industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Desarrollo de Software de Uso Profesional para la Inspección de Piezas en la Industria mediante Herramientas de Visión Artificial y Machine Learning

TRABAJO FIN DE MÁSTER

MÁSTER EN INGENIERÍA INDUSTRIAL

Autor: **Alejandro Olivo García**
Director: D. Javier Navarro Lorente

Cartagena, mayo de 2020

Universidad Politécnica de Cartagena

Índice

Agradecimientos

Siempre imaginé empezar un apartado de agradecimientos con el típico “Quisiera dar las gracias…” pero parece que he encontrado otra forma de hacerlo. Lo cierto es que jamás imaginé introducir un apartado como este en un trabajo, así que supongo que si lo estoy haciendo ahora es porque realmente tengo algo que decir.

Quiero empezar agradeciendo a la empresa MTorres el tiempo que han invertido en mí. Gracias a Leandro y a J. A. Rosillo por la mano que me han echado en todo momento en los que se convirtieron en mis primeros pasos de un mundo que creo es el más estimulante de los que he descubierto en este constructo social denominado *ingeniería*: la visión artificial.

Por otro lado, y no por compromiso como tengo constancia que se hace a menudo, a mi tutor P. J. Lorente, por animarme a descubrir un poco de la caja negra que es la Inteligencia Artificial. También, incluso de manera más importante, por introducirme en la programación más allá de lo que una escasísima asignatura introductoria lo hizo en una carrera cuya carencia de materia relacionada con este ámbito debería generar un incómodo sentimiento de vergüenza a aquellas personas encargadas de decidir todos estos asuntos.

A mis padres, que me lo han enseñado todo en esta vida y con los que jamás saldaré la deuda que contrajeron al traerme a este mundo. A mi hermano, del que podría escribir otro trabajo como este para enumerar las cosas buenas que me ha traído. A Sandra, cuya iniciativa y capacidad de emprender es de otro mundo, y de la que sigo aprendiendo cosas cada día.

Y a la música, que es movimiento.

Fdo.: Alejandro Olivo García, 2020

Abstract

Since the development of Artificial Intelligence since the middle of the last century, improved mathematical models and advances in hardware have opened a range of possibilities that must be properly integrated into the industry. Within this great paradigm that is AI, there are two fields that offer a quantity of tools that are not negligible to carry out the tasks of quality inspection or identification of defects at a more advanced level in relation to how it was being done by the end of the last century.

Computer Vision and Machine Learning are, therefore, the two protagonists in this software, Spector, which tries to stand out as a versatile, intuitive tool with a relatively high level of complexity for the task of inspecting parts or defects in various industrial contexts.

Resumen

Desde el desarrollo de la Inteligencia Artificial desde mediados del siglo pasado, los modelos matemáticos mejorados y los avances en hardware han abierto un abanico de posibilidades que deben ser integrados de manera adecuada en la industrial. Dentro de este gran paradigma que es la IA, destacan dos campos que ofrecen una cantidad de herramientas nada despreciable para llevar a cabo tareas de inspección de calidad o identificación de defectos a un nivel más avanzado con relación a como se venía haciendo a finales del siglo pasado.

La Visión por Computador y el Machine Learning son, por lo tanto, los dos protagonistas en este software, Spector, que intenta destacar como una herramienta versátil, intuitiva y de un nivel relativamente alto de complejidad para la tarea de inspeccionar piezas o defectos en diversos contextos industriales.

1. Introducción

En este primer capítulo se presenta el actual proyecto, un software de inspección de piezas para ámbito industrial que presenta una interfaz de uso sencillo y que integra herramientas de visión y de inteligencia artificial.

1.1. Objetivos del Proyecto

Los objetivos del proyecto son los que se desarrollan a continuación.

El objetivo principal es en sí la creación del software de inspección mediante el entorno de programación de Visual Studio, y en lenguaje C#. Dicho software permitirá, de manera sencilla, llevar a cabo la inspección de piezas a partir de las imágenes tomadas por una cámara directamente sobre la línea de producción, o bien en una estación por separado dedicada exclusivamente a tareas de calidad. Dicha inspección debe consistir en la identificación de si una pieza es o no correcta, en relación con otras piezas modelo que contengan o no defectos de fabricación, apreciables visualmente.

Más allá de este objetivo principal, se pretenden otros de carácter secundario, que son los siguientes:

- Creación de una interfaz gráfica con cierto nivel de complejidad que incluya funcionalidades típicas de los softwares utilizados en la industria: gestión de datos, log, configuración de parámetros…

- Almacenamiento de datos en formato XML: bases de datos de modelos de piezas, de conjuntos de entrenamiento o de cadenas de proceso entre otras.

- Profundización en los clasificadores y algoritmos relacionados, con el fin de ofrecer al usuario una experiencia completa con diversas posibilidades de elección para llevar a cabo la inspección.

- Creación de un manual de uso para explicar el funcionamiento del programa mediante un lenguaje asequible y dentro de un contexto industrial.

1.2. Descripción de los Capítulos

Lo que resta de memoria del presente proyecto quedaría estructurada de la siguiente manera:

1) Fundamento teórico

Se trata de del capítulo que abre el proyecto, buscando arrojar luz sobre los fundamentos teóricos que sustentan todas las decisiones y acciones tomadas en la realización del software.

2) El Software

Es la parte central de la memoria, donde se pasa a describir detalladamente el software, haciéndolo desde dos puntos de vista:

Por un lado, en un primer apartado se describe el funcionamiento del software, a nivel usuario. Esto es, cómo debe ser usado, las diferentes posibilidades que ofrece, y cada una de las funciones y configuraciones posibles dentro del mismo.

Por otro, se entra en un apartado más técnico en un segundo apartado dedicado a desarrollar las características del software a nivel programaciones: funciones de librería utilizadas y fragmentos de código de interés para el proyecto en particular.

3) Caso Práctico

Se trata de la parte del proyecto donde se pone a prueba el software, llevando a cabo el proceso completo de edición, entrenamiento e inspección, e intentando pasar por el máximo número de puntos posible.

4) Conclusión y Trabajo Futuros

En este apartado se busca extraer las conclusiones más importantes que derivan de la realización de este proyecto, alumbrando el camino hacia futuros proyectos relacionados.

5) Bibliografía

Contiene la lista de trabajos, artículos, webs y libros consultados durante la elaboración del proyecto y la memoria.

6) Anexo: Manual de Usuario

Se trata de una primera versión del manual que contendría la información necesaria para poner en marcha el software por parte de un futuro cliente que lo haya adquirido. Contiene de manera clara y sencilla la información necesaria para sacar provecho de todas las funcionalidades del programa.

2. Fundamento Teórico

A lo largo de este capítulo se va a desarrollar el fundamento teórico que sustenta todo el software de inspección. Esto incluye los principios fundamentales de las interfaces gráficas centradas en el usuario, así como un repaso por la historia y los algoritmos más importantes que han resultado de utilidad en este proyecto en los dos campos de más peso: la visión artificial y el Machine Learning.

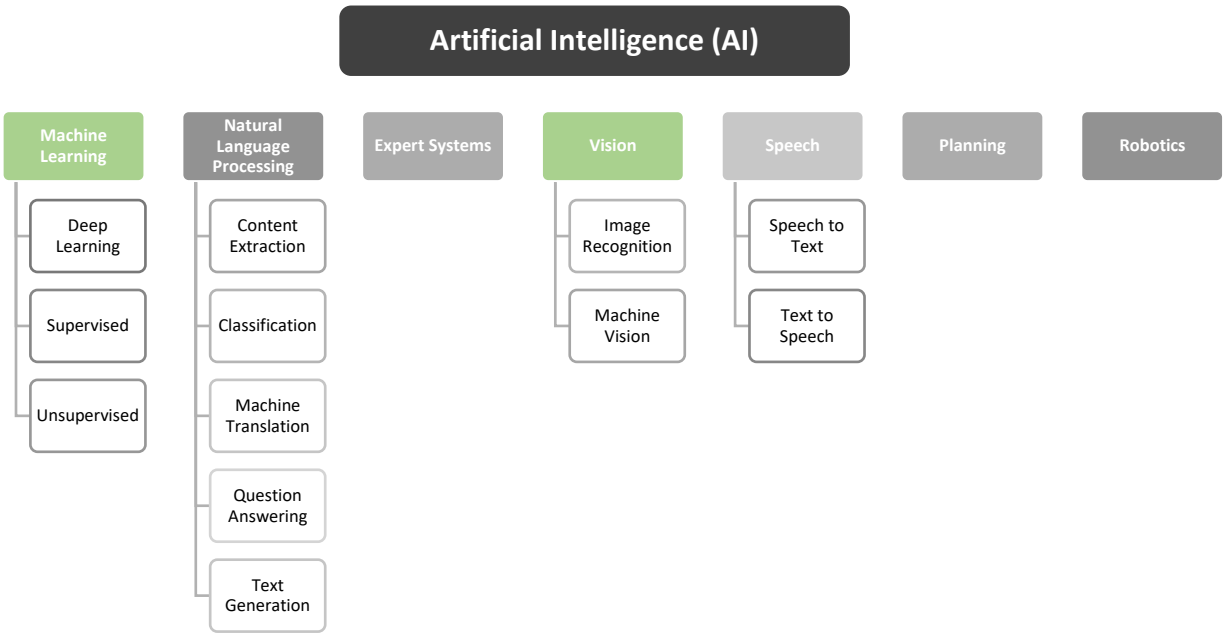


Figura 2.1 – División del Campo de la Inteligencia Artificial

En la figura 2.1 puede apreciarse la cantidad de campos que componen el mundo de la Inteligencia Artificial, tan de moda en el presente siglo, pero mucho más diverso de lo que podría pensarse a priori.

Antes de comenzar con el fundamento teórico, conviene estudiar en qué punto se encuentra la industria actual con respecto a este tipo de softwares de inspección en los que la inteligencia artificial toma un peso tan importante.

2.1. Estado del Arte

La primera parte de esta memoria está dedicada al análisis de los softwares actualmente en el mercado con funcionalidades similares a las que presenta Spector.

En la actualidad cualquier industria concienciada con la calidad de sus productos dedica una parte importante de su actividad y sus recursos a establecer los sistemas necesarios para llevar a cabo este control de manera óptima y eficaz.

Los sistemas de visión e inteligencia artificial se encuentran en un importante auge ya que son capaces de dar solución a los problemas de inspección mediante el uso de herramientas

que hasta ahora han sido desarrollados más bien de cara a la investigación. El empleo de clasificadores y otros algoritmos de este campo es aún apenas conocido por los ingenieros que tienen peso dentro de las empresas dedicadas a la producción y a menudo es común que los softwares que incluyen esta tecnología sean una caja negra para estos, en parte debido a la complejidad matemática y estadística que conllevan estos algoritmos. Es por esto por lo que es común encontrar empresas o partes de estas en el campo de la programación que propongan soluciones de este tipo mediante softwares y paquetes que se instalan directamente en la planta de sus clientes.

En la figura 2.2 se muestran algunos de los softwares que llevan a cabo este tipo de tareas, realizando una comparativa con Spector según el tipo de servicios que ofrecen y más adelante se describen algunos de estos programas [1].

	IOS/Linux	Clasificación	Reconocimiento de Formas	Extracción de Características	Procesamiento de Imagen	Imagen a Texto	Red Neuronal
Spector	⛔	✔	✔	✔	✔	✔	⛔
Matrox Imaging	✔	✔	✔	✔	✔	✔	✔
N.I. Vision	✔	✔	✔	✔	✔	✔	✔
HALCON 13	✔	⛔	✔	✔	✔	✔	✔
Sapera Vision	✔	⛔	✔	✔	✔	⛔	⛔
Sherlock	✔	⛔	✔	✔	✔	✔	⛔

	Calibración y Medida	Interfaz Accesible	Independencia de otro Software	Baja Cualificación para el Uso	3D	Servicio Técnico
Spector	⛔	✔	✔	✔	⛔	⛔
Matrox Imaging	✔	✔	✔	⛔	✔	✔
N.I. Vision	✔	⛔	⛔	⛔	✔	✔
HALCON 13	✔	✔	✔	⛔	✔	✔
Sapera Vision	✔	✔	✔	⛔	⛔	✔
Sherlock	✔	✔	✔	✔	⛔	✔

Figura 2.2 – Tabla Comparativa de Softwares de Vision

La librería *Matrox Imaging,* desarrollada por Matrox® (1976), es, más que un software de visión, un conjunto de herramientas dedicadas a la creación de uno [2]. El campo de aplicación es la visión por computador, aunque también está compuesta por algoritmos de inteligencia artificial y, entre los diversos usos que recibe, los más importantes están enfocados al campo de la industria, aunque también al de la medicina.

La empresa National Instruments™ tiene un módulo de visión que permite programar y configurar algoritmos de visión y procesamiento de imágenes por medio de LabVIEW.

HALCON 13 es una librería de programación para el desarrollo de aplicaciones de visión artificial, siendo de hecho una de las más famosas en el mundo. Tiene importantes aplicaciones en visión industrial y también en procesamiento de imagen.

El software Sapera Vision, de DALSA, tiene la misma utilidad que en los casos comentados anteriormente, aunque además proporciona un gran número de ejemplos de programación que sirve de gran utilidad a la hora de aprender el manejo de la librería.

Sherlock es un software de visión industrial específicamente diseñado para facilitar el desarrollo de aplicaciones de visión tales como alineación, medida, inspección, verificación de conexiones y tareas de guiado de maquinarias.

2.2. Interfaces Gráficas

En este capítulo se desarrollan algunos de los fundamentos más importantes centrados en la creación de interfaces gráficas, tenidos en cuenta en la elaboración del software que se describe en este proyecto.

2.2.1. Diseño Centrado en el Usuario

La creación de una interfaz gráfica centrada en el usuario se basa en una metodología conocida como Diseño Centrado en el Usuario, y parte de esta con el objetivo de facilitar y viabilizar la interacción entre un ser humano y una máquina o una computadora [3]. El UCD, por sus siglas en inglés, se centra en ciertos objetivos que no son otros que la búsqueda de una buena usabilidad y la integración de las características de los usuarios así como del contexto que los rodea: espacio productivo, flujo de trabajo, etc [4].

Figura 2.3 – Fundamentos de UCD

Los principios básicos en los que se fundamenta esta metodología son los 3 que se describen a continuación, representados en la figura 2.3:

- Pensamiento de Diseño

  Consiste en el planteamiento y comprensión de las necesidades del usuario para llegar a una creación de interfaz tecnológicamente asumible y en constante consideración de la evolución de las tareas involucradas en el caso [5]. Este método se sustenta sobre algunos factores diferenciadores como la búsqueda de empatía hacia los usuarios involucrados, el trabajo en conjunto de personal de diversos campos, la creación previa de prototipos y versiones de prueba, un ambiente de trabajo y creación distendido o el empleo de técnicas de esquematización y contenido visual.

