de la señal de entrada para obtener ciertas propiedades.

Una característica de un patrón representa alguna propiedad escalar medible, como la altura, la anchura o el área de pixels. Es obvio que una única propiedad es insuficiente para caracterizar un objeto, por lo que las diferentes propiedades tienen que agruparse. Como NessieOcr utiliza métodos de clasificación estadísticos necesitamos hacer uso de vectores. Cada patrón se representa mediante un vector de características X que ubica el objeto en un punto de un espacio de características $\mathbb{R}^n$ conocido.

4.1.1. Análisis de imágenes usando momentos

La descripción de imágenes en base a momentos ha evolucionado en las últimas décadas hasta convertirse en una potente herramienta para las aplicaciones de análisis de imágenes. Los momentos de una imagen permiten describir las propiedades de manera eficiente y a bajo nivel. Su principal virtud es que esas descripciones son invariantes a las transformaciones que se hagan en la imagen. Tras años de investigación se han desarrollado diferentes formas de calcular los momentos de una imagen. La tabla 4.1 describe los más utilizados en aplicaciones hoy día.

Tipo	Sensibilidad al ruido	Complejidad computacional	Invarianza
Momentos geométricos	Alta	Baja	Mediante transformaciones
Momentos complejos	Alta	Baja	Nativa
Momentos de Legendre	Baja	Alta	Nativa
Momentos de Zernike	Baja	Muy alta	Nativa
Momentos de Tchebichef	Baja	Media	Mediante transformaciones
Momentos de Hahn	Baja	Media	Mediante transformaciones

Tabla 4.1: Resumen de los momentos de una imagen más utilizados.

Los momentos geométricos tienen un coste computacional bajo y son muy sencillos de implementar, pero son muy sensibles al ruido en la imagen de entrada. Además, la reconstrucción

de imágenes en base a momentos geométricos es bastante complicada. Aunque no tienen invarianza a transformaciones de manera nativa, pueden calcularse a través de los momentos invariantes de Hu. Considerando una imagen como una función discreta $f(x, y)$ con $x = 0, 1, \ldots, M$ y $y = 0, 1, \ldots, N$, la siguiente ecuación define los momentos geométricos m_{pq}:

$$m_{pq} = \sum_{x=0}^{M} \sum_{y=0}^{N} x^p y^q f(x, y) \tag{4.1.1}$$

Los momentos complejos son invariantes a la translación de manera nativa, pero igualmente tienen las mismas desventajas con respecto al ruido y a la reconstrucción de imágenes que los momentos geométricos. La siguiente ecuación define los momentos complejos:

$$m_{pq} = \sum_{x=0}^{M} \sum_{y=0}^{N} (x + yi)^p (x - yi)^q f(x, y) \tag{4.1.2}$$

Hay un grupo de momentos propuesto hace pocos años que usan una base ortogonal discreta. Son rápidos de implementar, presentan una tolerancia al ruido aceptable y son muy precisos para reconstrucción de imágenes. Entre ellos, los principales son los momentos de Zernike, los momentos de Tchebichef y los momentos de Hahn.

Los momentos de Zernike, desarrollados a partir de los polinomios de Zernike, tienen invarianza a la rotación de manera nativa y una excelente tolerancia al ruido. A través de la normalización de momentos se puede alcanzar la invarianza para la translación y el escalado. La ecuación que los define es

$$Z_{pq} = \frac{p+1}{\pi} \sum_{x} \sum_{y} f(x, y) W_{pq}(r, \theta) \tag{4.1.3}$$

donde

$$\begin{aligned} W_{pq}(r, \theta) &= R_{pq} e^{iq\theta}, \\ R_{pq}(r) &= \sum_{k=q, p-|k|=impar}^{p} \frac{(-1)^{(p-k)/2}((p+k)/2)!}{((p-k)/2)!((q+k)/2)!((k-q)/2)!} r^k \end{aligned} \tag{4.1.4}$$

Los momentos discretos de Tchebichef son ortogonales en el dominio del espacio de coordenadas de la imagen, de manera que eliminan completamente la necesidad de aproximaciones discretas en su implementación numérica. Esto los hace más precisos y reducen su complejidad computacional.

Los momentos de Hahn son una alternativa aún más precisa a los momentos de Tchebichef, contribuyendo a una mejor reconstrucción de imágenes sin penalizar la complejidad computacional. Varios autores han demostrado que realmente los momentos de Hahn son una generalización de los momentos de Tchebichef. La ecuación que los define es

$$d\lambda(t) = \sum_{k=0}^{N} w_k d(t - k) dt \tag{4.1.5}$$

donde

$$w_k = \binom{a+k}{k} \binom{b+N-k}{N-k}, \quad a > -1, b > -1 \tag{4.1.6}$$

Para una mayor comprensión de todos los momentos mencionados y el desarrollo matemático asociado a todas las ecuaciones, puede consultarse el trabajo de L. Kotoulas y I. Andreadis en [20].

4.1.2. Momento geométrico de una imagen

Para obtener información invariante acerca de los patrones hemos decidido utilizar los *momentos geométricos*. El concepto de momento geométrico en matemáticas evoluciona del concepto de momento en física. El momento n-ésimo de una función real en torno a un cierto valor a se expresa como:

$$m'_n = \int_{-\infty}^{\infty} (x-a)^n f(x) d(x) \tag{4.1.7}$$

Dependiendo del valor n los valores resultantes de cada momento tiene un significado particular. El momento de orden 1 en torno al valor 0 representa la esperanza estadística de la población, el momento de orden 2 representa la varianza, y así con muchos otros. Si calculamos los momentos en torno a la media μ obtenemos los *momentos centrales*, que se caracterizan por ser invariantes a la translación.

En el caso de las imágenes, el uso del concepto de momento interpreta los valores de intensidad de los pixels de una imagen como una función de densidad de probabilidad de una variable aleatoria en dos dimensiones. Las propiedades de esta variable aleatoria pueden ser descritas usando características estadísticas, a través de momentos de distinto orden. Asumiendo que tras las segmentación trabajamos con imágenes binarias, podemos redefinir la ecuación (4.1.7) para hallar el momento de orden $(p+q)$ de una imagen como:

$$m_{pq} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} i^p \, j^q \, f(i,j) \tag{4.1.8}$$

En esta ecuación, los valores i y j representan las coordenadas de los pixels de la región que estemos tratando. La independencia con respecto a la translación la podemos conseguir usando los momentos centrales μ_{pq}:

$$\mu_{pq} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} (i-x_c)^p \, (j-y_c)^q \, f(i,j) \tag{4.1.9}$$

Los valores x_c e y_c representan las coordenadas del centro de gravedad (baricentro) de la imagen, que puede obtenerse a través de las siguientes relaciones:

$$x_c = \frac{m_{10}}{m_{00}}, \quad y_c = \frac{m_{01}}{m_{00}} \tag{4.1.10}$$

Al tratar con imágenes binarias, el momento m_{00} representa el área de la región que estamos tratando. También podemos conseguir propiedades que sean invariantes al escalado usando los *momentos centrales normalizados* ϑ_{pq}:

$$\vartheta_{pq} = \frac{\mu_{pq}}{(\mu_{00})^\gamma}, \quad \gamma = \frac{p+q}{2} + 1 \tag{4.1.11}$$

Otras propiedades interesantes de los caracteres son la curtosis y la asimetría. La asimetría es un coeficiente que permite medir hacia qué lado de la media tienden los valores de una población, ya que una distribución de probabilidad no es perfectamente simétrica en torno a la media, ni siquiera en el caso de la distribución normal. Puede calcularse a partir de los momentos de orden 3: m_{03} y m_{30}. Por su parte, la curtosis κ es un coeficiente que mide lo "picudo" de una distribución de probabilidad en torno a la media. Puede calcularse a partir de los momentos de orden 4: m_{04} y m_{40}. En la tabla 4.2 aparecen resumidos los principales momentos y su significado.

Parámetro	Función
m_{00}	Área de la región de pixels.
m_{10}/m_{00}	Coordinada en el eje X del baricentro.
m_{01}/m_{00}	Coordinada en el eje Y del baricentro.
m_{20}, m_{02}	Varianza.
m_{30}, m_{03}	Asimetría de la región.
m_{40}, m_{04}	Curtosis.

Tabla 4.2: Significado de los momentos de una imagen.

4.2. Clasificación estadística

Los métodos de clasificación estadísticos utilizan como parámetro de entrada un vector de características X, que representa el patrón en cuestión en un espacio $\mathbb{R}^n$. Si suponemos que el conjunto de clases Ω divide el espacio n-dimensional en c partes, se pueden establecer fronteras de decisión que separen una clase de otra, como puede verse en la figura 4.2.

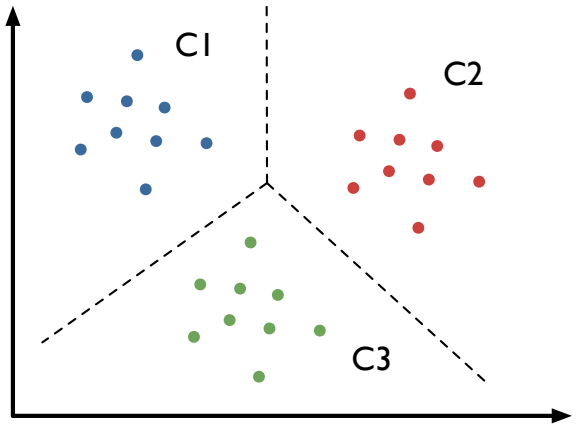


Figura 4.2: Espacio de características separado en c regiones. Cada región agrupa los patrones de la misma clase.

El objetivo de la clasificación estadística es encontrar un conjunto de funciones para obtener

dichas fronteras, conocidas como funciones discriminantes. La naturaleza de estas funciones define qué alternativa tomamos al problema:

- Si se considera un espacio de representación de naturaleza estadística, donde las distribuciones de las clases son conocidas u obtenidas por algún método, el problema de clasificación es de naturaleza paramétrica, y las fronteras se obtendrán en base a funcionales estadísticos.
- Si, por el contrario, no se considera dicha naturaleza estadística, el problema se plantea como uno de decisión geométrica, en el que las fronteras de decisión se obtienen en base a funcionales deterministas no paramétricos.

NessieOcr toma la aproximación no paramétrica y centra su diseño en un clasificador por funciones de distancia, ya que constituye un modelo simple e intuitivo. Si un vector de características define un punto en el espacio de características, la proximidad entre esos puntos es una manera obvia de determinar la pertenencia a una clase o a otra. Para obtener resultados satisfactorios las clases deben ser lo más compactas posible y estar bien separadas entre sí.

4.2.1. Funciones de distancia

Las funciones de distancia asignan un valor numérico a la noción de cercanía o lejanía entre dos formas representadas por vectores de características, de manera que dos objetos muy parecidos —muy cercanos— reflejan un valor pequeño del funcional de distancia. Las condiciones para considerar una función D como distancia ya fueron definidas en la sección 3.1.4, como método para establecer relaciones de localidad entre los pixels de una imagen. La función de distancia más conocida es la *distancia Euclídea*, que corresponde a la generalización a n dimensiones de la distancia entre dos puntos en un plano:

$$D_2(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{4.2.1}$$

La métrica Euclídea es invariante a rotaciones y translaciones, lo cual es una ventaja dado que si los recortes de prensa proceden de periódicos escaneados la imagen puede estar rotada con respecto al original. Otra función muy usada es la *distancia de Mahalanobis*, que tiene en cuenta la correlación entra las variables:

$$D_\Sigma(X,Y) = \sqrt{(X-Y)^T\Sigma^{-1}(X-Y)} \tag{4.2.2}$$

En la ecuación (4.2.2), Σ representa la matriz de covarianzas de las variables en X e Y, y el exponente T indica la operación vectorial de transposición. La distancia de Mahalanobis tiene la propiedad de que es invariante a cualquier tipo de transformación lineal no singular de las variables sobre todas las muestras.