- Desarrollo Ágil

  Este tipo de metodologías consiste en la creación y desarrollo de un software que esté sujeto a cambios en el tiempo [6]. Dicho de otra forma, a modo que las necesidades vayan modificándose se deberán implementar cambios en el programa que lo dote de la flexibilidad y dinamismo propios de los procesos que suelen tener lugar en un ambiente productivo.

- Usabilidad y Experiencia de Usuario

  Puede definirse la usabilidad como un grado de efectividad y satisfacción en la realización de cierta tarea por parte de un usuario. Este principio habla precisamente de la búsqueda de maximizar dicha cualidad, unido a otras cualidades como la facilidad de aprendizaje por parte de un futuro usuario y a la integración de otros elementos abstractos como las emociones: comodidad, simpatía y otras percepciones positivas [7].

2.3. Fundamentos de Visión Artificial

En este capítulo se presenta un acercamiento al cada vez más importante mundo de la visión artificial, o visión por computador. Es este campo de la programación uno de los dos en los que el presente proyecto se sustenta, junto al también creciente mundo del Machine Learning, tratado en el siguiente capítulo.

2.3.1. Historia y Antecedentes

El campo de la visión artificial se inicia hace apenas unos sesenta años, cuando en el MIT el denominado padre de la VA, Larry Roberts (al que puede verse en la figura 2.4), consigue extraer información tridimensional de un conjunto de bloques en una imagen 2D. En este punto, multitud de investigadores empiezan a desarrollar algoritmos que continúan con este trabajo, como los más básicos para la detección de bordes o segmentación [8].

Figura 2.4 – Larry Roberts, 1960's

El siguiente gran avance en este campo llega con el libro "*A Computational Investigation into the Human Representation and Processing of Visual Information"*, de David Marr (1982), donde el científico proponía una división de la visión en tres niveles: un primer nivel computacional (describe el sistema y sus problemas y causas), un segundo nivel de algoritmo (describe la implementación de la teoría computacional, la transformación a llevar a cabo) y el último nivel, el físico (que describe cómo puede llevarse a cabo de manera física el nivel anterior) [9].

De manera posterior a los avances e investigaciones de Marr, una gran cantidad de científicos han apostado por un acercamiento más diverso al problema de la visión por computador. Esto se debe a la dificultad que conlleva, en ciertos casos, recrear un modelo 3D a partir de las imágenes en 2D, mediante el proceso indicado por el propio Marr: obtención del "sketch primario" (bordes, segmentos, aristas, vértices…), procesamiento complejo estructural mediante conocimientos *a priori* y obtención del modelo final en 3D. La realidad es que en diversas aplicaciones no es necesario llegar al punto final indicado por el investigador, sino que la mayoría de las aplicaciones que hacen uso de este tipo de algoritmos son capaces de extraer información útil y suficiente de un análisis más sencillo.

Finalmente, en las últimas décadas del siglo pasado el avance en el campo de la visión por computador ha ido dirigido a la mejora matemática de los algoritmos de detección de bordes, contornos, márgenes, etc [10]. Del mismo modo, los softwares de reconocimiento de objetos se enfrentaron en los noventa al problema del análisis del movimiento y ya en el presente siglo se ha apreciado un aumento progresivo y muy notable de la cantidad de investigaciones y proyectos que ha ido mejorando poco a poco dichos algoritmos.

En los últimos años, el campo de la inteligencia artificial se ha unido estrechamente al de la visión en términos de segmentación y clasificación, parte fundamental del proceso de análisis de imágenes.

2.3.2. Sistema Industrial de Visión por Computador

Un sistema de visión por computador, que es lo que en definitiva se pretende crear con este software, puede definirse como el conjunto de herramientas de software y hardware que permiten, de manera automática y minimizando o eliminando la intervención del ser

humano, cuantificar y gestionar las características físicas de un elemento [11]. En este sentido, se habla de un sistema de visión artificial industrial al tratarse este elemento de una pieza, objeto, producto, materia prima, etc., dentro de un contexto puramente industrial. Es precisamente en este contexto donde estos algoritmos permiten, mediante la toma y procesamiento de imágenes, determinar características del elemento en cuestión que a simple vista serían imperceptibles o difícilmente medibles mediante métodos convencionales.

Se puede establecer una diferenciación básica entre los diferentes elementos que conforman el sistema de visión artificial al completo, representados en la figura 2.5, que de manera simplificada serían cuatro:

- Línea de Producción

  Se trata del entorno en el que se monta el sistema, que puede ser muy variado en cuanto a características: depósitos de líquidos o graneles, cintas transportadoras, mesas de trabajo, etc. Es en este entorno donde aparecen los elementos que se pretenden cuantificar, medir o inspeccionar: piezas, niveles, materiales… Además, dicho lugar debe contar con un sistema adecuado de iluminación, parte fundamental para asegurar el correcto funcionamiento del sistema al completo.

- Cámara

  Existen en el mercado cámaras de diversos tipos: matriciales, lineales, 3D, espectrales, de alta velocidad, infrarrojas… Además, se trata de un elemento que trae consigo la necesidad de contar con accesorios auxiliares como iluminación, soportes, lentes, cables de conexión o sistema de alimentación.

- Software

  En este punto se alcanza el objeto de este proyecto, pues el mismo consiste en la creación de un software, dejando a un lado el desarrollo del resto del sistema.

- Elementos Actuadores

  Se trata del conjunto de elementos mecánicos con el que el sistema interfiere sobre la línea de producción o, más bien, sobre un elemento de esta. En esta parte se aglutinan brazos robóticos, mecanismos, actuadores neumáticos, etc., y cualquier elemento que permita modificar el estado de la línea de producción de acuerdo con los resultados obtenidos del procesamiento de las imágenes tomadas por la cámara y gestionadas o cuantificadas por el software de visión.

Figura 2.5 – Sistema de Visión por Computador

Si se pone el foco sobre la parte del software de visión artificial, dejando de lado las actuaciones físicas sobre los elementos, se reconocen de manera tradicional cuatro fases en todo sistema de visión [12]:

I. Fase de Captura: adquisición de las imágenes, ya sea directamente de la cámara o de una base de datos donde se han ido acumulando las mismas.
II. Fase de Procesamiento Previo: consiste en el tratamiento digital de las imágenes mediante algoritmos de transformación, filtro o cualquier otro tipo de edición.
III. Fase de Segmentación: consiste en aislar los elementos de interés en el contexto de la imagen.
IV. Fase de Reconocimiento o Clasificación: es la comprensión y caracterización de los elementos de interés dentro de la imagen.

Como se comenta anteriormente, la parte del sistema de visión a la que pertenece este proyecto sería precisamente la de software, lo que hace necesario hablar de cuáles son las herramientas de programación relacionadas con este campo. Dichas herramientas no son otra cosa que los algoritmos típicos de la visión artificial, disponibles en librerías como OpenCV.

2.3.3. OpenCV

OpenCV es una librería de código abierto enmarcada dentro del campo de la visión artificial, pues contiene una gran cantidad de algoritmos de tratamiento de imágenes que permiten contar con las herramientas necesarias para resolver cualquier problema de este ámbito. Esta librería fue desarrollada como un proyecto de investigación de Intel en 1998, pero la primera versión no estuvo disponible al público hasta el año 2000 [13].

Escrita en C y C++ y funcional para Linux, Windows y Mac OS X, es una librería que tiene como objetivo aportar las herramientas necesarias y la estructura de clases suficiente como para crear, de manera relativamente sencilla, aplicaciones de visión artificial con un nivel bastante sofisticado [14]. En OpenCV hay cabida para algoritmos de procesamiento de

imagen de bajo nivel al mismo tiempo que para algoritmos de alto nivel que incluyen reconocimiento facial, de peatones, tracking, etc.

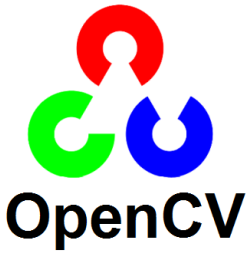


Figura 2.6 – Logo de OpenCV

OpenCV, cuyo logo puede verse en la figura 2.6, contiene más de 500 funciones que, englobadas dentro de la visión, se expanden hacia otros campos como la robótica, interfaces de usuario, calibración de cámaras, seguridad… Además, esta librería permite el uso de algunas funciones de Machine Learning, si bien en este software no han sido utilizadas.

La popularidad de OpenCV es tal que hoy en día es ampliamente utilizada por profesionales de diversos ámbitos, ya sea en etapas educativas como aquellas más puramente profesionales. A los usos más convencionales como análisis biomédicos o conducción autónoma se unen ahora importantes aplicaciones industriales, como en programas de inspección, ya sea mediante interfaz hombre-máquina como en análisis de imágenes en robots de fabricación.

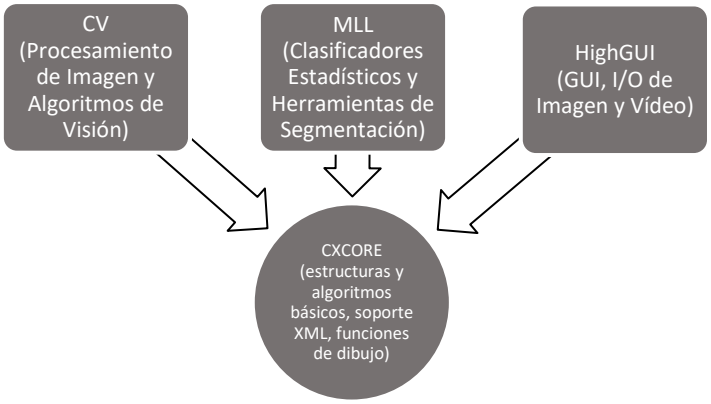


Figura 2.7 – Estructura Básica de OpenCV

En la figura 2.7 se representa la estructura básica de OpenCV, donde se observa que la librería está formada por 4 componentes, fundamentalmente. CV contiene la mayoría de las funciones que resultan de interés para este proyecto, así como también lo resultan las estructuras básicas contenidas en CXCore [14]. Como ya se comentaba anteriormente, las funciones recogidas en MLL, dedicadas al análisis estadístico, no son de interés en este proyecto, pues se ha optado por el uso de otra librería como es Accord.

2.3.4. Estructuras y Algoritmos de Visión

En este capítulo se van a presentar todos aquellos algoritmos que han sido utilizados en el software, concretamente en la parte referente al procesamiento previo de las imágenes: desde las ediciones previas hasta el análisis de contornos y creación de blobs de los que extraer las características de utilidad.

- Imágenes

Antes de entrar en el desarrollo de los algoritmos conviene conocer que una imagen es, en definitiva, una matriz de números. Como es de conocimiento general, una imagen está formada por píxeles, celdas que adquieren un valor que representa la intensidad de dicho pixel. Si se va al caso más sencillo, esto es, una imagen en blanco y negro, lo que se tiene es una cuadrícula donde cada píxel adquiere un valor de intensidad entre 0 (blanco absoluto) y 255 (negro absoluto).

En el caso de las imágenes a color (RGB), lo que se tienen son 3 matrices: una para el nivel de intensidad rojo (R), otra para el verde (G) y otra para el azul (B). En la figura 2.8 se ve de forma gráfica la estructura de matrices que se comenta en este apartado.

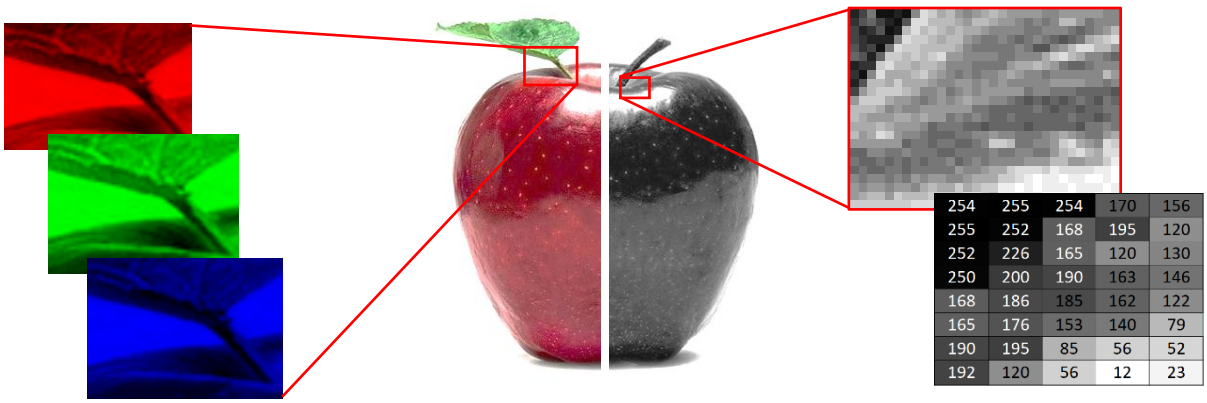


Figura 2.8 – Representación Imágenes ByN y a Color

- Binarización

Una de las transformaciones más importantes que se utiliza a menudo sobre la imagen es la binarización o *Threshold.* Esta operación consiste en establecer un valor frontera situado entre 0 y 255 para establecer una condición sobre los píxeles de la imagen. Cada píxel tomará el valor de 0 o 255 en función de si su valor previo era inferior o superior a ese valor límite. En la figura 2.9 se muestra esta transformación con el histograma de la imagen a binarizar:

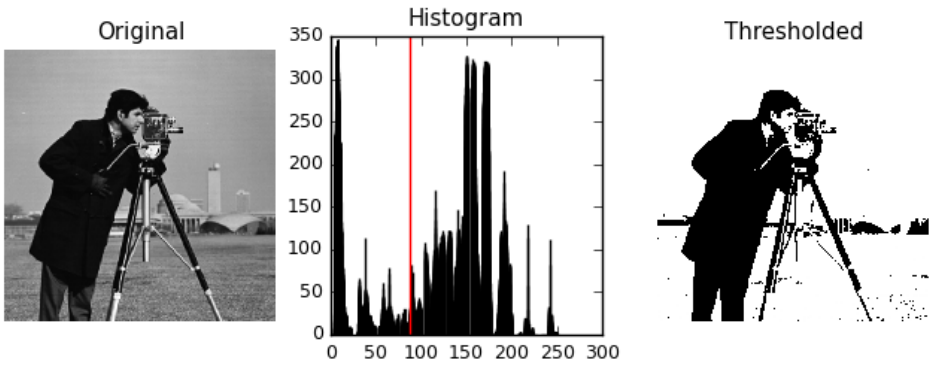


Figura 2.9 – Binarización de una Imagen

A la hora de realizar esta transformación cabe preguntarse cuál es el valor para establecer el límite. Si bien puede establecerse en cualquier punto, existen algoritmos que calculan dicho valor en función de los valores de los píxeles de la imagen. Uno de ellos es el algoritmo OTSU, que establece que el valor óptimo de umbralización se elige de manera que la varianza entre clases de una imagen umbralizada sea máxima [15]. Matemáticamente:

$$t^* = Max\{\sigma_B^2(t)\} \qquad 1 \leq t \leq L$$

donde

$$\sigma_B^2 = \omega_1 \cdot (\mu_1 - \mu_T)^2 + \omega_2 \cdot (\mu_2 - \mu_T)^2$$

$$\mu_T = \omega_1 \cdot \mu_1 + \omega_2 \cdot \mu_2$$

- Operaciones Morfológicas

Las operaciones morfológicas sobre imágenes son transformaciones que se realizan para simplificarlas sin que se vean modificadas las características principales de forma de los objetos que en estas aparecen. Es precisamente gracias a esta capacidad que estas transformaciones resultan de gran utilidad en el procesamiento previo de las imágenes, sobre todo a la hora de suprimir ruidos o simplificar formas. Las operaciones morfológicas principales son dos, erosión y dilatación, aunque también existen otras como la apertura o el cierre [16].

La dilatación es el proceso por el cual los objetos que aparecen en una imagen binarizada aumentan su tamaño. En la figura 2.10 puede verse un ejemplo de esta transformación. Para ello se lleva a cabo una transformación tal que, dada una imagen A, y un elemento estructural B, (siendo ambas imágenes binarias con fondo blanco), se define como:

$$A \oplus B = \left\{x | \left(\hat{B}\right)_x \cap A \neq \emptyset\right\}$$

Donde, para la intersección solo se tienen en cuenta los píxeles negros de A y B.

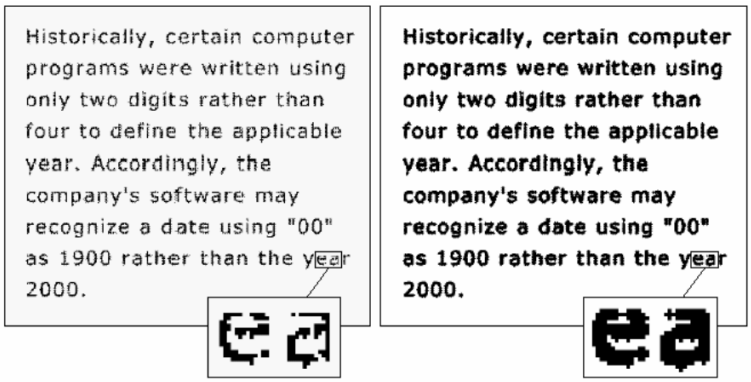


Figura 2.10 – Ejemplo de Proceso de Dilatación

En el caso de la erosión lo que se pretende es reducir el tamaño de los objetos de la imagen, al contrario que lo visto en el caso anterior. Ahora, la erosión de una imagen A por un elemento estructural B se define como:

$$A \ominus B = \{x | B_x \subseteq A\}$$

Se considera, pues, que la erosión es la operación morfológica dual de la dilatación. En la figura 2.11 se observa un ejemplo del proceso de erosión.

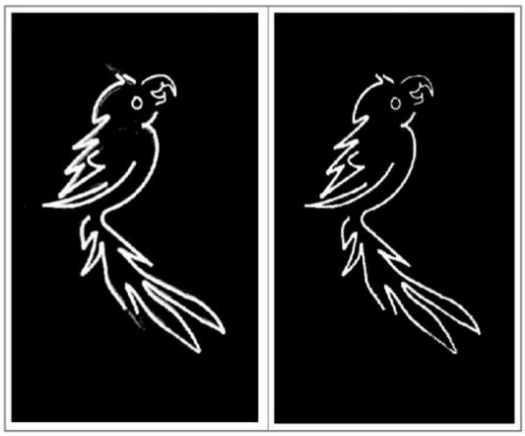


Figura 2.11 – Ejemplo de Proceso de Erosión

Las otras operaciones morfológicas comentadas anteriormente no son más que combinaciones de las dos explicadas. La apertura consiste en llevar a cabo una erosión y posteriormente una dilatación. Del mismo modo, el cierre es realizar las dos transformaciones anteriores en el orden inverso [17].

- Contornos:

La búsqueda de contornos en una imagen binarizada consiste en la obtención de una nube de puntos que delimitan una región determinada de una imagen que es de color negra, frente al fondo blanco de la misma. Esta nube, o contorno, es una secuencia de puntos que define el borde del objeto en cuestión [18].

Si bien OpenCV proporciona las funciones necesarias para llevar a cabo este proceso, es relativamente sencillo entender en qué consiste un algoritmo típico de detección de contornos, como puede ser el propuesto por Moore [19]. Dicho procedimiento parte de

la idea de que se pueden obtener los puntos de un contorno mediante el estudio de los píxeles "vecinos" de cada punto del contorno. Siendo P un punto del contorno, los puntos vecinos serían los ocho que comparten un lado o una esquina con dicho píxel, como se observa en la figura 2.12:

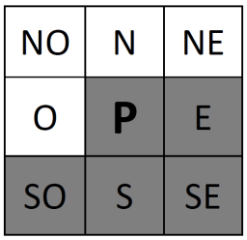


Figura 2.12 – Píxeles Vecinos

El desarrollo del algoritmo puede describirse en varias partes:

- En un principio es necesario encontrar el primer punto de un contorno. Para ello se recorren los píxeles de la imagen hasta que se encuentra un pixel "negro".
- Cuando se encuentra un punto de un contorno nuevo, éste pasa a formar parte del contorno (se establece como su primer punto). El siguiente paso es evaluar los puntos vecinos de P empezando por el punto del que veníamos. La manera en la que seleccioné el punto de comienzo es sencilla: cuando se llega del punto S o SO (en la figura) y se encuentra P, el primer punto vecino a evaluar es NO (O no forma parte del contorno, pues si lo fuera este sería el nuevo P. De esa manera se van encontrando nuevos P's que forman parte del contorno, y que se encuentran consecutivos cada uno con el anterior y el siguiente de la lista de puntos.
- El contorno se ha obtenido completamente cuando se alcanza el primer punto de la lista de nuevo.

Cabe destacar que el criterio de parada es débil, y es que para una determinada forma de contornos puede ser que no se obtengan todos los puntos deseados, como en el caso de la figura 2.13. La manera de solucionar esto es mediante el criterio de parada de Jacob, según el cual el algoritmo se detiene cuando se alcanza el píxel inicial del contorno entrando al mismo desde la misma posición desde la que se entró la primera vez.

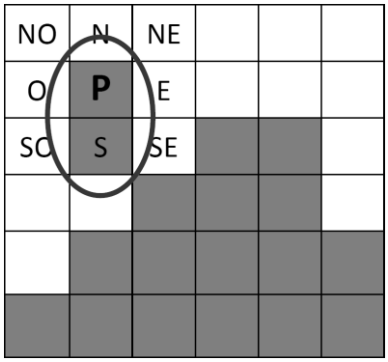


Figura 2.13 – Contorno

- Algoritmo PCA:

El Análisis de las Componentes Principales, o PCA, es un algoritmo estadístico de gran utilidad en el campo del Machine Learning, aunque sus posibilidades se extienden más allá de él. Se trata de un procedimiento que consiste en llevar a cabo una transformación de los datos a nuevas variables no correlacionadas llamadas componentes principales, o PC [20]. Cada uno de estos PC es una combinación lineal de las variables originales y todos ellos forman la base del espacio vectorial, siendo además ordenados de mayor a menor varianza.

La aplicación más común de este algoritmo es la de reducir la dimensionalidad de un conjunto multivariable en función de la importancia que cada PC tiene, medida mediante su varianza. Sin embargo, en este apartado se va a desarrollar dicho algoritmo simplificando la terminología para adaptarla al caso en concreto en el que se va a hacer uso de este: la rotación de contornos.

Como ya se ha explicado anteriormente, un contorno es un conjunto de puntos en un plano, los cuales poseen 2 coordenadas, X e Y. Así, en este problema en particular se parte de un espacio de 2 dimensiones como, por ejemplo, la situación representada en la figura 2.14:

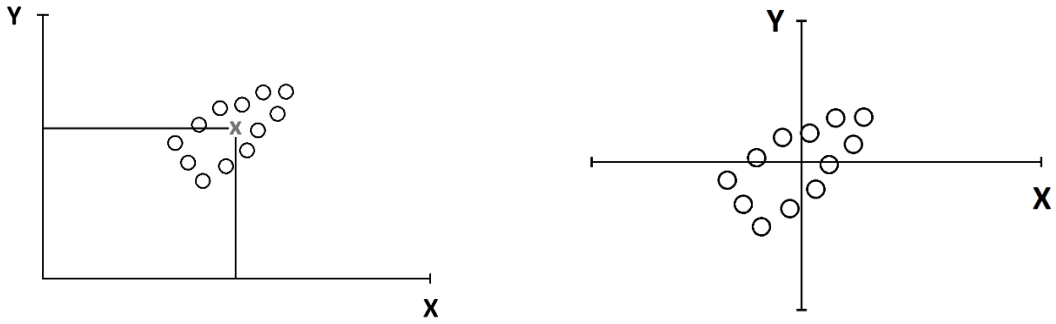


Figura 2.14 – Contorno y Punto Medio

Sea el punto en azul el punto medio del contorno, esto es, el resultado de calcular la media de todas las coordenadas X e Y. Mediante una transformación de traslación, se puede llevar dicho punto al origen de coordenadas, trasladando cada punto del contorno mediante la sustracción de las coordenadas medias calculadas.

Una vez los datos están centrados, se debe buscar una línea que represente los datos de la manera más ajustada posible, esto es, que minimice la distancia total entre cada punto y dicha recta. Así, la recta en rojo en la figura 2.15 sería la recta que mejor se ajustaría a los datos, siendo la recta azul la recta perpendicular a la misma.

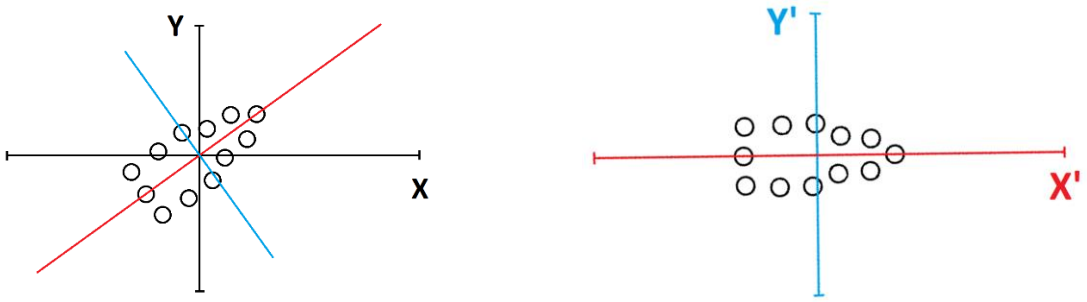


Figura 2.15 – Nuevos Ejes obtenidos mediante el Algoritmo PCA

Matemáticamente, la recta roja es la recta que minimiza la suma de los cuadrados de las distancias de cada punto a la misma. En este punto, no resulta interesante terminar el fundamento teórico del algoritmo PCA según su uso más común, la reducción de dimensiones, pero será explicado en capítulos posteriores. Llegado este punto, la utilidad de este algoritmo en la rotación de contornos es la posibilidad de establecer unos nuevos ejes que queden siempre orientados de la misma manera con relación a la geometría del contorno.

2.4. Fundamentos de Machine Learning

En este capítulo se trata el cada vez más importante campo de la inteligencia artificial, concretamente del Machine Learning, o aprendizaje de máquina. Junto con la rama de la visión artificial, ambos temas resultan cada vez más importantes en el mundo de la ingeniería.

2.4.1. Historia y Antecedentes

El Machine Learning, o aprendizaje automático, es un campo de la inteligencia artificial que posee gran importancia en diversos ámbitos de la ciencia y la tecnología, pues se trata de un método de análisis de datos que busca la automatización en la construcción de modelos estadísticos y analíticos [21]. Algunas de las áreas donde este tipo de algoritmos han alcanzado gran relevancia son la neurobiología, la ingeniería o el mundo de la bolsa.

Los comienzos del aprendizaje automático se remontan a mediados del siglo pasado. La primera red neuronal data de 1943 cuando, en una publicación científica, Warren McCulloch y Walter Pitts establecen un modelo que pretende explicar el funcionamiento del sistema nervioso, lo cual resulta tremendamente ambicioso en aquellos años [22]. Dicho modelo tomaba ciertas variables de entrada, las cuales eran multiplicadas por unos pesos escogidos aleatoriamente y sumadas, dando lugar a una salida que era comparada con un patrón que determinaba si el resultado del cálculo era positivo o negativo (salida binaria).

Posteriormente, matemáticos como el famoso Alan Turing comenzaron a establecer los conceptos en los que se asentaría este campo, como el propio Test de Turing. Este proponía que la manera de saber si una máquina era capaz de aprender por sí misma era que esta

fuese capaz de comunicarse con un humano sin que este supiera que no había otro humano como emisor del mensaje [23]. Poco después, en 1952, Arthur Samuel (IBM), al que puede verse en la figura 2.16, diseñó el primer algoritmo que jugaba al ajedrez con la suficiente capacidad de aprendizaje como para alcanzar el nivel de los jugadores profesionales.

Figura 2.16 – Arthur Samuel, 1952

Desde dichos comienzos, el campo del Machine Learning no ha hecho más que crecer, y en particular resulta de gran interés para este proyecto el desarrollo de los algoritmos de clasificación, o clasificadores.

El algoritmo de los K valores vecinos (*K-Nearest Neighbors*) fue creado en 1967 a partir de los estudios de Thomas M. Cover y P. E. Hart, estableciendo las bases del reconocimiento de patrones [24]. En sus inicios, tuvo importantes aplicaciones como en el mapeo y la optimización de rutas y recorridos.

Otro algoritmo de gran importancia, el clasificador de Naive Bayes, aun anterior al de los K-valores, pues aparece por primera vez de forma teórica a finales de los 80 con el objetivo de llevar a cabo una comparación con otros métodos de clasificación, de la mano de Cestnik y Col (1987) [25].