4.2.2. Reglas de decisión

Una vez definidas las funciones de distancia es necesario establecer las reglas de decisión para realizar la clasificación. Una opción es utilizar el criterio de *distancia mínima*, en el que

cada categoría tiene un vector prototipo Z. La clasificación de un vector incógnita X en una clase se obtiene eligiendo el valor mínimo de distancia con el prototipo Z:

$$X \in \Omega_i \quad \text{si} \quad D(X, Z_i) = \min_{\forall i=1,2,...,c} \{D(X, Z_i)\} \tag{4.2.3}$$

El criterio de distancia mínima puede generalizarse para permitir más de un prototipo por cada clase, lo que se conoce como el criterio del *vecino más próximo* (NN, del inglés "nearest neighbour"). Ahora tendremos para cada clase de Ω un conjunto de prototipos $\{Z_1, Z_2, ...Z_n\}$, de manera que Z_i^j representa el prototipo j-ésimo de la clase i-ésima. Se dice que un vector incógnita X se clasifica según el vecino más próximo si en la clase i-ésima existe al menos un prototipo Z_i^p que sea, dentro del conjunto de los prototipos de todas las clases, el más próximo a la muestra:

$$X \in \Omega_i \quad \text{si} \quad D(X, Z_i^p) = \min_{\forall i=1,2,...,c \; \forall j=1,2,...N_i} \{D(X, Z_i^j)\} \tag{4.2.4}$$

El criterio de clasificación por el vecino más próximo puede también modificarse en el sentido de que la regla no suministre el prototipo más cercano al vector incógnita, sino el conjunto de prototipos más cercanos. Es lo que se conoce como el criterio de los *k-vecinos más próximos* (KNN). Según este método, se examinan k prototipos por cada clase, y se obtiene la categoría a la que pertenece la muestra por un criterio de mayoría entre todos los resultados.

Los métodos de vecindad descritos presentan el inconveniente de tener una estructura de clasificación de naturaleza exhaustiva. Por cada nuevo patrón hay que calcular su distancia con todo el conjunto de prototipos de cada clase. Si el número de muestras de aprendizaje es muy elevado el costo computacional en la toma de la decisión también lo es.

4.2.3. Alternativas a las funciones de distancia

NessieOcr está diseñado por defecto con el método de clasificación KNN, ya que es sencillo de implementar y ofrece una herramienta de referencia para implementar otros métodos a posteriori. Pero dado que el método es caro en cuanto a recursos de memoria y tiempo, es conveniente plantear otros criterios de decisión para el clasificador.

Una alternativa que desde hace relativamente poco tiempo ha conseguido una gran aceptación en la comunidad científica son las *máquinas de vectores soporte* (SVM, del ingles "Support Vector Machines"). Una SVM es un clasificador lineal de dos clases formulado a partir de una combinación ponderada de una serie de funciones kernel. Cada función kernel representa el producto interior de dos vectores en un espacio de características, lineal o no lineal. Se considera que utilizando esta aproximación se puede separar perfectamente todas las clases del espacio de clases. Sin embargo, la implementación para un reconocedor de caracteres es compleja, ya que habría que construir c máquinas para clasificar un carácter, siendo c el número de clases distintas.

Existe aún otra regla de decisión bastante conocida, basada en *redes neuronales*. Una red neuronal artificial permite modelar una función matemática compleja que a priori no puede ser programada, pero de la que se dispone conjunto básico de ejemplos de entrada. Las redes neuronales artificiales tienen la habilidad de aprender, crea su propia representación de la información en su interior, tienen una alta tolerancia a fallos y pueden manejar cambios no importantes en la información de entrada, como señales con ruido.

El diseño de NessieOcr, como veremos en posteriores capítulos, se ha planteado de forma que pueda aplicarse cualquiera de estos algoritmos de clasificación sin afectar al resto de etapas.

Aprovechando la modularidad de la biblioteca, un método puede ser reemplazado por otro en cualquier momento.

4.2.4. Estrategias de aprendizaje

Cualquiera de las reglas de decisión mencionadas necesitan de una etapa de aprendizaje o entrenamiento para disponer de una información mínima sobre la distribución de los patrones en sus clases. Por ejemplo, el método KNN necesita un conjunto mínimo de prototipos de cada clase, mientras que las redes neuronales necesitan un conjunto de muestras previamente clasificadas para aprender qué características pertenecen a cada clase.

De todos los enfoques de entrenamiento posibles podríamos quedarnos con dos. El primero de ellos consiste en realizar un aprendizaje por fuerza bruta, en el que suministramos al clasificador un conjunto de patrones extraído de una imagen sin realizar ningún tipo de organización. Este enfoque es útil para etapas iniciales, en los que necesitamos una implementación sencilla para depurar el funcionamiento del método. Sin embargo no garantiza que todas las clases parten de las mismas condiciones. Si hay más muestras de una clase que de otra obtendremos un buen clasificador para ciertos patrones, pero mediocre para otros.

Una mejor estrategia consiste en organizar los patrones de aprendizaje de manera equitativa, recopilando el mismo número de muestras de cada clase y entrenando con cada clase el mismo número de veces. De esta forma no se beneficia el reconocimiento de unos patrones frente a otros, y el clasificador estará mejor preparado.

Parte II

Diseño del software e implementación

5 Arquitectura del software

El análisis de requisitos visto en el capítulo 2 junto con el diseño del software que vamos a ver en este, define el modelo arquitectónico de la aplicación. Mientras que el primero se centra en cuáles son las necesidades del usuario y la funcionalidad de la aplicación, el diseño del software define cómo se organizan las clases y sus métodos para implementar dicha funcionalidad.

En este capítulo describimos la arquitectura del software de NessieOcr, basándonos en los casos de usos del modelo de análisis. El diseño de la arquitectura se divide en dos partes. El primer componente es el *modelo de datos*, que define las entidades de información que maneja la aplicación: los datos de entrada, los datos de salida y los datos que se generan internamente. El segundo componente es el *modelo de software*, que define las clases y servicios necesarios para obtener la funcionalidad deseada.

Antes de continuar advertimos al lector que en todo este capítulo los diagramas presentados estarán en inglés, para mantener la coherencia con la metodología de implementación. El motivo de esto es la posibilidad de que en un futuro NessieOcr se publique para que la comunidad de desarrollo de software libre tenga acceso a ella.

5.1. Modelo de datos

El modelo de datos representa la información que maneja el software y cómo se relacionan entre ellas las entidades que representan dicha información. En el caso de NessieOcr, este modelo describe las estructuras de datos de entrada y salida al subsistema, junto con las estructuras internas necesarias para implementar los algoritmos vistos en capítulos anteriores.

Cada caso de uso del modelo de análisis recibe unos datos de entrada y produce un resultado. Los casos de uso "Reconocer el texto" y "Entrenar el reconocedor" reciben como entrada una imagen de la página de un periódico, con las coordenadas del recorte de prensa que hay que procesar. El caso de uso "Entrenar el clasificador" recibe además un texto de referencia con el que comparar el resultado de la clasificación. A partir de la imagen de entrada se genera un texto en el caso de uso "Reconocer el texto", mientras que "Entrenar el reconocedor" no produce ningún resultado sino que sólo modifica los parámetros internos de la etapa de clasificación. Por su parte, el caso de uso "Obtener estadísticas" no recibe datos de entrada, pero da como resultado las estadísticas del reconocimiento como complemento a los otros dos casos de usos (véase la sección 2.4.3).

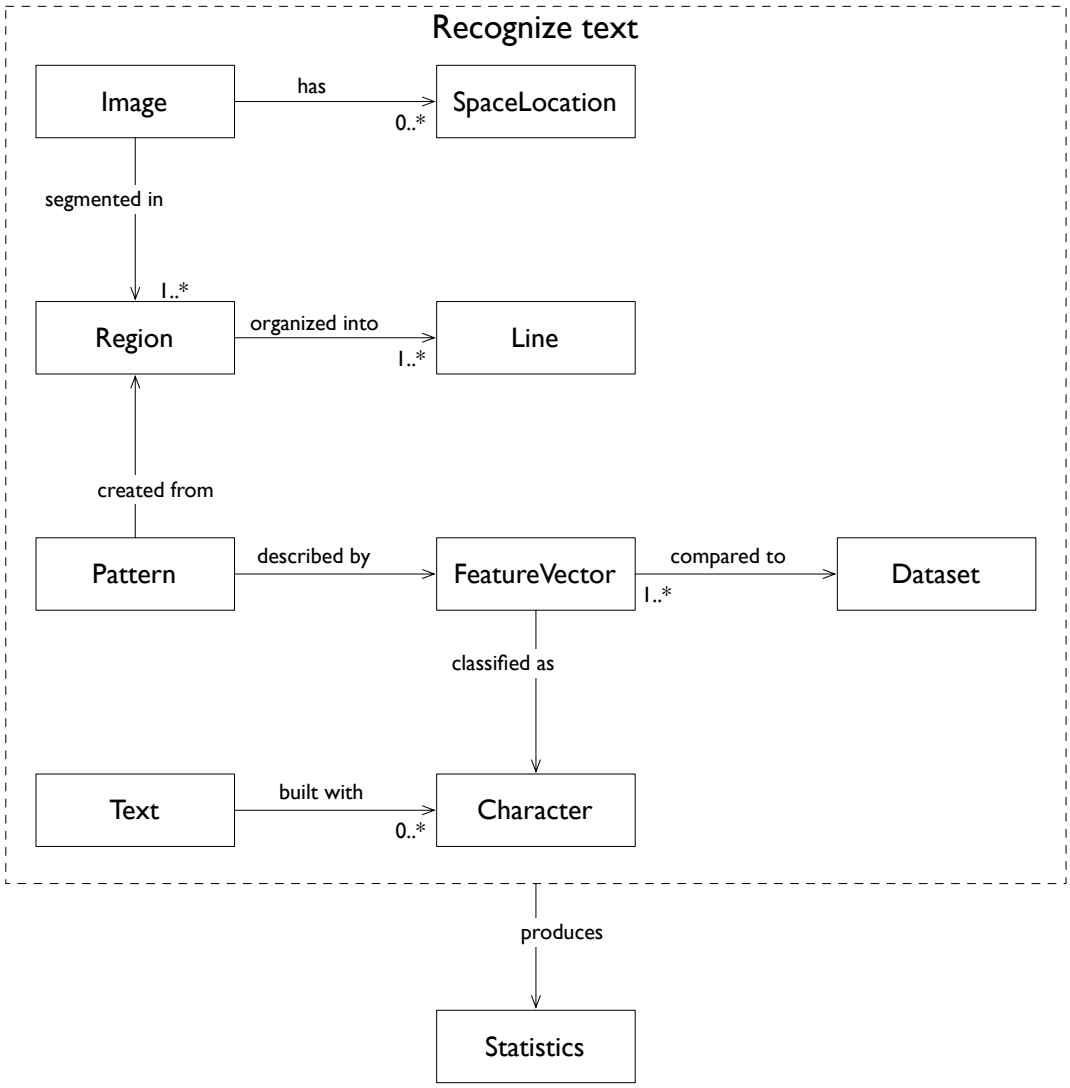


Figura 5.1: Modelo de datos de NessieOcr.

Para representar el modelo de datos completo vamos a tomar como referencia el caso de uso "Reconocer el texto", que es el más completo de todos. En él observamos cuatro tareas que se corresponden con las etapas de un sistema de reconocimiento de patrones: preprocesamiento, extracción de características, clasificación y postprocesamiento. En los capítulos 3 y 4 vimos que cada una de ellas usa distintas estructuras de datos para implementar los algoritmos. La relación entre todas las estructuras de datos necesarias está reflejada en el diagrama de la figura 5.1.

El preprocesamiento extrae una lista de regiones (`Region`) de una imagen de entrada (`Image`) y las transforma en un conjunto de patrones (`Pattern`) . La lista de regiones se organiza en líneas (`Line`) para poder localizar los objetos que formaban un carácter compuesto. Por otro lado también se obtiene una lista de índices (`SpaceLocation`) para insertar espacios en blanco al construir el texto final.

Por su parte, la etapa de extracción de características y clasificación necesita un conjunto de vectores de características (`FeatureVector`) que compara con un conjunto de prototipos

(`Dataset`) . Mediante un método de clasificación concreto asociamos cada vector a una clase, para a continuación construir el texto resultante (`Text`) concatenando cada carácter (`Character`) reconocido. Al finalizar el caso de uso podemos obtener de manera adicional las estadísticas (`Statistics`) del proceso completo.