2.4.2. Algoritmos de Machine Learning

El Machine Learning, o aprendizaje automático, es, como su propio nombre indica, un conjunto de algoritmos y métodos encaminados a la creación de un modelo matemático/estadístico como resultado del aprendizaje o análisis de un conjunto de datos. Consta de dos fases o tareas bien diferenciadas: el entrenamiento y las pruebas [23]. En la figura 2.17 se muestra esta división en las dos fases comentadas:

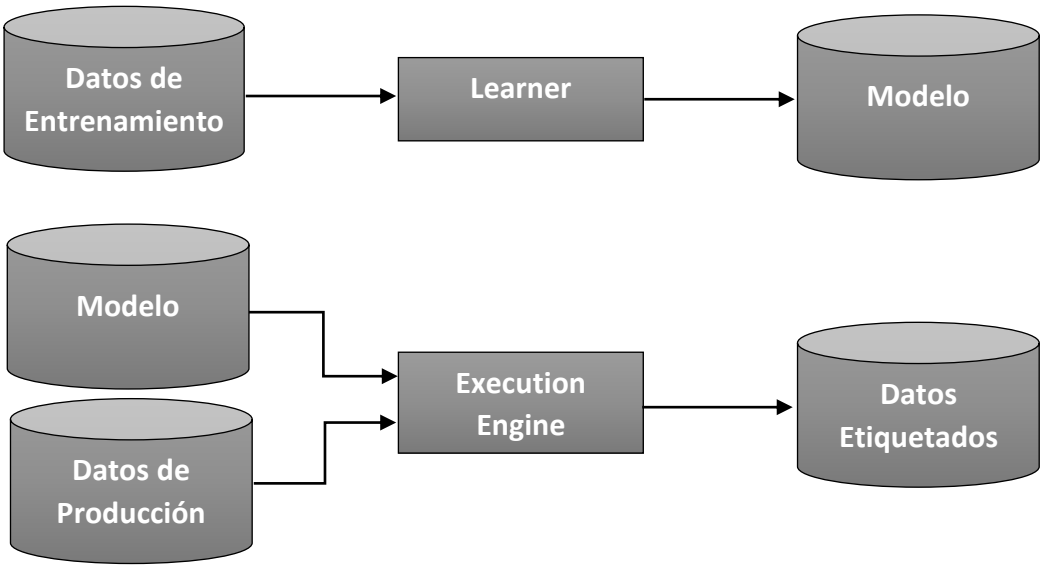


Figura 2.17 – Modelo Operativo del Machine Learning

Los algoritmos de Machine Learning pueden clasificarse en función del tipo de datos que se posea, o más bien, del etiquetado de estos de la siguiente manera:

- Aprendizaje supervisado

  El aprendizaje supervisado es un tipo de procedimiento de aprendizaje en el cual los datos que se manejan están etiquetados. Esto significa que para cada conjunto de datos de entrada aprendido hay un resultado que es conocido por este.

  El objetivo de un algoritmo de aprendizaje supervisado no es otro que establecer las normas matemáticas que permitan llegar a un resultado a partir de un nuevo conjunto de datos de entrada no perteneciente al conjunto de entrenamiento previamente comentado [26]. Por ejemplo, un algoritmo de aprendizaje supervisado podría aprender a predecir si una casa se venderá en sus primeros 3 meses de disponibilidad a partir de un conjunto de datos que pertenezcan a las ventas de los últimos años: tamaño de la casa, localización, distancia a un colegio, etc. Si, además, se sabe si cada casa contenida en esos datos fue vendida o no en los 3 primeros meses, lo que tenemos es un conjunto de datos etiquetados con el que entrenar un algoritmo de aprendizaje supervisado. En general podría decirse que el aprendizaje supervisado tiene una importante aplicación cuando se tienen ciertos volúmenes de datos históricos y se pretenden hacer predicciones sobre el futuro.

  Este proceso de aprendizaje supervisado consiste en llevar a cabo un entrenamiento controlado de manera externa: esto significa que el modelo va estableciendo un valor de salida a cada entrada dada en el conjunto de datos de entrenamiento [27]. Si la salida no coincide con la que se tiene, el modelo varía algunos valores o su propia arquitectura a fin de dar con una versión óptima que proporcione los mejores resultados de acuerdo con los que ya se conocen.

Existen dos grandes áreas dentro de los algoritmos de aprendizaje supervisado: la regresión y los clasificadores [28]:

- Regresión

La regresión lineal es una técnica ampliamente usada y conocida que permite llevar a cabo una predicción a partir de datos cuantitativos estableciendo relaciones entre estos. Mediante esta técnica se pueden realizar análisis como la influencia entre el tamaño de un solar con su precio de venta, por ejemplo. La figura 2.18 ejemplifica una recta de regresión:

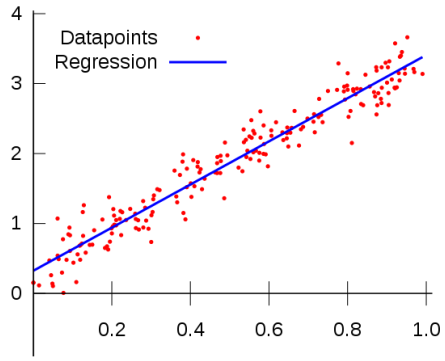


Figura 2.18 – Regresión Lineal

- Clasificadores

Los algoritmos de clasificación son aquellos destinados a proporcionar una respuesta cualitativa a partir de un conjunto de datos cuantitativos mediante el reconocimiento de patrones en los datos. De esta manera se pueden realizar gran cantidad de análisis como la determinación de si un tumor es o no maligno a partir de datos extraídos del mismo, de si un email es o no spam o de la potencialidad de defraudar de un trabajador público, como en el ejemplo representado en la figura 2.19:

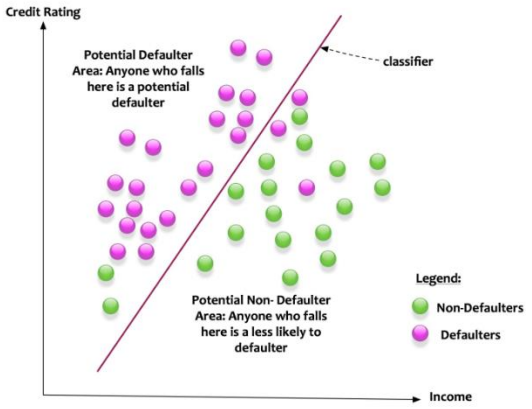


Figura 2.19 – Clasificador

- Aprendizaje no supervisado

El aprendizaje no supervisado es aquel en el que se parte de un cierto conjunto de datos que no están etiquetados, esto es, cuyas salidas no son conocidas [23]. El objetivo de este tipo de algoritmos es encontrar y definir las relaciones existentes entre los objetos del conjunto de datos mediante la creación de un modelo matemático que exprese dichas relaciones de la forma más ajustada posible.

Un ejemplo del empleo de este tipo de algoritmos podría ser el estudio de un conjunto de datos extraído de una población que permita establecer diversos grupos de interés a la hora de predecir el resultado electoral en dicha población.

El Clustering es la técnica más importante dentro del aprendizaje no supervisado y consiste en la formación de grupos o subconjuntos que no eran previamente conocidos.

- Aprendizaje semi-supervisado

El aprendizaje semi-supervisado es un tipo de técnica que está entre las dos vistas anteriormente: si bien las aplicaciones suelen ser similares a las vistas en el caso del aprendizaje supervisado, es posible que no siempre se pueda tener el conjunto de datos etiquetado al completo. Por lo tanto, este tipo de algoritmos parte de un conjunto de datos que se encuentran parcialmente etiquetado y, sin dejar de lado la tarea principal de llevar a cabo predicciones sobre los datos, cobra importancia la búsqueda y definición de relaciones entre subconjuntos pertenecientes al conjunto total.

- Aprendizaje reforzado

Este último tipo de algoritmo de aprendizaje búsqueda una mejora progresiva en el proceso mediante el análisis de los propios resultados obtenidos a partir de los nuevos datos analizados [29]. De manera sencilla, se trata de la técnica empleada en la creación de algoritmos que, por ejemplo, juegan al ajedrez. El continuo análisis de los resultados de las partidas jugadas por el programa lleva a una mejora progresiva del proceso y la optimización de los pasos a realizar por el mismo.

2.4.3. Clasificadores

En este capítulo se describe el fundamento teórico de los clasificadores utilizados en el software, entre los que se encuentran los más importantes en la actualidad.

- Support Vector Machine (SVM)

Si bien la máquina de soporte vectorial tiene diversas aplicaciones en la actualidad, este algoritmo tiene una gran importancia como clasificador desde que fuera creado por Vladimir Vapnik en 1995.

El fundamento de este clasificador, en su versión primigenia como clasificador binario, consiste en el cálculo de una frontera que separe dos subconjuntos en el espacio de dimensión dada por el número de variables [30].

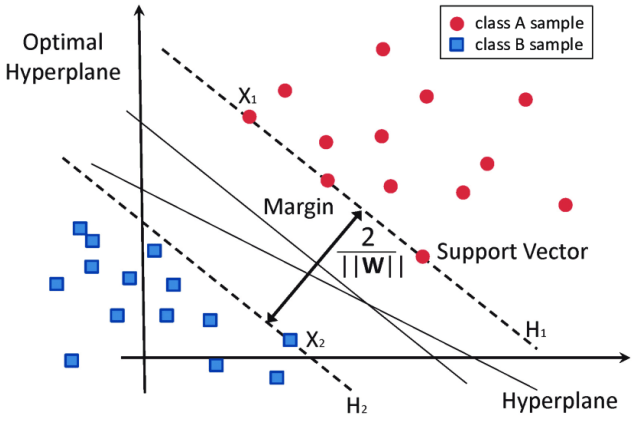


Figura 2.20 – Ejemplo de clasificación mediante SVM

En la figura 2.20 se observa un conjunto de puntos en los que se identifica a simple vista la existencia de dos subgrupos, denominados como clase A y clase B. El objetivo del SVM no es otro que calcular el hiperplano que separe de una forma óptima ambas clases, es decir, que tenga la distancia máxima con los puntos más cercanos al mismo. El algoritmo clasifica una nueva muestra en función de en qué lado del hiperplano esté representada, decidiendo la clase a la que pertenece. En este proceso se conoce como vector de soporte al vector formado por los puntos más cercanos al hiperplano frontera.

El algoritmo SVM está recogido dentro de los clasificadores lineales y a su vez dentro de los algoritmos de aprendizaje supervisado.

- K-Nearest Neighbors (KNN)

El algoritmo de los K vecinos más cercanos es un algoritmo no paramétrico que se usa en problemas de clasificación [31]. Este método busca tomar una decisión acerca de una muestra en función de las clases a las que pertenezcan las muestras más cercanas en un espacio de dimensión igual al número de entradas.

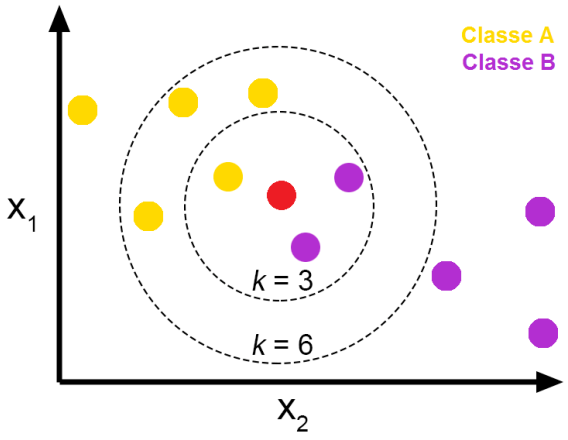


Figura 2.21 – Ejemplo de clasificación mediante KNN

En el ejemplo de la figura 2.21 la muestra roja es la que debe ser clasificada. Si se aplica el algoritmo fijando el valor de K en 3, cuando se comprueben los 3 puntos más cercanos el resultado será que la muestra pertenece a la clase B. Por el contrario, si se toman 6 puntos, es decir, se fija la K a 6, el algoritmo decidirá que la nueva muestra pertenece a la clase A.

- Naive Bayes

El algoritmo de Naive Bayes es un método de aprendizaje supervisado que ha sido de gran importancia en aplicaciones como la clasificación de documentos o búsqueda de correos spam [32]. La característica principal de este método es la posibilidad de obtener cierta información cuantitativa referente a la importancia que cada variable toma en el problema de clasificación. Esta información viene dada en forma probabilística.

El algoritmo de Naive Bayes funciona de la siguiente manera: dada una muestra representada por k valores, se trata de encontrar la hipótesis más probable que la describa. Si los k valores se expresan como $<a_1, a_2 \dots a_k>$, dicha hipótesis sería la que cumpliera que:

$$v_{MAP} = \max_{v_j \in V} P(v_j | a_1, \dots a_k)$$

Esto es la probabilidad de que si se conocen todos los valores que describen a la muestra, esta pertenezca a la clase v_j, que no es otra cosa sino el valor de la función de clasificación $f(x)$ en el conjunto finito V. Así, aplicando el teorema de Bayes:

$$v_{MAP} = \max_{v_j \in V} \frac{P(v_j | a_1, \dots a_k) p(v_j)}{P(a_1, \dots a_k)} = \max_{v_j \in V} P(a_1, \dots a_k | v_j) p(v_j)$$

Además, se puede estimar $P(v_j)$ realizando la división del número de veces que la muestra v_j aparece en el conjunto entre el número total de muestras de dicho conjunto de entrenamiento. Para llegar a la estimación del término $P(a_1, \dots a_k | v_j)$ se hace necesario recorrer todo el conjunto de entrenamiento, por lo que se suele simplificar gracias a la hipótesis de independencia condicional, buscando poder factorizar la probabilidad:

$$P(a_1, \dots a_k | v_j) = \prod_i P(a_i | v_j)$$

- Árbol de Decisión

El algoritmo del árbol de decisión en los problemas de clasificación fue desarrollado por Breiman et al. (1984) y suponen un tipo de algoritmo de aprendizaje supervisado que tiene bastante uso en la actualidad [33]. Una de las características más interesantes en este método es que las variables de entrada pueden ser categóricas o continuas.

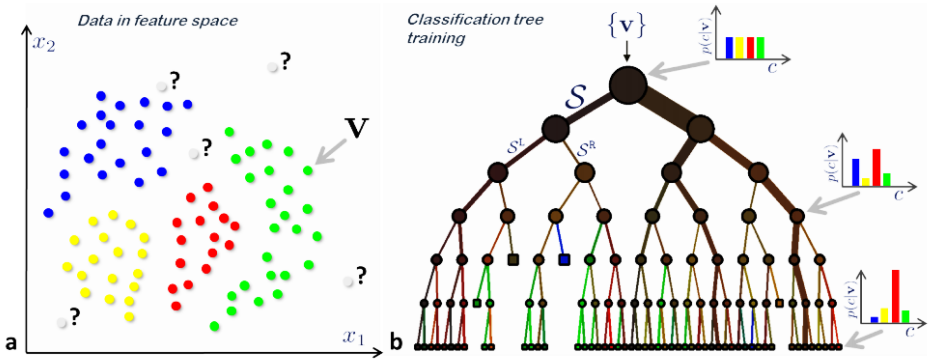


Figura 2.22 – Ejemplo de Clasificación mediante Árbol de Decisión

En este algoritmo la muestra es dividida en conjuntos homogéneos en función de la variable de entrada que resulte más significativa. El árbol se construye siguiendo un enfoque *top-down greddy approach*, esto es, de división binaria recursiva mediante el análisis de la variable que resulta más adecuada para la ramificación según el proceso de división actual. En la figura 2.22 se muestra un ejemplo de este tipo de clasificación.

2.4.4. Reducción de Dimensionalidad

Existen diversos métodos para reducir la dimensionalidad de un sistema con el objetivo de reducir el coste computacional, pero los más importantes son los dos que se han implementado en este software, explicados en este capítulo de manera teórica.

- Análisis de Componentes Principales (PCA)

El algoritmo PCA, o Principal Component Analysis, es un método de gran utilidad en el campo del Machine Learning, concretamente en tareas de clasificación, de cara a la simplificación del coste computacional gracias a la reducción de la dimensionalidad del problema en cuestión [34].

Lo que este método lleva a cabo es una proyección de un espacio original establecido a partir de las variables de entrada a otro espacio de menor dimensionalidad mediante la búsqueda de combinaciones lineales entre dichas variables. Expresado de forma matemática, el algoritmo PCA se basa en la descomposición de la matriz de covarianza de las variables de entrada a lo largo de las direcciones que dan una explicación más eficiente de las principales causas de la variabilidad de la información contenida en dichas variables. De forma sencilla podría decirse que el algoritmo busca la eliminación de aquellas variables que no influyen de manera relevante en la información que se busca procesar.

Se considera una entrada formada por n muestras de la que se tienen un total de m características, representada por una matriz $\boldsymbol{X} \in \Re^{n \times m}$, tal que:

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & \dots & x_{1m} \\ \dots & \dots & \dots \\ x_{n1} & \dots & x_{nm} \end{pmatrix}$$

Antes de continuar con el proceso es conveniente llevar a cabo un proceso de normalización para cada característica, esto es, de manera que en cada columna la media sea nula y la varianza la unidad. Esto hace que todas las variables tengan el mismo peso, eliminando el fallo de que algunas de estas con mayor varianza influyan en el proceso.

La matriz de covarianzas, R, puede calcularse como:

$$\boldsymbol{R} = \frac{1}{n-1}\boldsymbol{X}^T\boldsymbol{X}$$

Y mediante la descomposición de valores simples se tiene:

$$\boldsymbol{R} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$$

donde $\boldsymbol{\Lambda} \in \Re^{m\times m}$ es una matriz diagonal con los valores propios reales de R ($\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m \geq 0$) y cuyo valor es $\lambda_i = \sigma_i^2$. Por lo tanto, los vectores propios de R están en V, y llegar a un sistema con reducción de dimensionalidad es tan sencillo como eliminar aquellos valores propios más pequeños, hasta el valor umbral que se desee. La proyección de los datos en el nuevo espacio de dimensionalidad menor sería:

$$\boldsymbol{T} = \boldsymbol{X}\boldsymbol{P}$$

siendo P los valores propios a conservar. P es la matriz que marca las direcciones del nuevo espacio. Sus columnas son los vectores de dicho espacio, también llamados *loadings*, y los nuevos componentes del espacio T son los *scores*.

Se puede llegar al espacio de partida mediante la operación:

$$\widehat{\boldsymbol{X}} = \boldsymbol{T}\boldsymbol{P}^T$$

Además, se conoce como matriz de residuos **E** a la diferencia entre $\boldsymbol{X}$ y $\widehat{\boldsymbol{X}}$.

Finalmente, la expresión que representa el conjunto de variables de partida en función del nuevo espacio de componentes principales con los residuos queda como:

$$\boldsymbol{X} = \boldsymbol{T}\boldsymbol{P}^T + \boldsymbol{E}$$

El primer componente principal define la dirección de la mayor variabilidad de las características, lo que se traduce en una mayor suma de varianza en la matriz $\boldsymbol{\Lambda}$. Del mismo modo, los componentes principales quedan ordenados de mayor a menor, según la influencia que tienen en la variabilidad del conjunto [35]. Las componentes principales con el mayor peso en el sistema serán menos en cantidad cuanto mayor correlacionadas estén las variables de entrada.

- Linear Discriminant Analysis (LDA)

El algoritmo LDA es el método de reducción de dimensionalidad más utilizado en tareas de clasificación. Su objetivo es el de maximizar la distancia entre las proyecciones de los datos pertenecientes a clases diferentes, a la vez que se busca minimizar la distancia entre las proyecciones de la misma clase [36]. Dicho de forma sencilla, lo que se pretende es buscar

el sistema de referencia que haga a las diversas clases existentes en el conjunto de datos de entrada lo más "diferentes" posible.

Una de las características de este método es la necesidad de que los datos estén etiquetados, esto es, que se trate de un problema de aprendizaje supervisado. En el caso del aprendizaje semi-supervisado se hace necesario, por lo tanto, un preetiquetado de los datos mediante la representación de los datos, aunque esto puede llevar a errores derivados de pasar por alto las relaciones entre clases aún desconocidas o por la existencia de ruido aun no identificado.

Matemáticamente, el algoritmo LDA consiste en que, dado un conjunto de entrenamiento expresado como:

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & \dots & x_{1m} \\ \dots & \dots & \dots \\ x_{n1} & \dots & x_{nm} \end{pmatrix} = (\boldsymbol{x_1}, \boldsymbol{x_2}, \dots \boldsymbol{x_n})$$

dicho conjunto **X** puede ser dividido en cierto número C de clases independientes: y_i ($y_i \in \{1, 2, \dots C\}$). Además, mediante un esquema *1-of-K*, se tiene una representación de las pertenencias a cada clase, tal que $\boldsymbol{Y_{ij}} = 1$ cuando $y_i = j$, siendo el resto de $\boldsymbol{Y_{ij}} = 0$. El algoritmo LDA lleva a cabo una proyección lineal de $\boldsymbol{x_i}$ sobre un espacio de menor dimensión **W** tal que $\tilde{x}_i = W^T x_i$.

Con esta nomenclatura, el método LDA define tres matrices de dispersión conocidas como matriz de dispersión dentro-de-clases (1), entre-clases (2) y total (3), expresadas matemáticamente como:

$$\boldsymbol{S}_w = \frac{1}{n}\sum_{i=1}^{C}\sum_{y_j=i}(\boldsymbol{x}_j - \boldsymbol{m}_i)\,(\boldsymbol{x}_j - \boldsymbol{m}_i)^T \tag{1}$$

donde

$$\boldsymbol{m_i} = \frac{1}{n_i}\sum_{y_j=i}\boldsymbol{x}_j$$

$$\boldsymbol{S}_b = \sum_{i=1}^{C}\frac{n_i}{n}(\boldsymbol{m}_i - \boldsymbol{m})(\boldsymbol{m}_i - \boldsymbol{m})^T \tag{2}$$

$$\boldsymbol{S}_t = \frac{1}{n}\sum_{i=1}^{C}(\boldsymbol{x}_i - \boldsymbol{m})(\boldsymbol{x}_i - \boldsymbol{m})^T \tag{3}$$

donde n_i es el número de muestras de la clase i.

Además, se cumple que:

$$\boldsymbol{S}_t = \boldsymbol{S}_w + \boldsymbol{S}_b$$

Si se lleva a cabo la resta de las medias para centrar los datos, las matrices $\boldsymbol{S}_b$ y $\boldsymbol{S}_t$ pueden expresarse como:

$$\boldsymbol{S}_b = \sum_{i=1}^{C}\frac{n_i}{n}\boldsymbol{m}_i\boldsymbol{m}_i{}^T = \frac{1}{n}\boldsymbol{X}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{X}^T$$

$$\boldsymbol{S}_t = \frac{1}{n}\sum_{i=1}^{C} \boldsymbol{x}_i \boldsymbol{x}_i{}^T = \frac{1}{n}\boldsymbol{X}\boldsymbol{X}^T$$

De esta manera se llega a la transformación óptima, resultado de solucionar el siguiente problema:

$$\boldsymbol{W} = \max_{\boldsymbol{W}} \frac{tr\{\boldsymbol{W}^T \boldsymbol{S}_b \boldsymbol{W}\}}{tr\{\boldsymbol{W}^T \boldsymbol{S}_t \boldsymbol{W}\}}$$

donde **W** es la proyección final.

2.4.5. Tests de Verificación

En este capítulo se presentan los dos métodos implementados en el software Spector relacionados con la validación y prueba de los diferentes clasificadores usados. Estos algoritmos, la validación cruzada y la curva ROC, son métodos que dan una idea del rendimiento del algoritmo de clasificación.

- Validación Cruzada

La validación cruzada, o *cross-validation* en inglés, es una forma de dar una medida cuantitativa de la eficacia de un clasificador, y existen dos formas principales de llevarla a cabo.

Por un lado, el método *hold-out* es el más sencillo. El funcionamiento de este consiste en dividir el conjunto de datos de entrenamiento en dos subconjuntos, de manera que se use solo uno de ellos para entrenar el algoritmo de clasificación seleccionado [37]. El resto de los datos que, por ser parte del conjunto de entrenamiento, tienen un resultado conocido, son utilizados para poner a prueba el mismo clasificador.

Es precisamente a partir de este método que se llega al más común, el *k-folds*, mucho más eficaz en cuanto se reduce la cantidad de datos de entrada disponible y, en general, mucho más fiable por ir un paso más allá del comentado anteriormente. Este método consiste en llevar a cabo una división del conjunto de datos en k subconjuntos, de manera que se lleva a cabo la aplicación del método *hold-out* con cada uno de ellos, un total de k veces. Este proceso queda representado en la figura 2.23. El error medio obtenido de las k validaciones da una idea de la efectividad del clasificador.

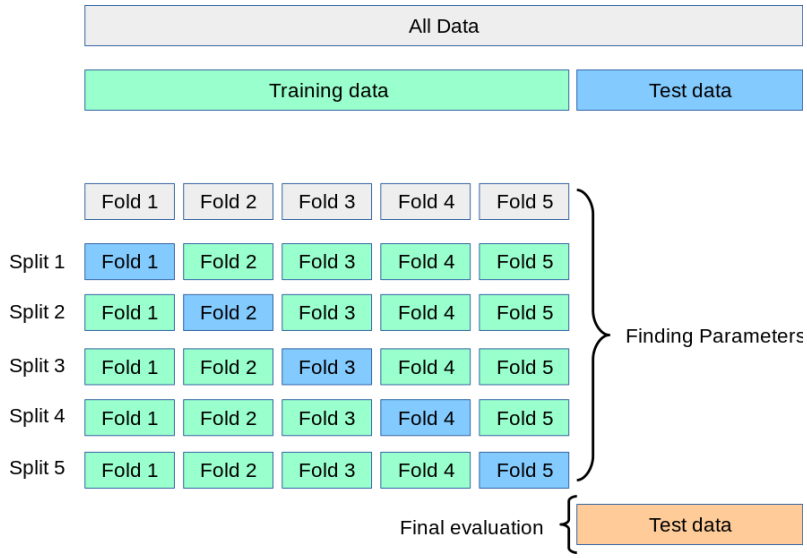


Figura 2.23 – Funcionamiento de la Validación Cruzada (*k-folds* con k=5)

- Curva ROC

La curva ROC, siglas de *Receiver Operation Characteristic*, es un método de validación para medir la actuación de un clasificador ampliamente utilizado actualmente, si bien el verdadero índice de interés es el área bajo la misma, AUC (*Area Under the Curve*) [38].

Para obtener los parámetros de interés ofrecidos por este método se parte de los datos de entrada, a partir de los cuales se crea una tabla de clasificación donde se contabilizan los datos correctos e incorrectos (etiquetas), quedando los mismos expresados mediante lo que se conoce como matriz de confusión.

Etiquetas	**Predicciones**		**Total**
	Negativo	Positivo	
Negativo	T_n	F_p	C_n
Positivo	F_n	T_p	C_p
	R_n	R_p	N

Figura 2.24 – Matriz de Confusión

En la figura 2.24, T_p y T_n son los aciertos positivos y negativos, respectivamente, mientras que F_p y F_n son los fallos positivos y negativos. Por otro lado, C_n y C_p es el número de datos negativos y positivos reales mientras que R_n y R_p son el número de predicciones negativas y positivas. N es el número total de datos de entrada.

Dicho esto, se definen los siguientes parámetros, matemáticamente:

$$Precisión\ (1 - Error) = \frac{(T_p + T_n)}{(C_p + C_n)} = P(C)$$

$$Sensibilidad(1 - \beta) = \frac{T_p}{C_p} = P(T_p)$$

$$Especificidad(1-\alpha) = \frac{T_n}{C_n} = P(T_n)$$

$$Valor\ Predictivo\ Positivo = \frac{T_p}{R_p}$$

$$Valor\ Predictivo\ Negativo = \frac{T_n}{R_n}$$

Además, se define el coste de fallo de clasificación como:

$$Coste = F_p \cdot C_{F_p} + F_n \cdot C_{F_n}$$

El coste será el parámetro a minimizar a la hora de mejorar el algoritmo de clasificación. En esta última expresión, C_{F_p} y C_{F_n} son los costes de falso positivo y de falso negativo, respectivamente.