5.2. Modelo de software

La arquitectura de NessieOcr sigue un modelo *cliente-servidor*, en el que una aplicación externa —el controlador— solicita a través de la interfaz de la biblioteca la realización de una serie de operaciones —reconocimiento de caracteres. En nuestro caso, los servicios que presta NessieOcr se corresponden exactamente con los casos de uso del modelo de análisis.

El diseño de cada caso de uso conlleva la definición de un conjunto de clases, las relaciones entre ellas y el flujo de ejecución adecuado para llevar a cabo el servicio. Al igual que al describir el modelo de datos, vamos a tomar como referencia el caso de uso "Reconocer el texto" para organizar el software en paquetes, donde cada paquete agrupa las clases pertenecientes a una tarea de este caso de uso. Los cuatro paquetes que forman la arquitectura de NessieOcr están en la figura 5.2.

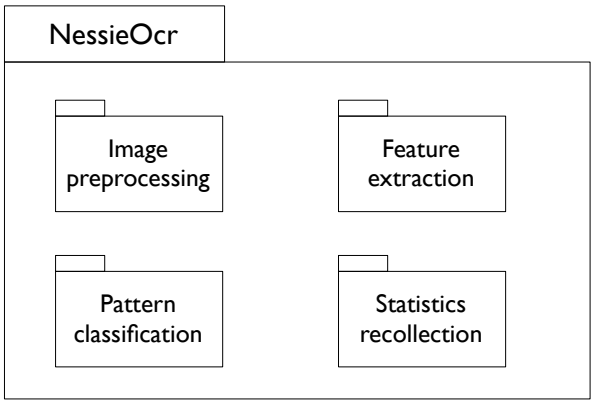


Figura 5.2: Organización en paquetes de la arquitectura de NessieOcr.

Cada paquete contiene una clase principal como interfaz de sus servicios, y clases adicionales que completan la funcionalidad de la tarea. De este modo, la interfaz del paquete `Image preprocessing` es la clase `Preprocessor` , la interfaz del paquete `Feature extraction` es la clase `FeatureExtractor` y la interfaz del paquete `Pattern classification` es la clase `Classifier` .

La tarea de postprocesamiento no tiene un paquete propio porque las transformaciones que hay que hacer sobre el texto son simples y las puede realizar el controlador manipulando directamente el texto resultante.

5.2.1. Interfaz de acceso a los servicios

Cuando un software cliente hace uso de una biblioteca de funciones tiene la intención de delegar en ella tareas de las que no necesita ser responsable. Por ello, la interfaz de NessieOcr debe facilitar lo máximo posible el acceso a sus servicios, en lugar de complicar su uso con detalles internos de la biblioteca.

NessieOcr da dos opciones a sus clientes. En primer lugar pone a su disposición todas las clases que controlan el manejo de un sistema de reconocimiento de caracteres, desde el preprocesamiento hasta la clasificación de caracteres. En este caso, es responsabilidad del programador el uso que se haga de ellas. Estas clases las veremos con más detalle en en el resto de la sección.

Hemos creído más conveniente diseñar una interfaz más amigable, para que el funcionamiento del reconocedor sea totalmente transparente. Mediante la aplicación del patrón de diseño *Facade*, la interfaz de NessieOcr queda reducida a la clase `NessieOcr` , que ofrece todas las operaciones básicas y más comunes. La relación de esta interfaz con el resto de paquetes puede verse en el diagrama de la figura 5.3.

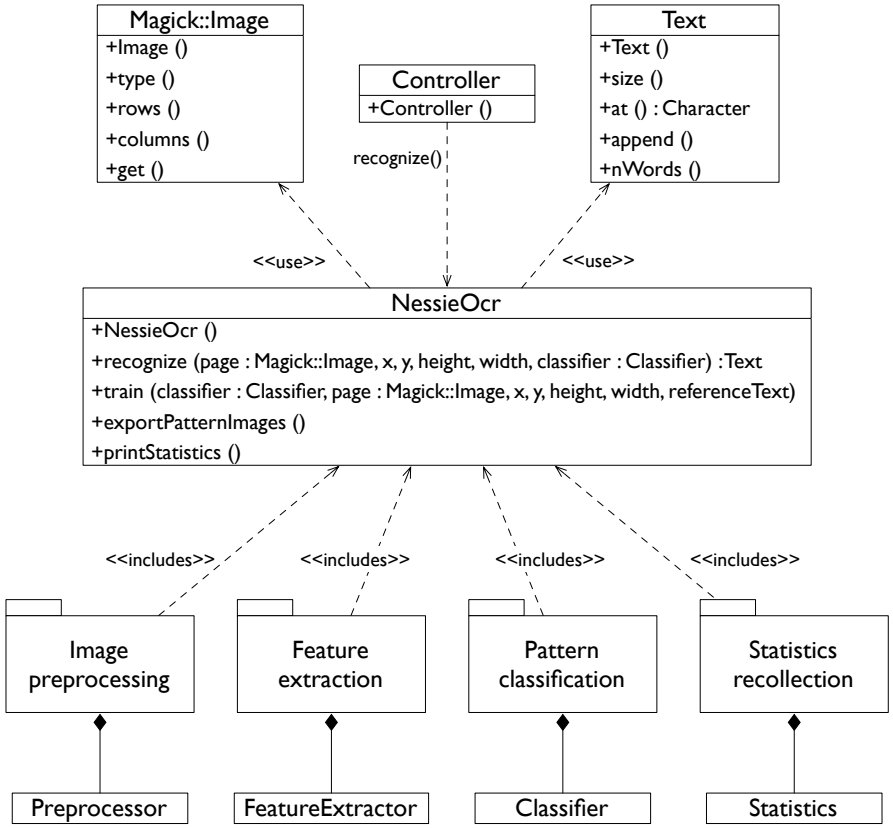


Figura 5.3: Interfaz de acceso a los servicios de NessieOcr. A través del patrón Facade se pueden agrupar los servicios más comunes en una única clase que sirve de interfaz para los clientes.

5.2.2. Diseño de la tarea "Preprocesar imagen"

El paquete `Image preprocessing` contiene la arquitectura de la tarea "Preprocesar imagen". En el diagrama de clases de la figura 5.4 podemos ver cómo se relacionan las clases de dicho paquete, donde la clase `Preprocessor` actúa como interfaz.

A través de los métodos de `Preprocessor` realizamos la eliminación de ruido y la binarización de la imagen, la extracción de las regiones y la construcción de los patrones. A medida que se ejecutan estos métodos se almacenan las estadísticas generadas en la clase `PreprocessorStatistics` , pudiendo ser solicitadas en cualquier momento.

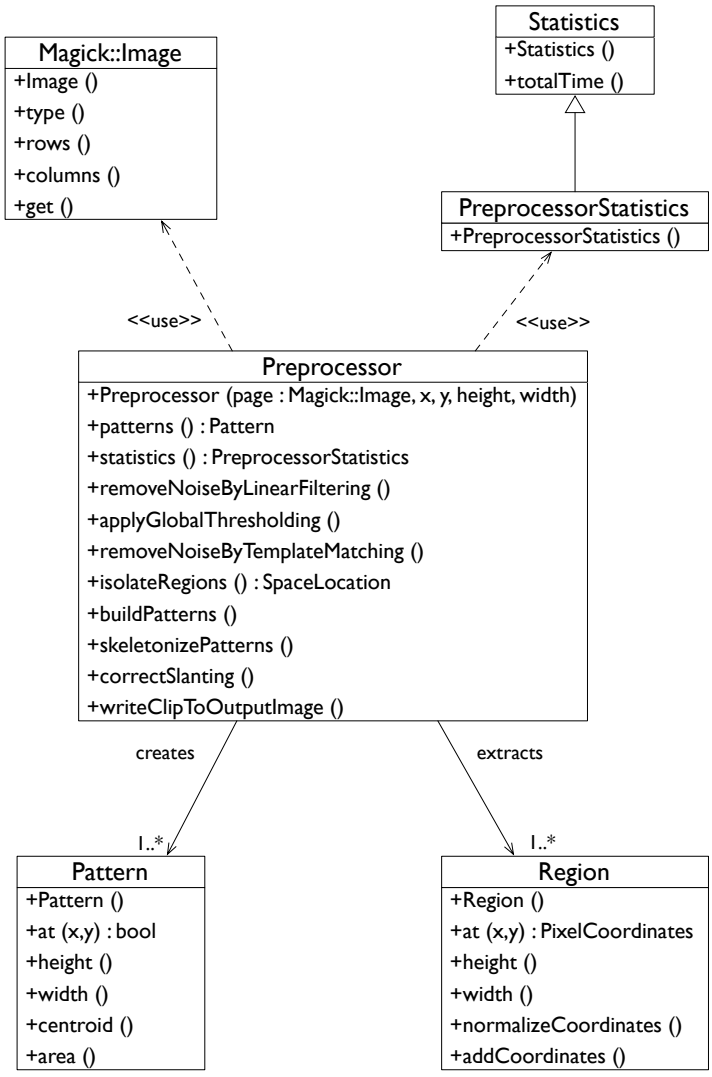


Figura 5.4: Diagrama de clases de la tarea "Preprocesar imagen".

El orden adecuado para ejecutar los métodos de la clase `Preprocessor` lo tenemos en el diagrama de secuencia de la figura 5.5, aunque en algunas circunstancias no es necesario ejecutar todos los pasos. Por ejemplo, si estamos tratando con periódicos en formato PDF generados de manera vectorial no tiene sentido aplicar los métodos de eliminación de ruido.

5.2.3. Diseño de la tarea "Extraer características"

El paquete `Feature extraction` contiene la arquitectura de la tarea "Extraer características". En el diagrama de clases de la figura 5.6 podemos ver la relación entre las clases de dicho paquete. La clase `FeatureExtractor` actúa como interfaz, y a través del método `computeMoments()` realizamos el cálculo de los momentos de cada patrón creado en la etapa de preprocesamiento. Las estadísticas generadas en el uso de este paquete quedan almacenadas en la clase `FeatureExtractorStatistics` .

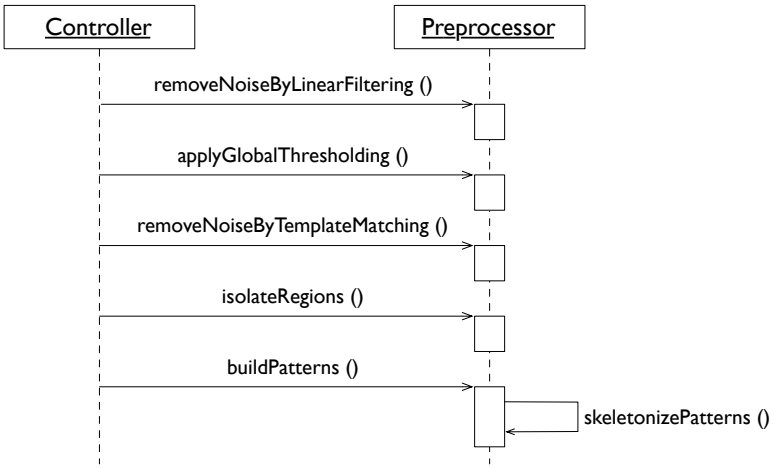


Figura 5.5: Diagrama de secuencia de la tarea "Preprocesar imagen".

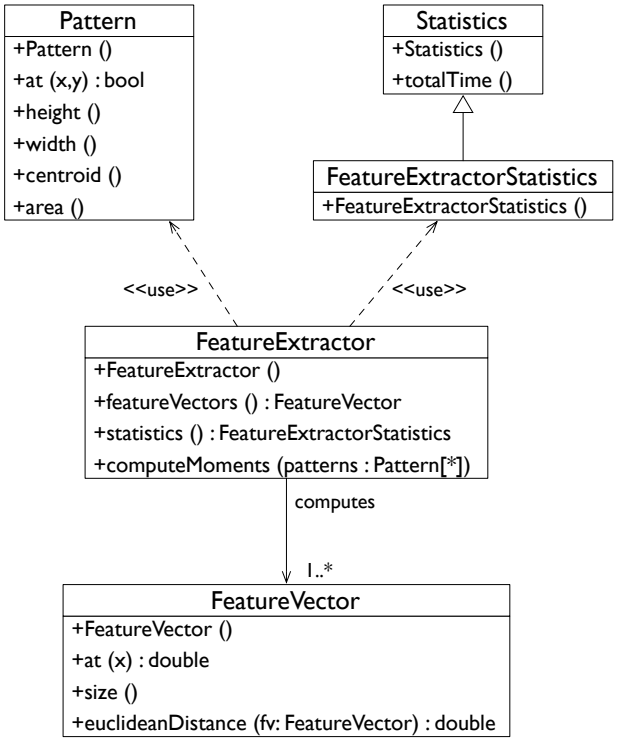


Figura 5.6: Diagrama de clases de la tarea "Extraer características".