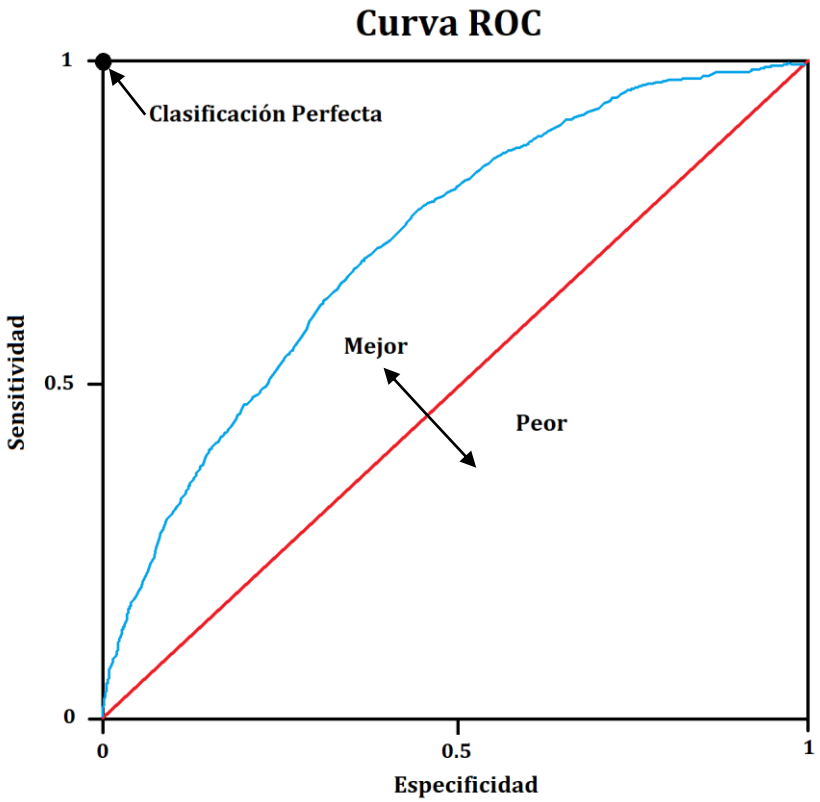


Figura 2.25 – Elementos y Ejemplo de Curva ROC

En la figura 2.25 se muestra un ejemplo de una curva ROC donde se han señalado algunos elementos previamente explicados.

3. El Software

En el presente capítulo se van a explicar cada una de las partes del proyecto, desde las tres funciones principales hasta aquellas adicionales. Sin embargo, antes de pasar a cada una de ellas se explicarán el objetivo principal y los secundarios que se pretenden con la utilización de este software.

- Inspección de piezas

Se trata del objetivo principal. Con el software Spector se puede comprobar si una pieza se ajusta geométricamente al resto de piezas correctas con las que se ha entrenado el programa. De esta manera se pueden lograr resultados acerca de si una pieza es o no correcta con una fiabilidad mucho mayor a la que tendría una simple inspección visual, y con mayor rapidez.

- Simplificación de sistemas dentro de una misma planta

Una de las cualidades de utilizar el Machine Learning es que no resulta necesario modificar el software cuando la pieza cambia, sino que el mismo software puede ser empleado para inspeccionar diversas piezas dentro de una misma línea o planta. Estoy reduce costes de aprendizaje por parte de los operarios que manejen el sistema, el único implementado para la tarea en cuestión dentro del mismo ámbito de trabajo.

- Trazabilidad

Llevar a cabo la inspección mediante este software permite llevar un registro de cada acción realizada sobre cada pieza gracias a las posibilidades que se han implementado: guardado de un log con cada acción, la hora, la parte del proceso en cuestión…

- Mejora continua

Una de las capacidades del Machine Learning es la capacidad de aprendizaje en sí. Una vez las piezas han sido inspeccionadas, los datos extraídos para ello pueden ser utilizados dentro del propio proceso de aprendizaje, dando lugar a un algoritmo más potente y ajustado. En cualquier momento del proceso puede ampliarse el conjunto de entrenamiento con nuevas imágenes de piezas que presenten fallos nuevos no existentes en el conjunto inicial con el que se contaba cuando se creó el modelo.

Dicho esto, solo queda comentar que la creación del software ha sido mediante la herramienta Visual Studio, y en lenguaje C# .NET. A continuación, se presentan en dos subcapítulos el funcionamiento del software: el primero a nivel usuario y el segundo desde el punto de vista de la programación de este.

3.1. Experiencia de Usuario

En este apartado de la memoria se desarrolla el funcionamiento del programa al completo, lo que incluye la apertura de este y el trabajo desde las diferentes fases hasta llegar al funcionamiento normal.

Cabe destacar que este capítulo expande el conocimiento del software aportado por el manual de usuario, mucho más sencillo y directo, que se muestra en uno de los anexos del proyecto.

3.1.1. Inicio del Programa

Para iniciar el software Spector se debe tener un sistema operativo de Windows de 32 o 64 bits. Además, dado que este software no gestiona la adquisición de imágenes, será necesario contar con una elevada cantidad de estas para poder realizar el entrenamiento cuando se llegue al punto de hacerlo. Del mismo modo es fundamental contar con una cantidad también alta de imágenes de piezas incorrectas (que posean defectos apreciables por el sistema).

Para iniciar el programa habrá que hacer clic sobre el icono formado por el logo creado para el software, como se aprecia en la figura 3.1:

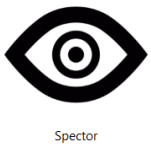


Figura 3.1 – Icono de Spector

Desde la pantalla de inicio se pueden identificar 4 zonas, señaladas en la figura 3.2:

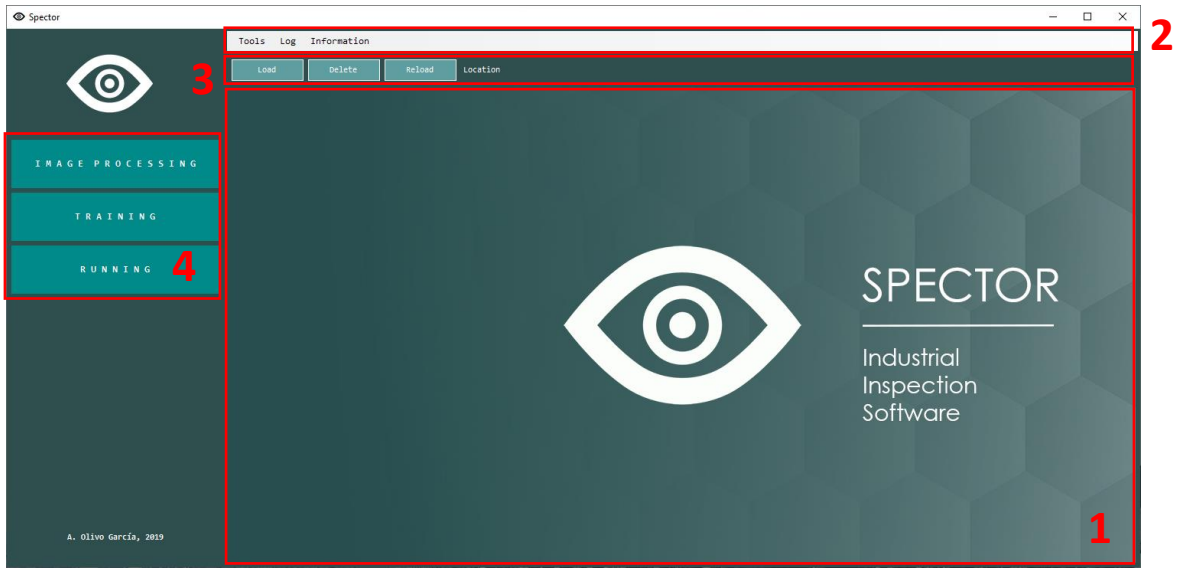


Figura 3.2 – Pantalla de Inicio

Dichas zonas son:

1- Zona de trabajo:

A medida que se avance en las distintas fases de trabajo con el software, las acciones principales se realizarán aquí. Se trata de un espacio vacío con un panel al que se acoplan los 3 formularios principales del programa.

2- Menú de herramientas y opciones:

Este menú posee tres submenús, que son los siguientes:

- Herramientas (Tools): mostrado en la figura 3.3, para llegar al formulario de gestión de modelos o para modificar los parámetros configurables que afectan a todo el programa.

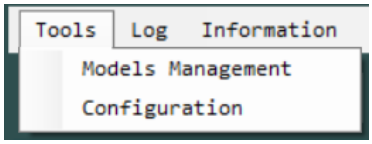


Figura 3.3 – Submenú: Tools

- Log: mostrado en la figura 3.4, para visualizar los procedimientos guardados en el log, es decir, llevar un registro de cada acción realizada en el software. También se puede exportar el archivo de texto correspondiente.

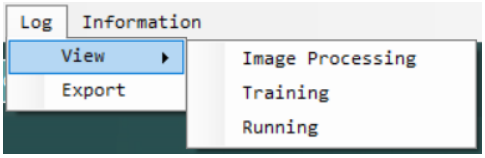


Figura 3.4 – Submenú: Log

- Información: para dar cierta información sobre el desarrollador del software, compatibilidades, etc.

3- Gestión de imágenes:

Desde esta parte de la pantalla principal se gestiona la carga y recarga de las imágenes, y se muestra en la figura 3.5.

Por un lado, si se pulsa *Load*, el usuario puede elegir la imagen que desea subir mediante un formulario de tipo de selección. Esta imagen es copiada para mantener la imagen original siempre disponible. Por medio del botón *Reload* se puede volver a mostrar dicha imagen original, cancelando todas las modificaciones realizadas sobre ella. Por último, se permite eliminar la imagen mediante el botón *Delete*, si bien para cargar una imagen nueva no es necesario que la anterior haya sido previamente eliminada.

Figura 3.5 – Gestión de las Imágenes

Cuando una imagen es cargada se muestra a la derecha la ubicación de esta, cambiando el letrero “Location” por la misma.

4- Menú de fases:

Para la selección de la fase de trabajo, lo que permite visualizar en la zona de trabajo los formularios correspondientes. Se muestra en la figura 3.6.

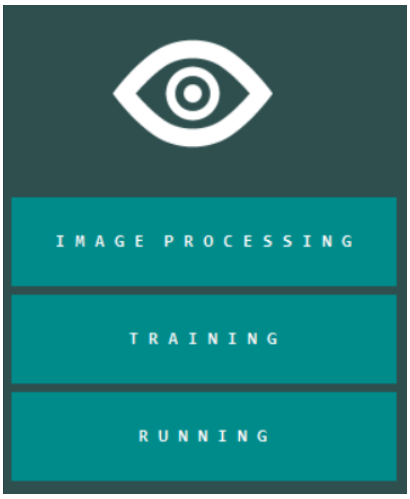


Figura 3.6 – Gestión de las Fases

La selección de cada fase de trabajo depende de en qué punto se encuentre de la inspección. El flujo de trabajo completo sería el que se muestra en el flujograma de la figura 3.7, sin embargo, es posible que una vez se haya trabajado en la edición de las imágenes (*IMAGE PROCCESING*), el software se inicie directamente en la fase de entrenamiento (*TRAINING*). Del mismo modo, cuando el conjunto de datos es suficiente, lo habitual será pasar directamente a la fase de inspección (*RUNNING*).

Figura 3.7 – Proceso Completo de Trabajo

3.1.2. Procesamiento de Imágenes

En este capítulo se detallan las diferentes acciones que pueden realizarse desde el formulario dedicado al procesamiento y edición de las imágenes.

El objetivo que se pretende en esta fase de trabajo es el de llegar a una cadena de procesos que pueda llevarse a cabo con todas las imágenes que desean procesarse, ya sea con intención de entrenar al software, como en la fase de funcionamiento normal para inspeccionar nuevas piezas.

Al igual que en la pantalla inicial, la zona de trabajo puede dividirse en 3 partes claramente diferenciadas e implementadas para las 3 funciones principales que hay que tener en cuenta. Dichas zonas se marcan en la figura 3.8:

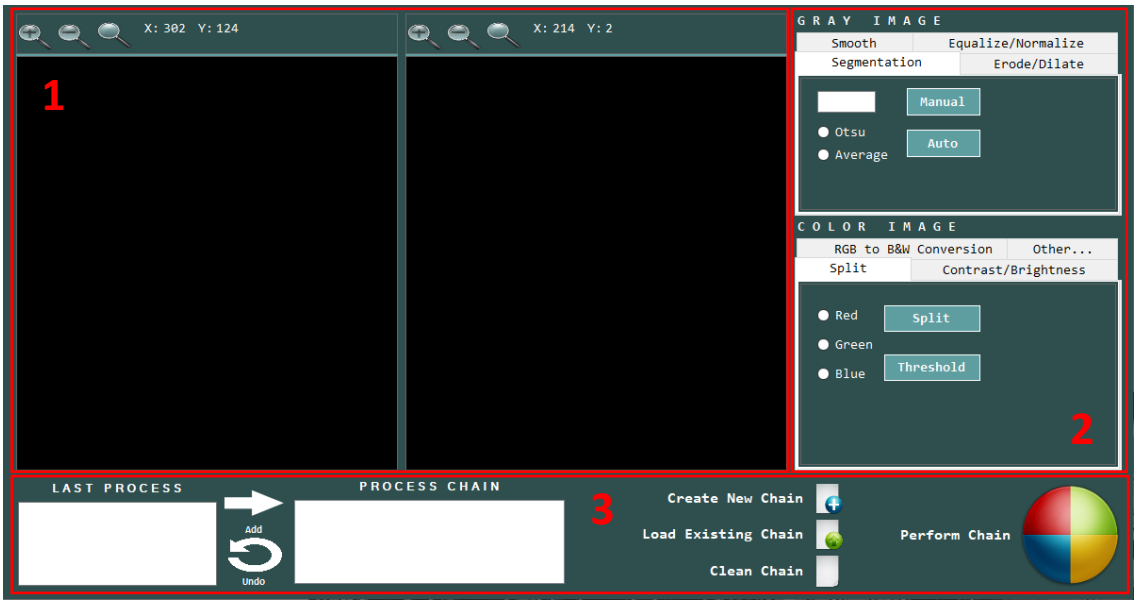


Figura 3.8 – Formulario 1: Procesamiento de Imágenes

En cada área se recogen los elementos necesarios dedicados a llevar acabo las siguientes acciones:

1- Visionado de Imágenes:

Si bien las imágenes se cargan gracias a los botones de la parte superior, presentes en todo momento, en este formulario se permite la visualización de dichas imágenes en blanco y negro y a todo color, lo que permite ir comprobando cada modificación llevada a cabo sobre ellas. Además, el control para la visualización de las imágenes permite realizar zoom sobre estas.

2- Opciones de Edición:

Se provee de diversas herramientas para llevar a cabo la modificación de las imágenes con el fin de aislar y obtener una imagen binarizada de la que se pueda extraer de manera limpia y clara el blob de la pieza que se desea analizar. Las posibilidades de edición que permite el software son las que se muestran en las siguientes tablas resumen, donde se especifica el modo de proceder para llevar a cabo la edición, y el efecto deseado sobre la imagen ejemplo de la figura 3.9:

Figura 3.9 – Imagen de Pruebas en Color y ByN

En la tabla de la figura 3.10 se muestran las transformaciones sobre la imagen en blanco y negro:

Proceso	Modo de Edición	Resultado
Erosión	Selección de Valor	
Dilatación	Selección de Valor	
Ecualización	Botón Automático	
Suavizado	Selección de Valor	
Binarización (OTSU)	Botón Automático	
Binarización Manual	Selección de Valor	

Figura 3.10 – Procesos de Edición en Blanco y Negro

En la tabla de la figura 3.11 se ejemplifican los procesos de edición que pueden realizarse sobre imágenes a color:

Proceso	Modo de Edición	Resultado
Contraste	Trackbar	
Brillo	Trackbar	
Ecualización de Histograma	Botón Automático	
Split en R	Botón Automático	
Split en G	Botón Automático	
Split en B	Botón Automático	
Threshold de 1 Canal	Botón Automático	-
Conversión ByN	Botón Automático	-

Figura 3.11 – Procesos de Edición en Color

3- Gestión cadenas de procesos:

Cada vez que se lleve a cabo una edición sobre la imagen, el cuadro de texto “Last Process” mostrará dicha acción. Por medio de los dos botones “Add” y “Undo” se puede añadir acciones a la cadena de proceso, de manera que el usuario puede diseñar una cadena de acciones y guardarla. La parte designada para esta tarea se muestra en la figura 3.12.

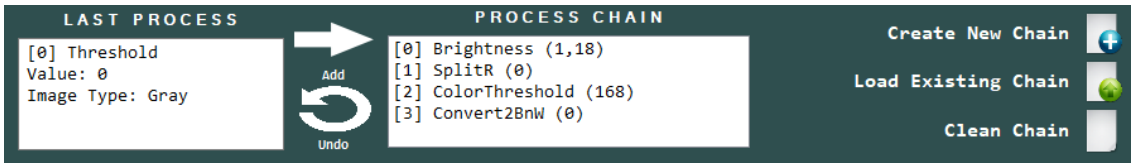


Figura 3.12 – Ejemplo de Cadena de Procesos

La cadena de procesos se guarda en formato .xml, y puede ser visualizada desde fuera de Spector gracias a la aplicación Bloc de Notas, por ejemplo. En la figura 3.13 se muestra un ejemplo de un archivo de cadena de procesos guardado, donde los procesos guardados son una ecualización, una dilatación de valor 1 y una umbralización de valor límite 134.

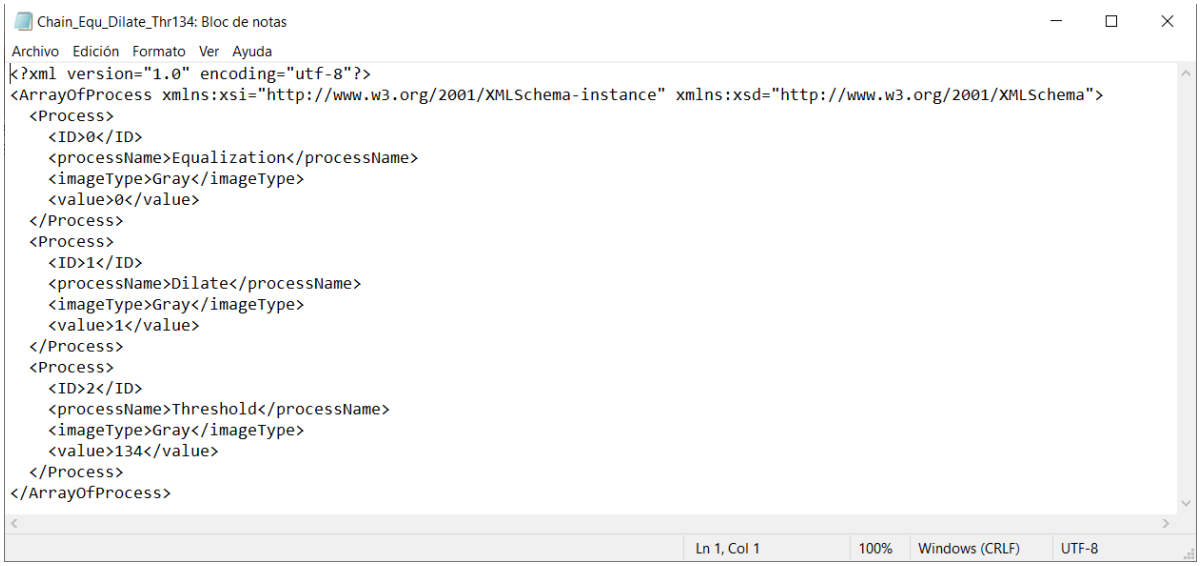


Figura 3.13 – Ejemplo de Archivo XML con Cadena de Procesos

El objetivo que se pretende con esta creación de la cadena de procesos es el de acelerar el proceso de edición de las imágenes. Partiendo de la idea de que todas las imágenes a analizar van a ser similares, la creación de una cadena que permita la identificación de la pieza a analizar de una manera óptima es lo que asegura el éxito de las siguientes fases de la inspección.

Mediante el botón inferior derecho mostrado en la figura 3.14 se puede transformar la imagen según la cadena creada/cargada.

Figura 3.14 – Botón de Ejecución de Cadena de Procesos

El flujo de trabajo en esta fase sería el que se muestra en la figura 3.15:

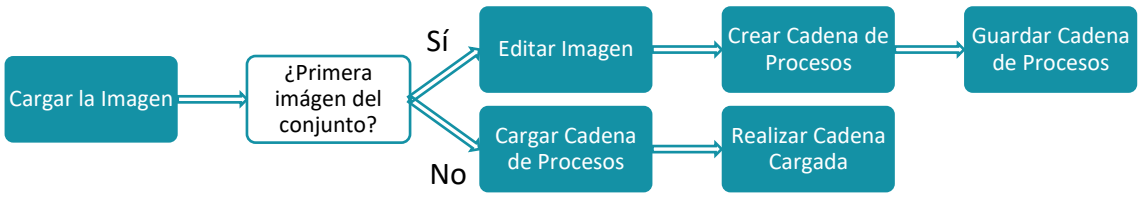


Figura 3.15 – Flujo de Trabajo de la Fase "Procesamiento de Imágenes"

3.1.3. Extracción y Creación de Datos de Entrenamiento

En este capítulo se detallan las diferentes acciones que pueden realizarse desde el formulario dedicado a la extracción de características y creación de la matriz de datos de entrenamiento.

En esta fase de trabajo se busca crear el conjunto de datos necesario para entrenar el clasificador. Para ello, será necesario definir el modelo, las variables a extraer e ir analizando cada imagen para obtener los datos. En la figura 3.16 se muestran las zonas más importantes:

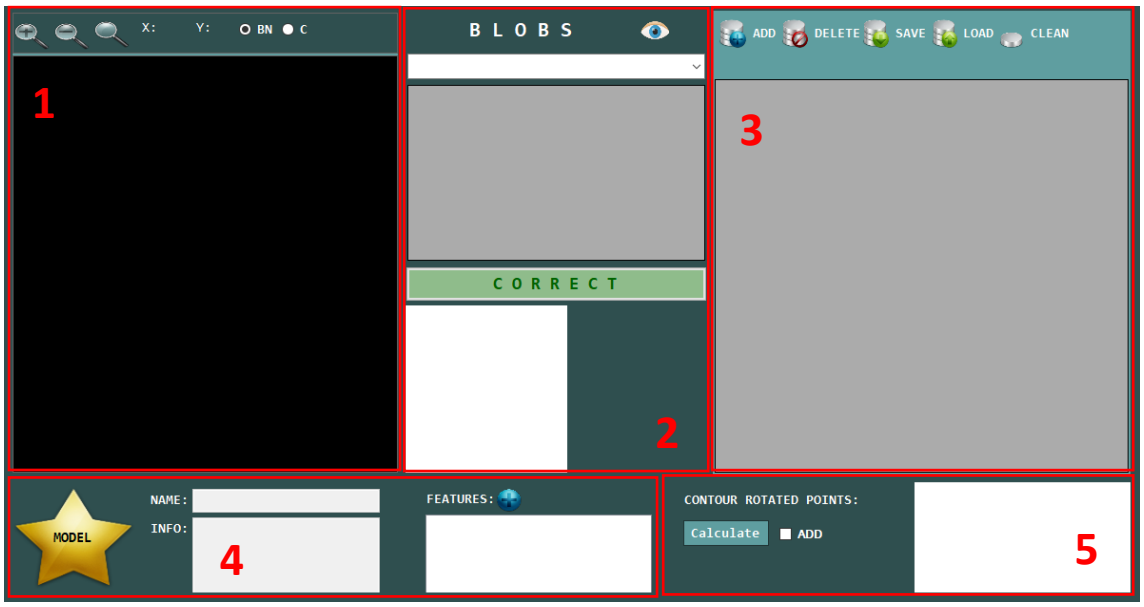


Figura 3.16 – Formulario 2: extracción de datos

Las 5 zonas que hay señaladas en la imagen superior son:

1. Visionado de Imágenes:

Para reducir el área de esta zona se ha añadido un *Radio Button* en la parte superior del control que permite cambiar entre la imagen en ByN y la imagen a color. La imagen mostrada es la que ya ha sido editada en el paso anterior.

2. Obtención de Blobs:

En este paso se deberá pulsar en el botón superior, justo el que tiene un icono de un ojo, para analizar la imagen en ByN en busca de contornos. Dichos contornos se añadirán al menú, y al seleccionar uno de ellos, los datos referentes a dicho blob se muestran en la tabla que se sitúa justo debajo. El botón "Correct/Fail" permite señalar si el contorno es correcto o falso. Además, en el la figura 3.17 se muestra el contorno seleccionado recortado.

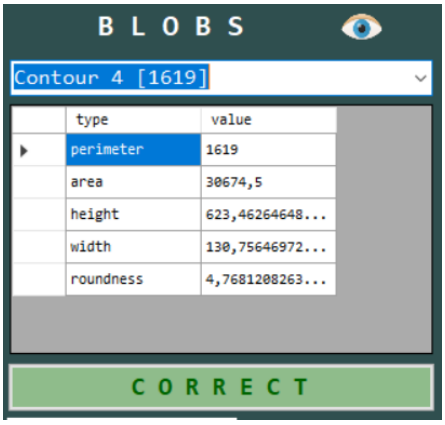


Figura 3.17 – Ejemplo de Parámetros extraídos de un Blob

Una de las cosas que se deben tener en cuenta antes de realizar esta búsqueda es que primero debe seleccionarse un modelo, pues dicho modelo contiene la información sobre qué parámetros se van a obtener de cada blob.

Cuando un contorno es seleccionado, la manera de saber que es el adecuado es mediante la visualización de este, justo debajo del botón "Correct/Fail". En la figura 3.18 se muestra un ejemplo de un blob correcto y uno derivado del ruido en la imagen

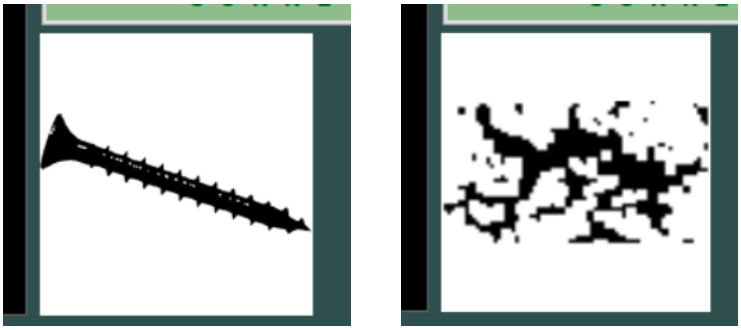


Figura 3.18 – Ejemplo de Blob Correcto (Izq.) e Incorrecto (Der.)

3. Conjunto de Datos:

En la tabla de la zona de la derecha se muestran los datos que van a ser usados para entrenar los clasificadores, mostrada en la figura 3.19. Los botones, de izquierda a derecha, permiten realizar las siguientes acciones:

a. Añadir Dato: Los datos del cuadro de la zona 2 son añadidos al conjunto.
b. Eliminar Dato: Se elimina el último dato introducido (fila).

c. Guardar Conjunto: muestra un formulario de guardado, lo que permite guardar con el nombre que se desee el conjunto de datos. Paralelamente se guarda un archivo con los resultados (Correcto/Incorrecto).
d. Cargar Conjunto: muestra un formulario de carga, lo que permite recuperar el conjunto de datos previamente guardado.
e. Limpiar Conjunto: deja en blanco la tabla, eliminando el conjunto actual.

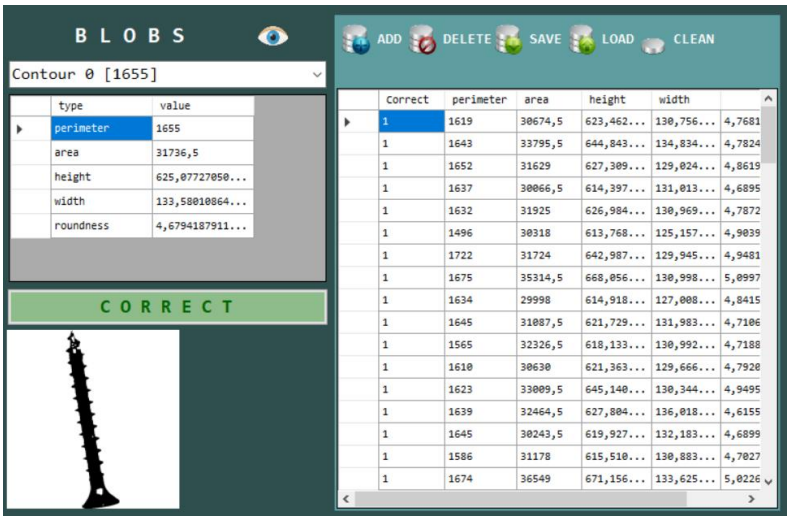


Figura 3.19 – Ejemplo de Extracción de Características y Conjunto de Datos

Al exportar o importar un conjunto de datos, este queda guardado en un archivo .xml que puede ser visualizado desde el blog de notas o cualquier otro programa que permita leer archivos de texto. En la figura 3.20 se muestra un ejemplo de un archivo que contiene un conjunto de datos:

Figura 3.20 – Archivo XML con el Conjunto de Datos

Además, junto a este archivo se crea otro del mismo formato donde se recogen las etiquetas de cada muestra, como el que se muestra en la figura 3.21. Dicho archivo tiene el mismo nombre que aquel con el conjunto de datos, pero se le añade "_RESULTS" al final.