5.2.4. Diseño de la tarea "Clasificar patrones"

El paquete `Pattern classification` contiene la arquitectura de la tarea "Clasificar patrones". En este caso queremos plantear el diseño de manera que en un futuro sea posible cambiar un método de clasificación por otro. Para ello vamos a utilizar el patrón de diseño *Strategy*, con el que independizamos la implementación de un método sin que esto afecte al resto de los elementos de la arquitectura.

El primer paso en la aplicación del patrón consiste en crear una clase abstracta `ClassificationAlgorithm` . Esta clase declara los métodos `train` y `classify` como virtuales. A partir de ella podemos crear tantas clases hija como algoritmos necesitemos. El diagrama de clases de la figura 5.7 muestra este caso de herencia simple, aunque realmente sólo se ha implementado la clase `KnnClassificationAlgorithm` .

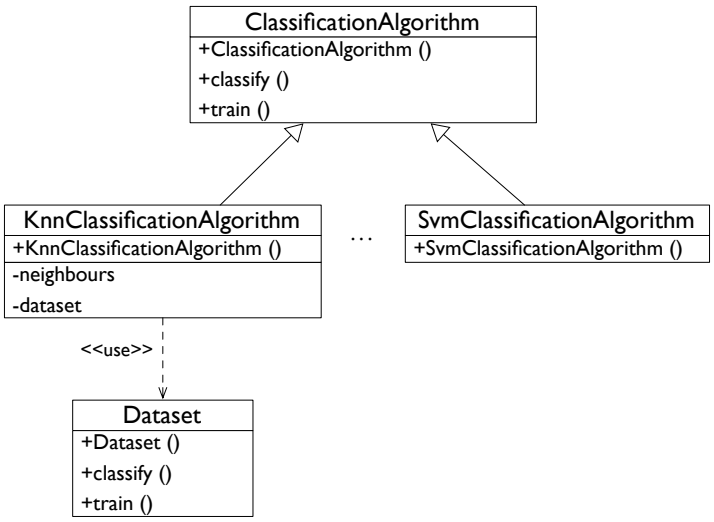


Figura 5.7: Diagrama de la jerarquía de clases `ClassificationAlgorithm` .

A continuación se crea una clase abstracta `Classifier` . Esta clase declara los métodos `performClassification` y `performTraining` como virtuales, y el atributo protegido `ClassificationAlgorithm` . A continuación definimos tantas clases herederas de Classifier como clasificadores distintos tengamos, tal y como ilustra el diagrama de la figura 5.8. Al crear un objeto se deberá especificar qué algoritmo se desea usar con el clasificador. De esta forma conseguimos tener diferentes clasificadores, y por cada clasificador diferentes algoritmos.

Un cliente de NessieOcr tan sólo deberá preocuparse de invocar el método `performClassification` , cuya implementación consiste en invocar el método `classify` del objeto `ClassificationAlgorithm` . Como resultado obtendrá un conjunto de caracteres, que puede utilizar junto con la información del resto de etapas para construir un objeto `Text` .

Para el caso concreto del clasificador KNN hemos diseñado la clase `Dataset` para que sea abstracta. Así también podemos elegir diferentes implementaciones de un "dataset" sin que esto afecte a la clase KnnClassificationAlgorithm. Por ejemplo, podemos tener un "dataset" almacenado en una base de datos o en fichero en texto plano. El diagrama de la figura 5.9 muestra un ejemplo de esta situación, y efectivamente el código fuente dispone de las tres clases representadas para manejar un dataset.

5.2.5. Diseño del caso de uso "Obtener estadísticas"

El paquete `Statistics recollection` contiene la arquitectura del caso de uso "Obtener características". En el diagrama de clases de la figura 5.10 podemos ver la relación entre las clases de dicho paquete. La clase `Statistics` actúa como interfaz, siendo una clase abstracta que permite declarar clases concretas para cada paquete de NessieOcr. De hecho, en los diagramas vistos en esta sección se han declarado tres clases que heredan de `Statistics` .

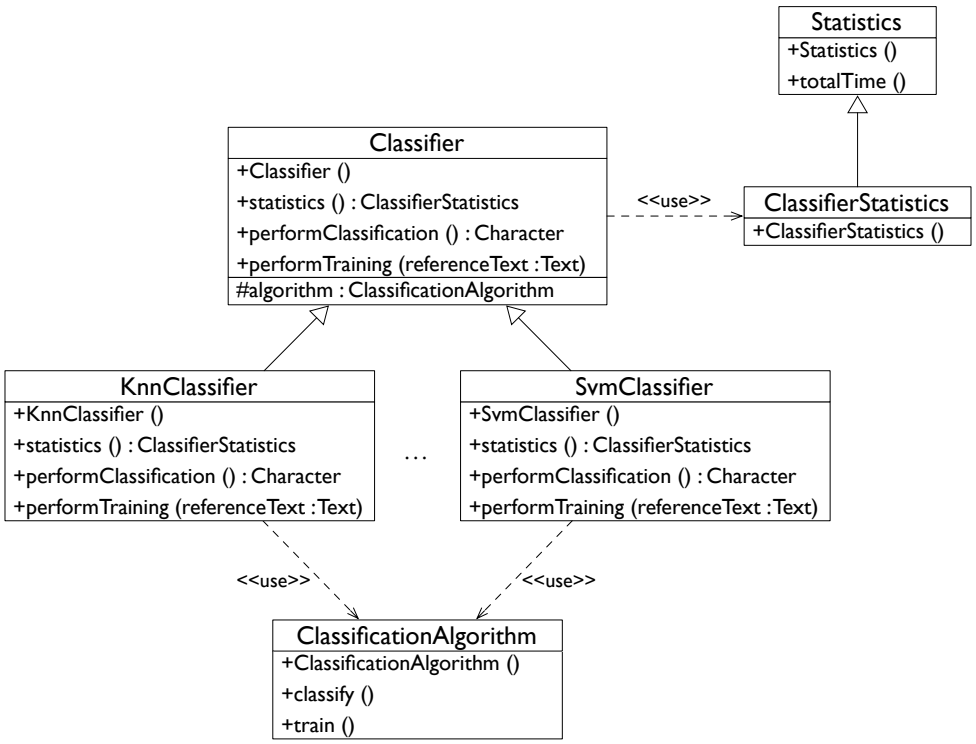


Figura 5.8: Diagrama de clases de la tarea "Clasificar patrones".

5.2.6. Flujo de ejecución

En el caso de que no se desee utilizar la clase `NessieOcr` para hacer uso de los servicios de la biblioteca, un desarrollador puede tomar como referencia el diagrama de secuencia de la figura 5.11 para saber cómo manejar las clases. En él podemos observar que el software cliente debe definir al menos un objeto de la interfaz de cada paquete: un objeto `Preprocessor` , un objeto `FeatureExtractor` y un objeto `Classifier` . Tras ejecutar los métodos en el orden sugerido podemos construir el texto resultante usando un objeto `Text` con la lista de caracteres que devuelve el método `performClassification` .

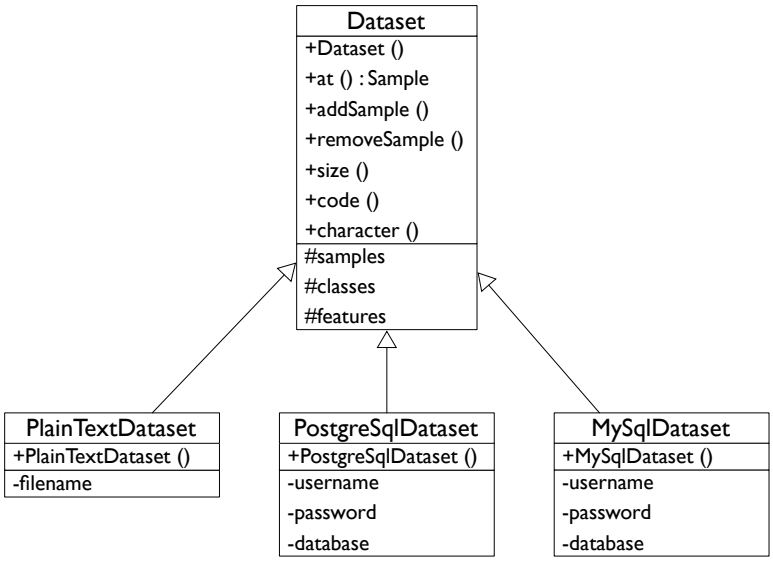


Figura 5.9: Diagrama de la jerarquía de clases `Dataset` .

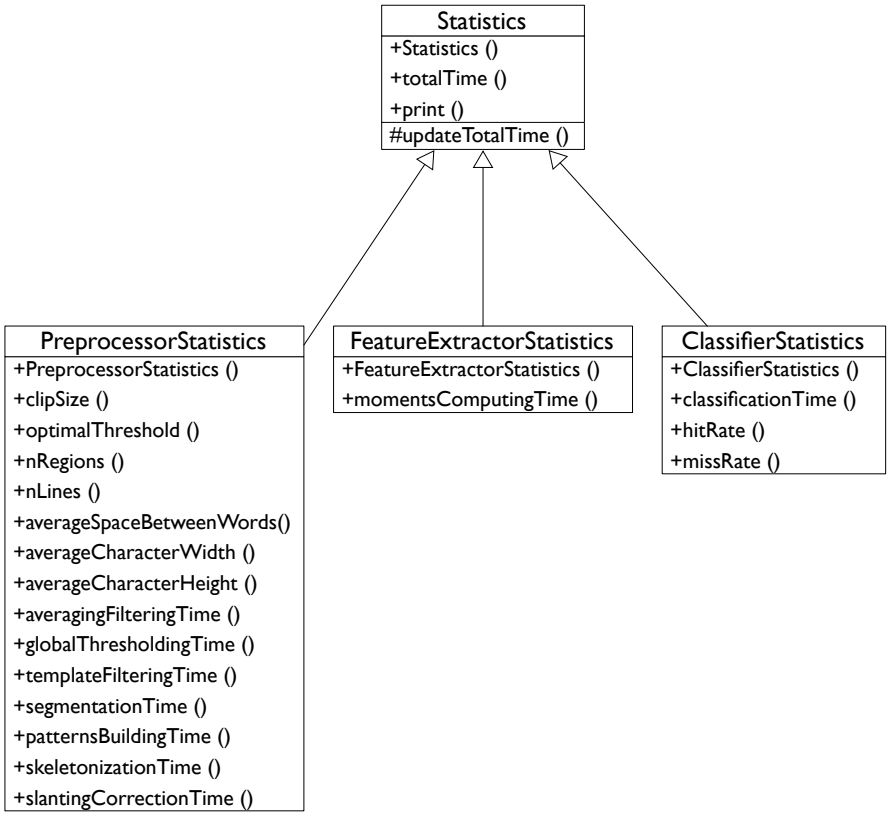


Figura 5.10: Diagrama de clases del caso de uso "Obtener estadísticas".

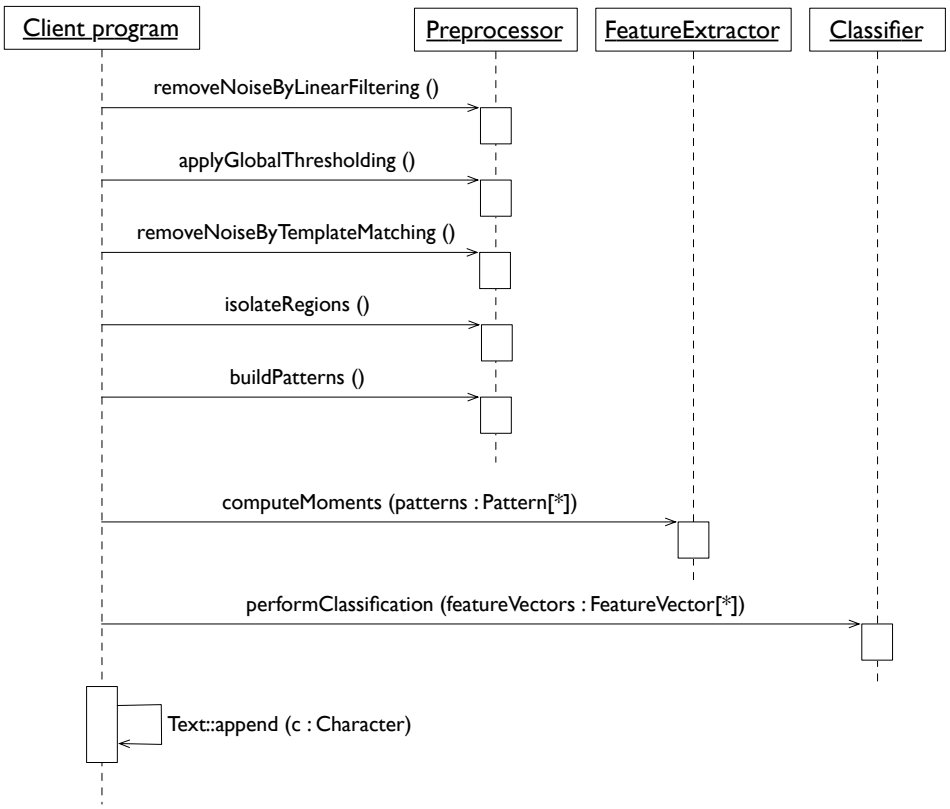


Figura 5.11: Diagrama de secuencia de un flujo de ejecución de referencia de NessieOcr.

6 Implementación y resultados

La rapidez con la que tienen que ejecutarse los métodos de NessieOcr es un factor crítico a tener en cuenta en la implementación. Por ello muchas veces no basta con hacer una traducción directa del algoritmo a código en C++, sino que es necesario optar por alternativas. La elección de las estructuras de datos o el uso de API para delegar tareas son ejemplos de particularidades asociadas al lenguaje de programación que merecen la pena ser comentadas. A lo largo de este capítulo presentamos algunos detalles de implementación de NessieOcr, la complejidad computacional de los principales métodos y los resultados de tiempo de ejecución obtenidos tras realizar una serie de pruebas.

6.1. La clase Preprocessor

El diseño de la tarea "Preprocesamiento de imágenes" coloca a la clase `Preprocessor` como interfaz para acceder a todas las operaciones necesarias para trabajar con la imagen que recibe NessieOcr. Al mismo tiempo es la clase más compleja con diferencia, ya que agrupa los principales algoritmos de manipulación de imágenes, segmentación y construcción de patrones. Es por ello que la creemos merecedora de resaltar algunos de sus aspectos, dejando de lado por un momento el resto de las clases que forman el código completo.

Aunque hemos delegado en Magick++ el proceso de carga de imágenes desde disco, hemos querido ir un paso más allá y hacer que `Preprocessor` también sea independiente de la API utilizada. Para ello el constructor de `Preprocessor` recibe como parámetro de entrada un objeto `Magick::Image`, cuya información se copia en un objeto `vector` de la STL de C++. De esta manera, Preprocessor ya no trabaja directamente con la API sino con un vector cuyos elementos representan el nivel de gris de los pixels de la imagen. Si fuera necesario utilizar una API distinta a Magick++ para cargar imágenes, tan sólo habría que añadir un nuevo constructor con los parámetros de entrada apropiados.

Como dijimos, la clase `Preprocessor` es con diferencia la más compleja de la arquitectura, ya que contiene muchos métodos que dependen unos de otros. A continuación vamos a comentar algunos detalles de implementación de los dos métodos más importantes.

6.1.1. El método isolateRegions

El algoritmo de segmentación de regiones por agregación de pixels (sección 3.3.1) está implementado en el método `isolateRegions`. Este método no es una traducción directa del algoritmo, sino que tiene en cuenta varios aspectos:

- Las regiones resultantes deben estar ordenadas con respecto a su posición en el texto, de forma que respeten el orden de lectura de izquierda a derecha y de arriba a abajo.
- Muchos de los caracteres del texto se forman uniendo dos o más regiones, tal como se explica en la sección 3.3.2.
- A medida que se reconocen caracteres hay que ir guardando la localización de los espacios que separan dos palabras.
- El tiempo de ejecución debe acotarse de forma que se respete el límite de tiempo por página impuesto a priori.

Previamente al proceso de segmentación generamos una lista de semillas a partir de las que se construyen las regiones. Como sabemos que la imagen está binarizada, esta lista se construye recorriendo la imagen una vez y guardando las coordenadas de los pixels con valor 1. A continuación se crea un *vector de visitados* del mismo tamaño que la imagen, y se inicializa al valor `false`. Luego definimos una región como un objeto de la clase `Region`, que guarda las coordenadas (i, j) de los pixels de la imagen que forman parte de la región.

La definición del algoritmo vista en la sección 3.3.1 implica tomar cada semilla e ir explorando sus vecinos, añadiendo a la región en construcción los pixels con valor 1. Esta operación realizada de manera recursiva es muy costosa, así que hemos decidido adoptar otra estrategia. El primer paso consiste en explorar sólo la vecindad inmediata de una semilla, marcando los pixels que se añaden a la región como visitados. A continuación recorremos la lista inicial de coordenadas de pixels que componen la región de manera iterativa, explorando en cada paso la vecindad inmediata del pixel que corresponda. Los nuevos pixels se añaden al final, lo que implica que la región se va construyendo a medida que la vamos explorando. Si en un paso llegamos al final de la lista de coordenadas quiere decir que la región ya está construida.

El incremento de velocidad en la ejecución del método con respecto a la versión recursiva reside fundamentalmente en la actualización correcta de la matriz de visitados, que evita explorar un pixel más de una vez, y en limitar la exploración a la vecindad inmediata de los pixels que ya forman parte de la región. Al final del proceso de exploración obtenemos una lista de objetos `Region`.

Los siguientes pasos del método son optimizaciones necesarias para reducir el tiempo de ejecución. Numerosas pruebas realizadas en una primera implementación demostraron que trabajar con la lista completa es muy costoso. Por ello, se añadieron estructuras de datos adicionales para manejarla. Tras obtener la lista de regiones buscamos un conjunto de delimitadores que separan los caracteres por líneas. Como muestra la figura 6.1, estos delimitadores se utilizan para organizar las regiones en filas, de forma que en sucesivos métodos se trabaje sólo con un conjunto de regiones reducido.

Una aplicación clara de organizar las regiones en líneas la vemos en la fusión de regiones que forman un único carácter. Si estamos buscando el acento de una vocal y trabajamos con la lista de regiones completa estaríamos buscando acentos en toda la imagen, cuando el acento que realmente nos interesa está en la misma línea de texto que la vocal.

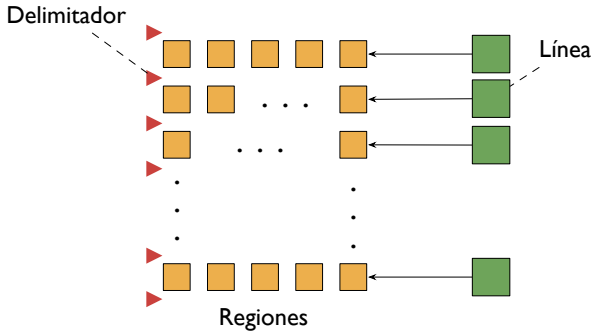


Figura 6.1: Organización de regiones por líneas en base a delimitadores.

La organización por líneas también es útil a la hora de encontrar los espacios que separan las palabras. De manera general esto se consigue buscando pares de caracteres entre los que hay un espacio en blanco mayor que el espacio medio que separa todos los caracteres. Si en lugar de buscar en toda la lista estos pares de caracteres limitamos la búsqueda a una línea, podemos reducir también el tiempo de ejecución.

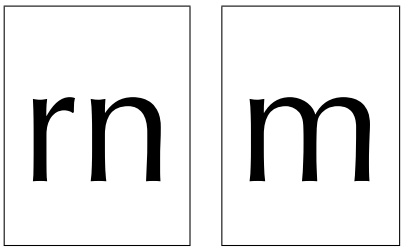
En la sección 3.3.1 explicamos que uno de los problemas con los que nos podemos encontrar es la aparición de caracteres que están unidos por alguna zona debido a la mala calidad de la imagen. Planteamos también que podíamos decidir entre implementar un método de separación de regiones o aumentar la calidad de la imagen para que la unión no se produjera. La primera opción la hemos descartado por las siguientes razones:

- Hasta que no se clasifica un patrón no es posible saber si la región encontrada es un carácter válido o no. Y dependiendo del método de clasificación empleado ni siquiera tenemos garantía de que esa detección pueda realizarse. Por tanto, no siempre es posible saber sobre qué patrón debemos aplicar el método de separación.

- Suponiendo que tras la clasificación seamos capaces de detectar cuándo un carácter es en realidad dos caracteres unidos, existen combinaciones de caracteres que por su naturaleza pueden llevar a confusión. En la figura 6.2 vemos un ejemplo de como la combinación "rn" es visualmente muy parecida a la letra "m".

- Los métodos de separación de regiones implican un alto coste en tiempo de ejecución, ya que cada separación implica construir dos patrones, calcular sus vectores de características, clasificarlos y observar si los resultados son aceptables o hay que volver a separar en otro punto distinto.

Por todas las consideraciones anteriores hemos decidido optar por la alternativa de aumentar la calidad de la imagen de entrada, imponiendo como precondición que todo recorte de prensa debe tener como mínimo una resolución de 600 dpi. Esto aumentará el tiempo de ejecución al comienzo de la etapa de preprocesamiento, por el gran volumen de datos a manejar, pero simplificará los métodos posteriores e incluso evitará tener que desarrollar métodos nuevos para resolver situaciones que sólo se darán de manera ocasional.



(a) Caracteres "rn" **(b)** Carácter "m"

Figura 6.2: Fenómeno de similaridad entre dos caracteres seguidos y un único carácter. De cara al proceso de clasificación, si los dos primeros caracteres se unen se reconocerá como la letra "m", y no es posible detectar que efectivamente se trata de dos caracteres unidos.

6.1.2. El método buildPatterns

El algoritmo de construcción de patrones (sección 3.4) está implementado en el método `buildPatterns`. El algoritmo implica la normalización de todas las regiones encontradas en la imagen, la creación de un patrón inicial y su esqueletización para reducir la información redundante. La normalización es una transformación complicada, ya que si recordamos lo visto en la sección 3.4.1, no sólo implica realizar un cambio en la relación de aspecto de la región sino aplicar un proceso de interpolación para preservar la conectividad entre pixels. En nuestro caso vamos a utilizar el método `sample` de la clase `Magick::Image`, que realiza una normalización de cada patrón en un tiempo despreciable con respecto al tiempo total del algoritmo. El único inconveniente de este estrategia es que hacemos que la clase `Preprocessor` sea dependiente de Magick++.

La normalización reduce la región a un plano estándar, pero no garantiza que esté alineada en el centro. Por tanto, es necesario desplazarla vertical y horizontalmente para hacer coincidir el centro de la región con el centro del plano. La operación es sencilla, ya que basta con tomar el pixel más al norte y al sur y calcular el desplazamiento a realizar para que ambos se encuentren a la misma distancia de los bordes. De manera análoga habría que proceder con los pixels más al este y al oeste. Al finalizar tendremos construido un patrón inicial que se almacena en la clase `Pattern`. A este patrón se le aplica a continuación el proceso de esqueletización, implementado en el método `skeletonizePatterns`.

Adicionalmente se podría añadir la corrección de inclinación para los caracteres en cursiva, pero tras realizar varias pruebas para comprobar su comportamiento hemos observado dos problemas. Primero nos encontramos con la dificultad de conocer a priori sobre qué patrones hay que aplicar la corrección, ya que realizarlo de manera incondicional deforma la silueta de caracteres normales. Y luego tenemos el problema de que un carácter en cursiva se convierta en otro al corregir la inclinación. Por todo esto, hemos decidido confiar en la generalidad de la clasificación para reconocer los caracteres en cursiva sin necesidad de aplicar ninguna transformación.

6.2. Bibliotecas auxiliares utilizadas

En la implementación de NessieOcr hemos utilizado otras bibliotecas de funciones aparte de Magick++, ya que nos pareció inteligente adoptar la filosofía de reutilización de código y evitar perder el tiempo en diseñar utilidades que ya otros programadores han implementado y comprobado de sobra. Mencionamos a continuación las más importantes.

6.2.1. Biblioteca Boost

Las bibliotecas *Boost* han sido uno de los recursos más importantes que hemos utilizado. Se trata de un conjunto de bibliotecas de funciones de código libre, desarrollada y mantenida por una extensa comunidad que se centra en producir código optimizado que sea compatible con la biblioteca estándar de C++. De hecho, muchas de las bibliotecas disponibles en Boost han sido incluidas en el "C++ Standards Committee's Library Technical Report" (TR1), y formarán parte del nuevo estándar de C++.

La biblioteca `timer` de Boost ha sido utilizada para monitorizar los tiempos de ejecución de los métodos más importantes de NessieOcr. Como muestra el código 6.1, un temporizador inicializado a cero se define a la entrada de cada método que se quiere monitorizar, y al final se registra el tiempo invertido en la ejecución invocando al método específico de la clase `Statistics` con la que se esté trabajando.

```
void Preprocessor::applyGlobalThresholding ()
{
boost::timer timer;
timer.restart();
...
statistics_.executionTime(timer.elapsed());
}
```

Algoritmo 6.1: Monitorización de tiempos con la clase `Boost::timer`.

Boost también se ha utilizado para postprocesar el texto resultante tras el reconocimiento, ya que el subsistema de clasificación de noticias lo necesita con unas características concretas. La biblioteca `regex` proporciona métodos para realizar operaciones de búsqueda y reemplazo de caracteres en un objeto `string` de C++, mediante la definición de expresiones regulares que pueden configurarse para responder a distintos motores. El código 6.2 muestra un ejemplo sobre cómo hemos usado esta biblioteca para eliminar varios caracteres de un texto.

```
std::string result;

// Remove non-alphanumeric characters
boost::regex pattern = "[\\?¿,;.:\\!+*/=<>'\\()\\{}\\[\\]|]+";
result.assign(regex_replace(result, pattern, ""));

// Remove trailing spaces
pattern = "\\s+";
result.assign(regex_replace(result, pattern, " "));
```

Algoritmo 6.2: Postprocesamiento de un texto con la clase `Boost::regex`.

6.2.2. Comunicación con MySQL y PostgreSQL

Hemos utilizado dos API para implementar las clases que heredan de la clase`Dataset` y que utilizan bases de datos como medio de almacenamiento. Por defecto NessieOcr trae dos clases de esta índole. Cada una de ellas está pensada como interfaz que encapsule el uso de las API que realizan el acceso a los motores de bases de datos que se desee usar. La clase `MySqlDataset` implementa un dataset a través de una base de datos MySQL, a la que accedemos con la biblioteca *MySQL++*. Esta biblioteca es una abstracción en C++ de la biblioteca de bajo nivel que proporcionan los desarrolladores de MySQL. NessieOcr también ofrece la clase `PostgreSqlDataset`, que implementa un dataset a través de una base de datos PostgreSQL. En este caso utilizamos *libpqxx* como soporte, que sirve como interfaz para la API que proporcionan los desarrolladores de PostgreSQL. Para conocer en más detalle su uso remitimos al lector a la documentación oficial que pueden encontrar en [37] y [18].

6.3. Complejidad computacional

En esta sección vamos a explicar la complejidad computacional de los principales algoritmos que componen NessieOcr. Para ello vamos a tener en cuenta el código implementado, no el algoritmo original en pseudolenguaje tal como se explicó en los capítulos 3 y 4. Para definir la complejidad vamos a utilizar la notación de Landau $O(g(x))$, que establece la cota superior asintótica $g(x)$ de una función $f(x)$ cuando x tiende a infinito. La complejidad computacional es un factor muy importante a tener en cuenta, dado que el tiempo de ejecución es una restricción impuesta en el análisis de requisitos. En la tabla 6.1 aparecen resumidas las complejidades de los algoritmos vistos en este capítulo. En todos los casos hemos contemplado el peor de los casos.

Algoritmo	Orden	Parámetro de referencia
Promediado en base a filtros	$O(n^2)$	n = tamaño de la imagen en pixels.
Cálculo del umbral óptimo	$O(n)$	n = tamaño de la imagen en pixels.
Binarización del espacio de color	$O(n)$	n = tamaño de la imagen en pixels.
Eliminación de ruido en base a plantillas	$O(n^2)$	n = tamaño de la imagen en pixels.
Segmentación de regiones por agregación	$O(n)$	n = tamaño de la imagen en pixels.
Organización de regiones en líneas	$O(l)$	l = tamaño de la lista de regiones.
Ordenación de regiones en una línea	$O(r^2)$	r = número de regiones en la línea.
Esqueletización	$O(p^2)$	p = tamaño del patrón en pixels.
Cálculo de los momentos normalizados	$O(p)$	p = tamaño del patrón en pixels.
Clasificación de un conjunto de patrones	$O(p{\cdot}d)$	p = tamaño del patrón en pixels, d = tamaño del dataset.

Tabla 6.1: Complejidad computacional de los principales métodos implementados en NessieOcr.

En primer lugar vamos a analizar los principales métodos de la etapa de preprocesamiento. En la implementación hemos escogido representar una imagen a través de un objeto `vector` de la STL de C++. El número de elementos del vector viene determinado por el producto del número de filas y columnas de la imagen, y cada elemento equivale al nivel de gris de un pixel de la imagen. Por tanto, todo algoritmo que recorra la imagen completa tiene como mínimo una

complejidad $O(n)$, siendo n el número de pixels. Si además se examina la vecindad de cada pixel la complejidad sube a $O(n^2)$.

Todos los métodos que trabajan con la imagen completa tienen el número de pixels n como parámetro de referencia para medir la complejidad. Los métodos de eliminación de ruido tienen complejidad $O(n^2)$, ya que examinan la vecindad de cada pixel. La binarización del espacio de color en base al umbral óptimo tiene complejidad $O(n)$ ya que accede a cada pixel una única vez para examinar y si es necesario modificar su valor.

El cálculo del *umbral óptimo* mediante el algoritmo de Otsu tiene dos análisis posibles. Si ya tenemos calculado el histograma de la imagen la complejidad es $O(k)$ constante, donde k es el número de niveles de gris de la imagen. Pero normalmente tenemos que calcular el histograma recorriendo la imagen, por lo que en la implementación final el algoritmo de Otsu también es de complejidad $O(n)$.

El método de *crecimiento de regiones* por agregación de pixels también tiene una complejidad $O(n)$, aunque pueda parecer que al explorar la vecindad de cada pixel debería ser mayor. La clave está en el uso de la matriz de pixels visitados, que en cada iteración del algoritmo evita examinar elementos que ya han sido descartados en iteraciones anteriores. Esto hace que cada pixel se examine una única vez.

Los métodos asociados a la segmentación de regiones tienen análisis distintos, ya que el parámetro de referencia ya no es el número de pixels de la imagen entera. La *organización de regiones por filas* tiene complejidad $O(l)$, siendo l el número de elementos de la lista de regiones obtenido en la segmentación. Esta complejidad refleja el hecho de que cada región es examinada una única vez para obtener sus coordenadas, extraerla de la lista y colocarla en su línea de texto correspondiente.

La *ordenación de regiones* dentro de una misma línea de texto tiene complejidad $O(r^2)$, siendo r el número de regiones por línea. Esta complejidad representa el peor caso del algoritmo de ordenación por inserción, que es el modelo de referencia para implementar la ordenación de regiones en la línea. En el mejor caso, este algoritmo presenta una complejidad $O(1)$, que implicaría que la lista ya está ordenada. La complejidad media también es $O(r^2)$, lo cual desaconseja el uso de este algoritmo para listas de gran tamaño. Sin embargo, este no es nuestro caso porque el número de elementos en una región es perfectamente abordable. Hay que tener en cuenta que el algoritmo de ordenación hay que ejecutarlo tantas veces como líneas de regiones se hayan encontrado.

El último método de preprocesamiento que nos queda por analizar es la *esqueletización*. En este punto ya tenemos un patrón creado con un tamaño estándar prefijado. Para realizar la esqueletización hay que explorar al menos una vez la vecindad de cada pixel, repitiendo el proceso completo tantas veces como sea necesario. Por tanto, podemos concluir que la complejidad computacional del método es $O(p^2)$, donde p es un valor constante que representa el número de pixels del plano estándar del patrón.

En la etapa de extracción de características sólo vamos a analizar el *cálculo de los momentos* normalizados de un patrón. Si recordamos la ecuación (4.1.11) introducida en la sección 4.1.2, para calcular el momento m_{pq} había que acceder al valor de cada pixel del patrón y ponderarlo con un factor que dependía de las coordenadas del pixel en cuestión y los valores p y q. Esto nos deja con una complejidad $O(p)$, siendo p el número de pixels del patrón. El cálculo se repite por cada momento distinto que queramos calcular para un patrón, para todos los patrones construidos.

Por último vamos a analizar el *algoritmo de clasificación KNN*. En nuestra implementación, el método calcula de manera iterativa la distancia de cada vector de características incógnita

con todas las muestras de entrenamiento almacenadas en un dataset. Es decir, que si tenemos v vectores incógnita y estamos tratando con un dataset que almacena d muestras, la complejidad de este algoritmo es $O(v \cdot d)$. Esto explica por qué el paradigma KNN no es recomendable para sistemas de tiempo real, ya que el tamaño del dataset condiciona enormemente la respuesta de la aplicación.

6.4. Tiempos de ejecución

Uno de los resultados más importantes que debemos obtener son los tiempos de ejecución de NessieOcr. A lo largo de todo el desarrollo —y también en este documento— hemos tenido en cuenta el límite de tiempo por página impuesto en los requisitos del producto (sección 2.2). Recordando dicha restricción, la suma del tiempo invertido en procesar todos los recortes de una página no podía sobrepasar los veinte segundos.

A medida que se ha implementado NessieOcr se han ido realizando diversas pruebas sobre el comportamiento de los algoritmos. Al finalizar la implementación se realizó una comprobación más exhaustiva en la que se detectaron partes del código que concentraban la mayor parte del tiempo de ejecución y generaban un "cuello de botella" que fue necesario optimizar. Los puntos críticos detectados tras las pruebas fueron cuatro:

- La creación de una estructura de datos intermedia para cargar el objeto `Magick::Image` de entrada.
- El algoritmo de fusión de caracteres compuestos por dos o más regiones.
- El algoritmo de construcción de patrones.
- El algoritmo de clasificación KNN.

El experimento consistió en ejecutar un programa de prueba que implementa las tareas del caso de uso "Reconocer el texto" para todos los recortes de una página de periódico de El País. La página almacenada en formato PNG ocupa aproximadamente 22.8MB y tiene dimensiones 12362×8931. El tiempo de ejecución se ha calculado promediando cien ejecuciones del programa, utilizando una máquina con un procesador de cuatro núcleos y 4 GB de memoria RAM. La siguiente tabla resume las condiciones del experimento:

Tras comprobar que el tiempo de ejecución se disparaba en torno a los 70 segundos se decidió acometer una revisión de los métodos, concentrándonos en aquellas partes que acumulaban la mayor carga. Los resultados de complejidad computacional y la implementación obtenidos tras la fase de optimización han sido los presentados en las secciones anteriores de este capítulo.

Una vez obtenida una versión optimizada de NessieOcr hemos repetido la ejecución del programa de prueba para comprobar las mejoras obtenidas. De manera general hemos conseguido reducir el tiempo de ejecución hasta aproximadamente 27 segundos, lo que supone una mejora del 63 %. Los tiempos concretos de cada etapa y sus métodos aparecen resumidos en la tabla 6.3, mientras que la figura 6.3 muestra la mejora realizada por cada etapa.

Como puede verse en los resultados presentados no hemos conseguido alcanzar el objetivo de procesar una página en 20 segundos como máximo. La causa principal de esta situación es el hecho de tener que manejar imágenes en resoluciones muy altas para evitar la unión de caracteres que provocarían la generación de patrones incorrectos. De hecho, en la distribución del tiempo por etapas de la figura 6.4 se aprecia que el preprocesamiento es la tarea que mayor carga concentra, ya que es cuando se trabaja con mayor cantidad de información.

Elemento	Valor
Tipografía	El País
Formato de imagen	PNG
Resolución	800dpi
Tamaño	22.8 MB
Dimensiones	12362×8931
Procesador	Intel Core 2 Quad, 2.83GHz
Memoria RAM	4GB
Caché L1	4×32KB
Caché L2	12MB
Número total de pixels en la página	112,816,392
Número de recortes en la página	33
Opciones de compilación	-O3 (optimización completa)
Número de ejecuciones del programa	100

Tabla 6.2: Condiciones del entorno para realizar las pruebas de tiempo de NessieOcr.

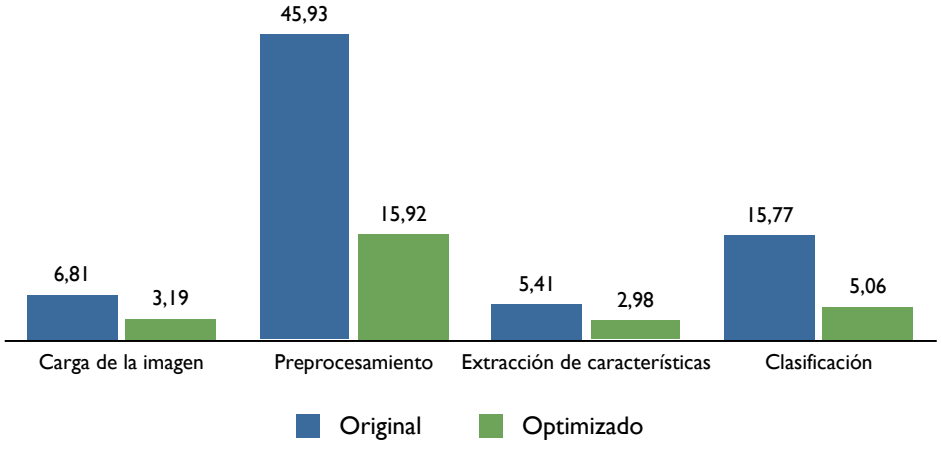


Figura 6.3: Mejora en el tiempo de ejecución tras la optimización de NessieOcr.

6.5. Tasa de acierto del clasificador

Si el tiempo de ejecución es un resultado importante, el factor principal por el que evaluamos la calidad de NessieOcr es la tasa de acierto del clasificador. La tasa de acierto η es un valor de referencia que estima el porcentaje promediado de caracteres por página que la biblioteca es capaz de reconocer con éxito con respecto al número total de caracteres de entrada:

$$\eta = \frac{\text{Caracteres reconocidos con éxito}}{\text{Número total de caracteres}} \tag{6.5.1}$$

Para medir la tasa de acierto del clasificador KNN hemos realizado un experimento similar al

Método	Tiempo (s)
Carga del dataset	0.54
Carga de la imagen desde disco	2.64
Copia de `Magick::Image` al vector interno	3.89
Eliminación de ruido en base a filtros	2.64
Binarización del espacio de color	0.29
Eliminación de ruido en base a plantillas	2.08
Segmentación de regiones	1.30
Fusión de caracteres compuestos	0.04
Ordenación de regiones	0.04
Construcción de patrones	4.48
Esqueletización de patrones	0.98
Cálculo de momentos normalizados	2.98
Clasificación KNN	5.06

Tabla 6.3: Resumen de tiempos de ejecución para los métodos más importantes de NessieOcr.

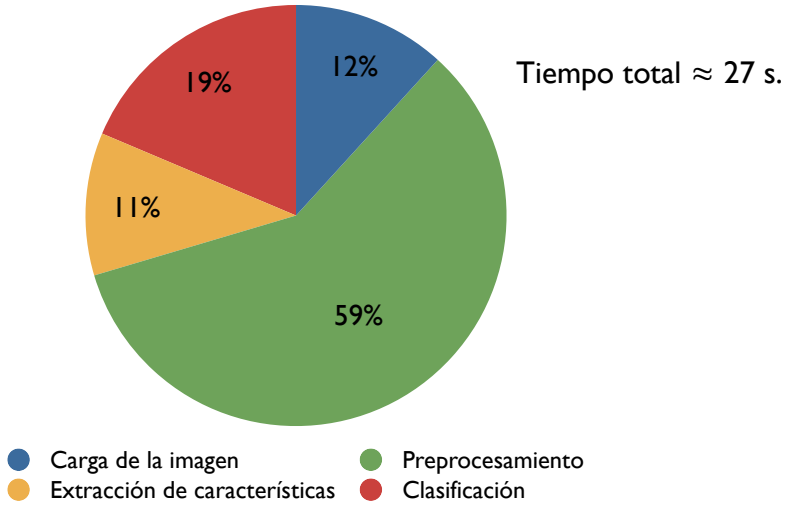


Figura 6.4: Distribución por etapas del tiempo de ejecución.

de la sección anterior, que medía los tiempos de ejecución. Sin embargo, ahora nuestro interés es calcular el resultado de la ecuación (6.5.1), comparando el texto resultante de la clasificación con un texto de referencia de entrada, que se pasa a través de un objeto `string` de C++. Las condiciones del experimento son las especificadas en la tabla 6.2.

Las pruebas se han realizado para valores de $K = 1, 3, 5, 7, 9$, utilizando diferentes vectores de características. Los resultados presentados a continuación son los obtenidos con los vectores de características de la tabla 6.4, que son los que mejores tasas de acierto han producido.

ID vector	Características
1	$m_{10}, m_{01}, m_{11}, m_{20}, m_{02}, m_{21}, m_{12}, m_{22}, m_{31}, m_{13}$
2	$m_{10}, m_{01}, m_{11}, m_{20}, m_{02}, m_{21}, m_{12}, m_{22}, m_{31}, m_{13}, m_{32}, m_{23}$
3	$m_{10}, m_{01}, m_{11}, m_{20}, m_{02}, m_{21}, m_{12}, m_{22}, m_{31}, m_{13}, m_{32}, m_{23}, m_{33}$

Tabla 6.4: Tasas de acierto obtenidas en la fase de entrenamiento para los tres mejores vectores de características.

Primero hemos evaluado la tasa de acierto en la fase de entrenamiento, teniendo en cuenta que para los primeros recortes el clasificador no tenía ninguna información previa. En la tabla 6.5 aparecen las tasas de acierto obtenidas para los tres vectores de características que ofrecen una mejor descripción de los patrones y por tanto una mejor clasificación.

ID vector	$K = 1$	$K = 3$	$K = 5$	$K = 7$	$K = 9$
1	74.39 %	75.90 %	75.19 %	74.44 %	73.03 %
2	79.12 %	77.87 %	77.25 %	75.84 %	74.25 %
3	77.81 %	75.81 %	74.51 %	73.50 %	71.75 %

Tabla 6.5: Tasas de acierto obtenidas en la fase de entrenamiento.

A continuación hemos evaluado el comportamiento del clasificador una vez ha sido entrenado, utilizando muestras de caracteres de un único periódico. En este caso la tasa de acierto es elevada porque la variabilidad en la morfología de la fuente es limitada. En la tabla 6.6 aparecen las tasas de acierto obtenidas.

ID vector	$K = 1$	$K = 3$	$K = 5$	$K = 7$	$K = 9$
1	99.31 %	96.08 %	93.94 %	91.99 %	90.08 %
2	99.35 %	95.96 %	93.37 %	92.06 %	90.51 %
3	99.42 %	96.13 %	93.32 %	90.39 %	89.19 %

Tabla 6.6: Tasas de acierto obtenidas en la fase de clasificación con tipografías usadas en el entrenamiento.

Por último hemos evaluado el comportamiento del clasificador cuando, habiendo sido entrenado para un conjunto de tipografías dado, se le presentan caracteres de otras tipografías. Para ello, hemos tomado el vector de características que presenta mejores resultados en promedio, y lo hemos contrastado con las tipografía de tres periódicos distintos. En este caso la tasa de acierto desciende alrededor de veinte puntos, siendo necesario realizar un nuevo entrenamiento para que el clasificador adquiera más generalidad. En la tabla 6.7 aparecen las tasas de acierto obtenidas.

Tipografía	$K = 1$	$K = 3$	$K = 5$	$K = 7$	$K = 9$
El Mundo	76.45 %	74.46 %	73.71 %	73.36 %	71.56 %
La Provincia	72.20 %	70.92 %	69.72 %	68.12 %	65.21 %
Canarias 7	72.54 %	71.31 %	69.59 %	68.12 %	66.31 %

Tabla 6.7: Tasas de acierto obtenidas en la fase de clasificación con tipografías diferentes a las del entrenamiento.

7 Conclusiones

NessieOcr es una biblioteca de funciones en C++ que permite realizar reconocimiento de caracteres en un entorno muy concreto. Aún pudiendo usarse para intentar reconocer cualquier texto de una imagen, lo cierto es que su funcionamiento está estrechamente ligado al desarrollo del proyecto Nessie. Por tanto, aunque parezca una herramienta poco versátil, esta característica tiene repercusiones positivas para el proyecto Nessie, dado que es posible hacer pequeños ajustes que beneficien a otros módulos, y que en otras herramientas más generales sería complicado.

Los métodos de preprocesamiento de imágenes expuestos, que incluyen la eliminación de ruido, la extracción de caracteres y su transformación en patrones susceptibles de ser clasificados, han sido elegidos teniendo en cuenta el compromiso entre efectividad y rendimiento. Si bien es necesario eliminar la mayor cantidad posible de información irrelevante, tampoco podemos permitirnos el lujo de perder mucho tiempo de ejecución. Las pruebas realizadas en esta etapa demuestran que acumula más de la mitad del tiempo de ejecución y es la más compleja computacionalmente. Sin embargo, esto beneficia las otras etapas, que ven reducida considerablemente la cantidad de información que tienen que procesar.

La implementación final de NessieOcr sólo utiliza como método de clasificación el paradigma KNN. Esta técnica ofrece buenos resultados a modo de referencia y como punto de partida para investigaciones posteriores. Pero su impacto negativo sobre el tiempo y el espacio consumidos desaconseja su uso en entornos de producción. Sin embargo, hemos preferido centrarnos en preparar un buen diseño que permita expandir las capacidades de NessieOcr, antes que lanzarnos a implementar técnicas adicionales de manera desordenada.

La arquitectura de NessieOcr ha sido diseñada con la intención de que futuros desarrolladores puedan realizar un mantenimiento y expansión de la biblioteca de manera sencilla:

- La aplicación del patrón *Facade* facilita enormemente el uso por parte de aplicaciones cliente, ya que inhibe de las particularidades de la biblioteca y presenta una interfaz de uso sencillo.

- La aplicación del patrón *Strategy*, con diferencia el punto más fuerte de la arquitectura, permite que nuevos algoritmos de clasificación puedan ser incorporados sin apenas modificar el código actual.

- Las recomendaciones de Scott Meyers, autor del libro *Effective C++* [26] entre otros, y en general, los prácticas de la comunidad de desarrolladores de C++, aportan un código de

fácil lectura, con estructuras y porciones de código conocidas como *idioms*.

Los resultados en cuanto a tiempo de ejecución de NessieOcr son relativamente aceptables teniendo en cuenta el tamaño de las muestras utilizados. Como media, una instancia del reconocedor procesa una página completa en veintisiete segundos. El lector recordará que había una restricción de tiempo en cuanto a la ejecución del reconocedor sobre una página: no podía sobrepasar los veinte segundos. Estudios posteriores a la imposición de dichas restricciones revelaron que los otros dos módulos del proyecto Nessie no necesitan veinte segundos para realizar su trabajo. Por tanto, el exceso de tiempo consumido por NessieOcr se ve compensado por el tiempo que le sobra a los demás módulos.

La tasa de acierto de NessieOcr está en torno varía entre un 70 % y un 80 %, dependiendo de las diferencias que existan entre la tipografía usada en el entrenamiento y la tipografía de entrada al sistema. Cuando se usa la misma tipografía del entrenamiento, la tasa de acierto varía entre un 90 % y un 95 %. Los resultados no están muy alejados de alcanzar cotas aceptables para un entorno en producción, ya que hemos obviado la parte de postprocesamiento que aumenta la tasa de acierto con el uso de diccionarios y otro tipo de técnicas. Además, otros paradigmas de clasificación, como las SVM o las redes neuronales, son capaces de aumentar la tasa de acierto por sí solas.

7.1. Trabajo futuro

Sin duda, un producto software en su primera versión no es perfecto, y siempre hay mejoras que pueden hacerse para ir puliendo errores y mejorando su rendimiento. A continuación proponemos varias líneas de trabajo que pueden continuar con la vida de NessieOcr en un futuro.

Con la aparición en el mercado de procesadores de múltiples núcleos, resulta atractivo desarrollar aplicaciones que hagan uso de programación multihilo. Dentro de la arquitectura de NessieOcr puede explotarse la concurrencia en varios puntos, pero creemos que los más importantes son los siguientes:

- Al finalizar el algoritmo de segmentación disponemos de una lista con las regiones que se han encontrado en el recorte de prensa de entrada. Si dividimos en partes iguales la lista y cada segmento lo asignamos a un hilo distinto podemos repartir la construcción de patrones entre los núcleos que dispongamos en el procesador.

- De manera análoga a la construcción de patrones, también podemos dividir el conjunto de patrones que sale de la etapa de preprocesamiento en segmentos que puedan asignarse a varios hilos. Así podemos calcular sus vectores de características en paralelo y ganar en tiempo de ejecución.

- En la etapa de clasificación puede dividirse el dataset que usa el algoritmo KNN en un número predeterminado de *subdatasets*. A la hora de calcular las distancias de un patrón incógnita con las muestras de entrenamiento, se puede lanzar un hilo independiente por cada subdataset y así repartir la carga entre los núcleos del procesador.

Uno de los problemas del paradigma de clasificación KNN es su alto consumo en espacio y tiempo. Recordemos que por cada muestra que quiera reconocerse hay que recorrer todo el dataset calculando distancias. Aún cuando dispongamos de soporte hardware para usar concurrencia y aliviar el consumo en espacio y tiempo, otra técnica que puede implementarse es la

condensación de datasets, que permite purgar un dataset eliminando muestras que no son muy representativas de una clase o que son muy parecidas a otras ya existentes.

El algoritmo KNN que viene por defecto en NessieOcr utiliza la distancia Euclídea como métrica para comparar cada muestra incógnita con las muestras de entrenamiento almacenadas. Sería interesante evaluar el comportamiento del clasificador con otras distancias, como por ejemplo la distancia de Mahalanobis.

Otra mejora importante que puede hacerse es el desarrollo de nuevos métodos de clasificación. Como hemos visto en capítulos anteriores, por defecto sólo está desarrollado el algoritmo KNN, que se comporta de manera aceptable y es un buen modelo de referencia para nuevos algoritmos. El problema es que arrastra la penalización de ser altamente costoso en tiempo y memoria. Por ello proponemos la implementación de otros algoritmos, como el algoritmo *back-propagation* usando redes neuronales artificiales, o el uso de las "Support Vector Machines" (SVM). El diseño de la arquitectura no debe suponer un problema, ya que basta con seguir el esquema propuesto a través del patrón *Strategy* en la sección 5.2.4.

Actualmente NessieOcr delega en Magick++ el algoritmo de normalización de regiones, que se ejecuta dentro del proceso de construcción de patrones. Sería ideal desacoplar esta dependencia desarrollando un algoritmo de normalización propio, para así mantener el espíritu global de mantener el código alejado de los mecanismos de carga de imágenes desde un soporte externo.

Uno los resultados que proporciona NessieOcr son las estadísticas. Hasta el momento, los requisitos de otros módulos del proyecto Nessie no requieren la presentación de estadísticas al usuario, sino que se han utilizado principalmente por los programadores para tareas de depuración. Si estos requisitos cambian puede que sea necesario revisar el diseño de la parte de la arquitectura que se encarga de ello, mejorando la interfaz de acceso a los datos por parte de programas cliente.

Los tiempos de ejecución expuestos en la sección 6.4 nos llevan a pensar que complicar la arquitectura de la aplicación para solventar los casos que hacen disminuir la tasa de acierto puede ser contraproducente. La cantidad de información que se maneja a bajo nivel es muy grande, y establecer caminos de realimentación para mejorar resultados puede aumentar el tiempo de ejecución. Por ello creemos que una mejor estrategia para mejorar la tasa de acierto puede ser el uso de diccionarios, trabajando en un nivel de abstracción mayor con el texto resultante, donde la cantidad de información es muy reducida y más sencilla de manejar.

7.2. Consideraciones personales

La realización de este proyecto ha tenido como consecuencia un enriquecimiento personal en varios aspectos, más allá de la culminación al trabajo realizado durante los meses de gestación o de la propia finalización de la carrera de Ingeniería Informática.

A nivel tecnológico, he sido capaz de adquirir un conocimiento más profundo sobre C++. La investigación sobre métodos de optimización de código específicos y la adopción de buenas costumbres de programación han sido fundamentales, y me han permitido un nivel alto de fluidez en el lenguaje. Para ello ha sido determinando la lectura de tres libros que recomiendo encarecidamente a todo programador de C++: *Effective C++* [26], *More Effective C++* [25] y *Effective STL* [27], todos del autor Scott Meyers.

A nivel académico, el proyecto ha significado profundizar en la aplicación de manera específica de muchos de los conceptos matemáticos adquiridos durante la carrera. Pasar de la teoría a los resultados es una experiencia muy interesante, más aún cuando uno va entendiendo el

proceso y adquiriendo nuevos conocimientos. Pero al mismo tiempo, el hecho de explorar un campo tan específico y amplio como la manipulación de imágenes y la inteligencia artificial me ha hecho reflexionar sobre los objetivos y la distribución de algunas materias en la Ingeniería Informática. Enfrentarse al diseño de un software empleando conceptos matemáticos es un proceso complicado, para el cual creo que es necesario tener un mayor bagaje en ambas áreas.

Por último, a nivel personal ha sido muy interesante la experiencia adquirida durante todo el desarrollo. El proyecto Nessie toca otros proyectos que era necesario tener en cuenta, lo cual se traduce en estar en contacto con otros grupos de trabajo. Y la relación "tutor-alumno" en muchas ocasiones es la antesala de lo que en un futuro puede considerarse la relación de un empleado con su jefe. Por todo ello, creo que ha sido una experiencia positiva como ensayo a la inserción en el mercado laboral.

Bibliografía

[1] G. Booch, J. Rumbaugh, and I. Jacobson.
UML: El Lenguaje Unificado de Modelado.
Addison Wesley, 2006.

[2] BurrellesLuce.
BurrellesLuce - Press Clipping Service, Media Directory, Media Monitoring and Media Analysis.
http://www.burrellesluce.com/, 2008.

[3] M. Cheriet, N. Kharma, and C. Liu.
Character Recognition Systems: A Guide for Students and Practitioners.
Wiley-Interscience, 2007.

[4] Alessio Damato et al.
LaTeX.
Wikibooks - WikiMedia, 2009.

[5] Dimitri van Heesch.
Doxygen.
http://www.stack.nl/~dimitri/doxygen/, 2009.

[6] R.O. Duda, P.E. Hart, and D.G. Stork.
Pattern Classification.
Wiley Interscience, 2001.

[7] Federation Internationale des Bureaux d'Extraits de Presse (FIBEP).
FIBEP History.
http://www.fibep.info/, 2008.

[8] Free Software Foundation.
GCC- The GNU Compiler Collection.
http://gcc.gnu.org/, 2009.

[9] E. Freeman, E. Freeman, B. Bates, and K. Sierra.
Head First Design Patterns.
O'Reilly & Associates, Inc., 2004.

[10] freshmeat.net.
Project details for Clara OCR.
http://freshmeat.net/projects/claraocr/, 2008.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides.
Design patterns.
Addison-Wesley, 2002.

[12] R.C. Gonzalez and R. Woods.
Tratamiento digital de imágenes.
Ediciones Diaz de Santos, 1996.

[13] Google Code.
Ocropus.
http://code.google.com/p/ocropus/, 2008.

[14] ImageMagick Studio LLC.
ImageMagick - Convert, Edit, and Compose Images.
http://www.imagemagick.org/, 2009.

[15] ImageMagick Studio LLC.
Magick++ - C++ API for ImageMagick.
http://www.imagemagick.org/Magick++/, 2009.

[16] International Association of Broadcast Monitors (I.A.B.M.).
IABM - About Us.
http://www.iabm.com/about/, 2008.

[17] I. Jacobson, G. Booch, and J. Rumbaugh.
El proceso unificado de desarrollo de software.
Addison Wesley, 2000.

[18] Jeroen T. Vermeulen.
libpqxx.
http://pqxx.org/development/libpqxx/, 2009.

[19] À. Jiménez.
Acceso a información periodística a través de servicios de press clipping.
*Hipertext.net (*http://www.hipertext.net/web/pag248.htm*)*, 2003.

[20] I. Kotoulas, L. y Andreadis.
Image analysis using moments.
IEEE International Conference on Technology and Automation (ICTA), page 360—364, 2005.

[21] L. Lamport.
LaTeX, a document preparation system.
Addison Wesley Professional, 1994.

[22] LAURIN Project, University of Innsbruck.
Laurin - Home.
http://laurin.uibk.ac.at/old/index.html.en, 2001.

[23] C.L. Liu and K. Marukawa.
Pseudo two-dimensional shape normalization methods for handwritten chinese character recognition.

Pattern Recognition, 38(12):2242—2255, 2005.

[24] C.L. Liu, K. Nakashima, H. Sako, and H. Fujisawa.
Handwritten digit recognition: investigation of normalization and feature extraction techniques.
Pattern Recognition, 37(2):265—279, 2004.

[25] Scott Meyers.
More effective C++: 35 new ways to improve your programs and designs.
Addison-Wesley, 1996.

[26] Scott Meyers.
Effective C++: 50 specific ways to improve your programs and designs.
Addison Wesley, 1997.

[27] Scott Meyers.
Effective STL.
Addison-Wesley, 2001.

[28] S. Morison and J.M. Pujol.
Principios fundamentales de la tipografía.
Ediciones del bronce, 1998.

[29] R. Mukundan, SH Ong, and PA Lee.
Image analysis by tchebichef moments.
IEEE Transactions on Image Processing, 10(9):1357—1364, 2001.

[30] National Library of Australia.
Pandora Archive - Digital Archiving System.
http://pandora.nla.gov.au/pandas.html, 2008.

[31] H. Niemann.
Pattern analysis and understanding.
Springer, 1990.

[32] Nobuyuki Otsu.
A threshold selection method from gray-level histograms.
IEEE Transactions on systems, man, and cybernetics, page 62—66, 1979.

[33] Garr Reynolds.
Presentation Zen.
New Riders, Berkeley (California), 2008.

[34] Olive Software.
Olive Software - the leading provider of digital edition & digital archiving solutions for the publishing industry.
http://www.olivesoftware.com/, 2008.

[35] M. Sonka, V. Hlavac, and R. Boyle.
Image Processing, Analysis, and Machine Vision.
Thomson Learning, 2008.

[36] B. Stroustrup.
The C++ programming language.
Addison Wesley Professional, 2000.

[37] Tangentsoft.
MySQL++.
http://tangentsoft.net/mysql++/, 2009.

[38] F.M.H. Tejera and J.J.L. Navarro.
Reconocimiento de formas: clasificación y aprendizaje.
Universidad de Las Palmas de Gran Canaria, 2002.

[39] Tigris.org.
Subversion.
http://subversion.tigris.org/, 2009.

[40] TNS Global.
TNS - Market Research, Information & Business Insight Services.
http://www.tnsglobal.com/, 2008.

[41] H. Zhu, H. Shu, T. Xia, L. Luo, and J. Louis Coatrieux.
Translation and scale invariants of tchebichef moments.
Pattern Recognition, 40(9):2530—2542, 2007.

[42] Zissor.
A world leading supplier.
http://www.zissor.com/, 2008.