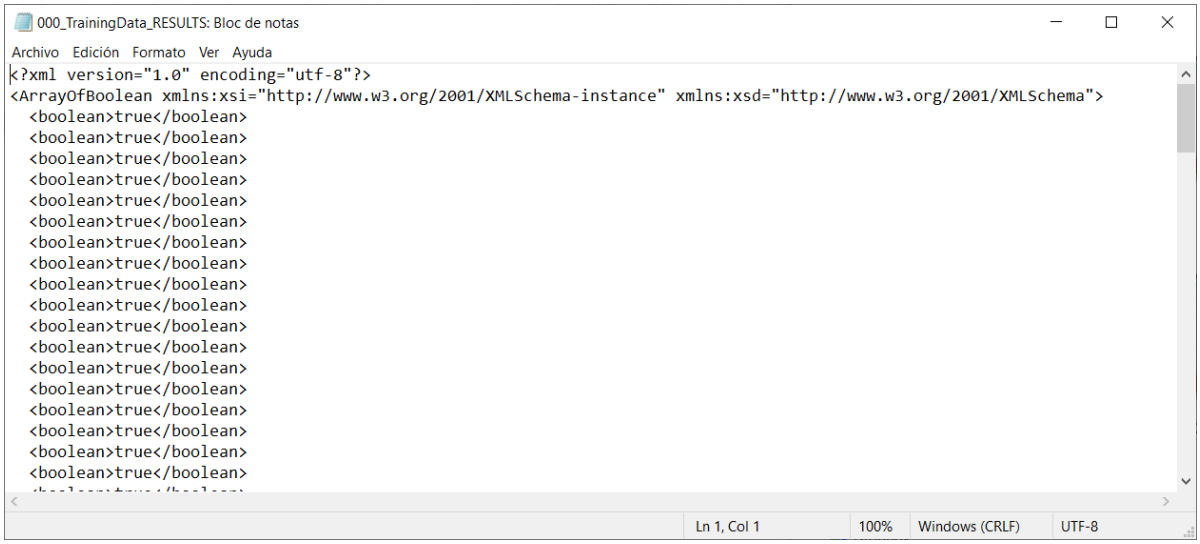


Figura 3.21 – Archivo XML con las Etiquetas del Conjunto de Datos

Ambos archivos quedarían guardados en la misma carpeta, si bien para trabajar con un conjunto de datos creado con Spector no es necesario mantener el archivo de resultados, pues las etiquetas también se guardan en el primer archivo. En la figura 3.22 se muestran dichos archivos, de extensión .xml.

Figura 3.22 – Archivos XML con los Datos de Entrenamiento

4. Gestión de Modelos:

En esta zona se selecciona el modelo con el que se está trabajando. Si bien existe un formulario para crear nuevos modelos, este será explicado más adelante. En esta zona se pulsa el botón *Model* (mostrado en la figura 3.23) para seleccionar uno.

Figura 3.23 – Botón para Seleccionar un Modelo

Lo siguiente que aparece es un pequeño formulario para la selección de entre todos los modelos creados y guardados por el usuario, tal y como el que se muestra en la figura 3.24.

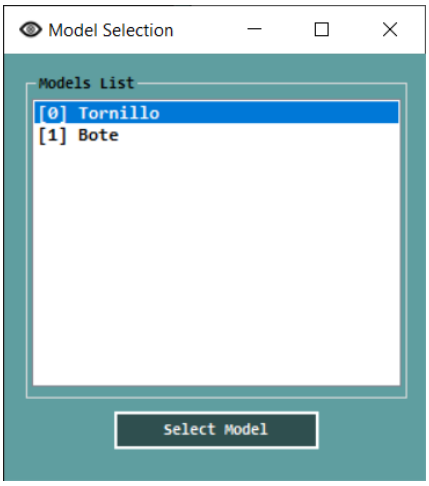


Figura 3.24 – Formulario parala Selección de Modelo

En definitiva, un modelo es una representación de la pieza que se está inspeccionando. Podría decirse que un modelo es un nombre y una descripción, acompañados de aquellas características que se calculan de cada blob, tal y como se aprecia en la figura 3.25. Cuando este ha sido seleccionado, se muestra en pantalla en todo momento, justo a la derecha del botón *Model*.

Figura 3.25 – Muestra del Modelo Seleccionado

Las características para extraer deberán seleccionarse antes de empezar a crear el conjunto de datos. Para ello se pulsa el botón con el símbolo “+”, junto a la palabra *Features*, lo que lleva al usuario a otro formulario, mostrado a continuación:

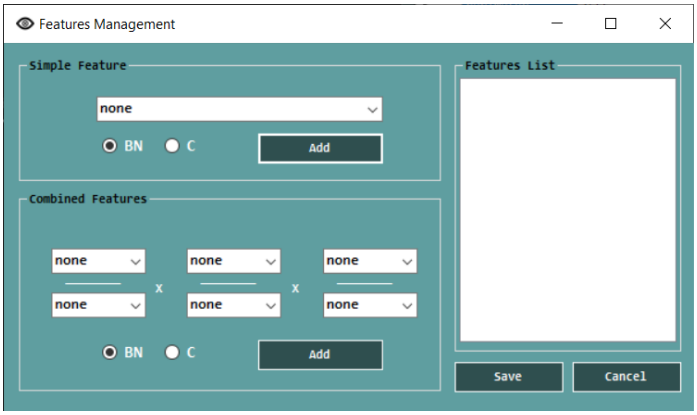


Figura 3.26 – Gestión de Características

En este formulario se permite al usuario seleccionar las características (simples o compuestas) a extraer de cada contorno. Una característica simple puede ser una longitud,

área, perímetro, etc., mientras que una característica compleja es la combinación de entre 2 y 6 características sencillas. Un ejemplo podría ser el área de la ROI (resultado del producto de la altura y la anchura de la ROI), o un coeficiente de rectangularidad (resultado del cociente del área del contorno y el área del mínimo rectángulo que lo encierra).

Las características simples que se contemplan en el software sector son las de la figura 3.27:

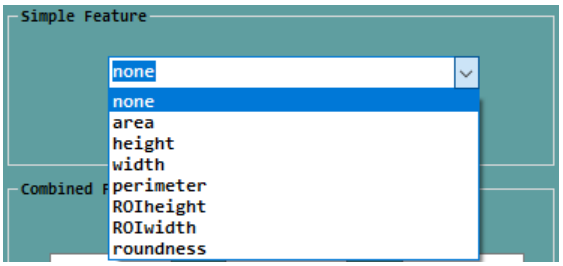


Figura 3.27 – Características Simples

Por otro lado, las características compuestas quedarían como en la figura 3.28:

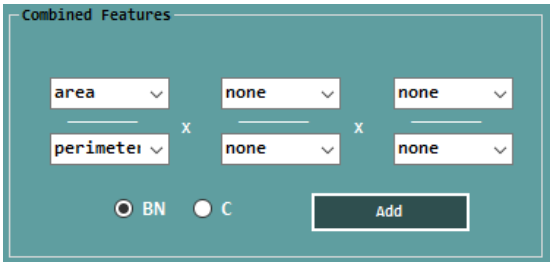


Figura 3.28 – Ejemplo de Características Compuestas

Finalmente, la lista quedaría como se muestra en la siguiente figura. Cuando se añade la característica hay que especificar sobre qué imagen se obtiene el blob (ByN o color). Esta lista de características quedará ligada al modelo y servirá para la creación del conjunto de datos de entrenamiento. Está disponible en la región mostrada en la figura 3.29.

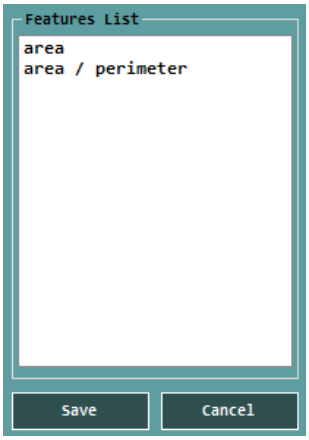


Figura 3.29 – Lista Final de Características

5. Perfil del Contorno:

Como funcionalidad extra, se puede añadir el perfil de la pieza al conjunto de datos. Para ello habrá que calcularlo (botón *Calculate*) y activar la opción *ADD* antes de añadir los datos

de la pieza actual al conjunto. En el cuadro de la derecha se muestra el contorno debidamente orientado y los puntos extraídos de su perfil, como se ve en la figura 3.30.

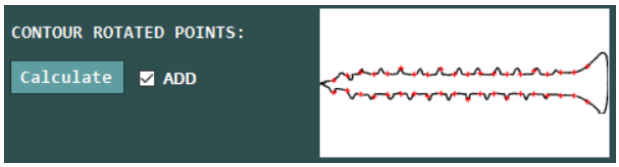


Figura 3.30 – Perfil de la Pieza

Para llevar a cabo este cálculo se computa el algoritmo PCA sobre la nube de puntos del contorno para hallar sus ejes y girarlo de modo que la pieza quede acostada. Lo que se realiza es un giro de los puntos en torno a su centro de gravedad de manera que el eje que minimiza la suma de sus distancias a todos los puntos quede horizontal.

Mediante un parámetro configurable por el usuario se define el número de puntos a medir de los perfiles superior e inferior del contorno. En la figura 3.31 se muestra la distancia a medir de cada punto, valor que pasa a formar parte sin otra transformación al conjunto de datos de entrenamiento.

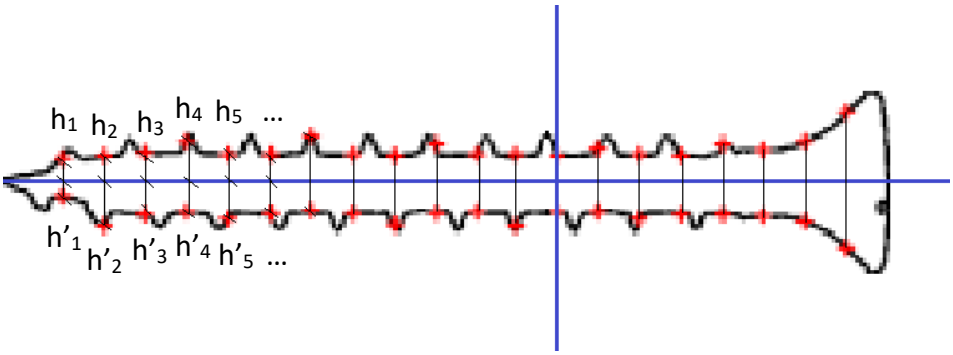


Figura 3.31 – Medidas del Perfil de la Pieza

Por último, se muestra en la figura 3.32 el flujo de trabajo que define cómo proceder en esta fase de la inspección:

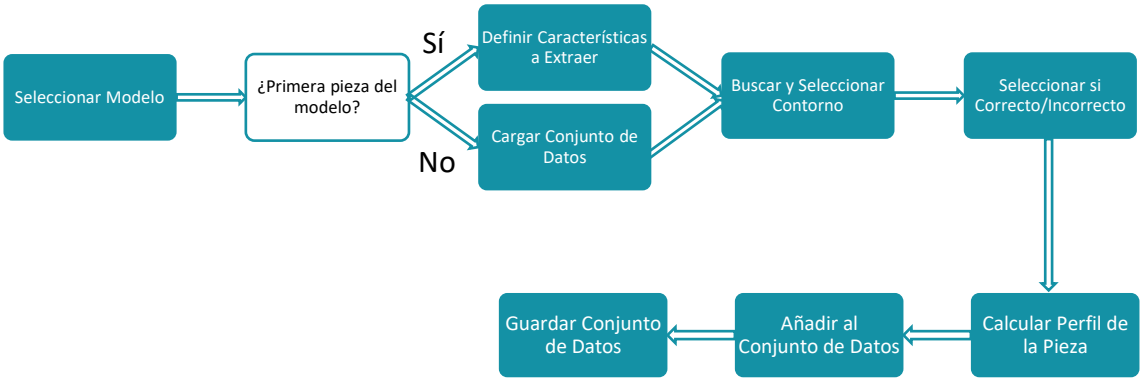


Figura 3.32 – Proceso del Conjunto de Datos

3.1.4. Pruebas e Inspección

En este capítulo se detallan las diferentes acciones que pueden realizarse desde el formulario dedicado a la selección, prueba y uso de los diferentes clasificadores.

En esta fase de trabajo se busca trabajar en la inspección de las piezas nuevas, seleccionando el clasificador que mejor funcione con el conjunto de datos del que se dispone.

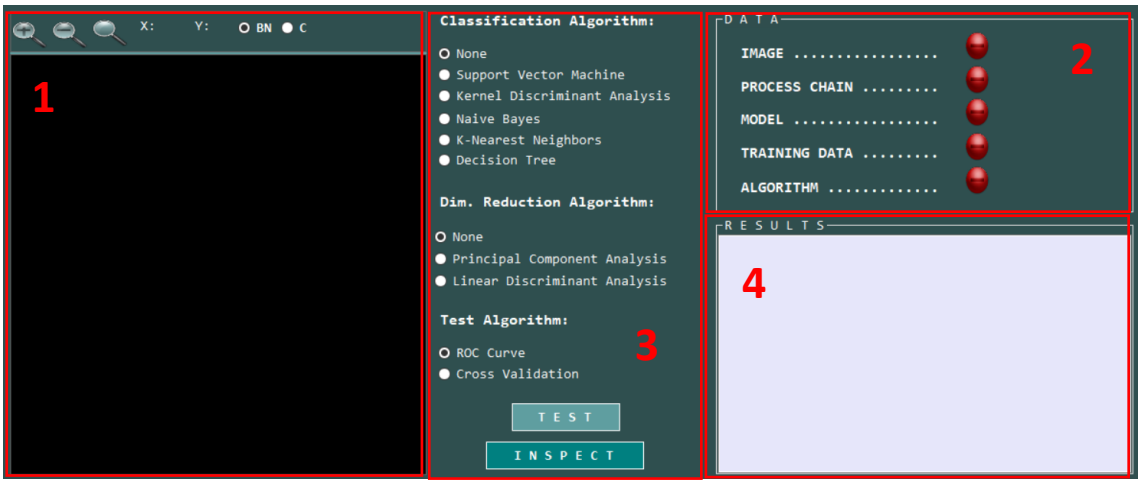


Figura 3.33 – Formulario 3: Pruebas e Inspección

Las 4 zonas que hay señaladas en la imagen de la figura 3.33 son:

1. Visionado de Imágenes:

Similar a la del formulario anterior.

2. Comprobaciones Previas:

En esta parte se comprueba que todos los elementos necesarios para llevar a cabo el análisis han sido seleccionados correctamente por el usuario. Dichos elementos son:

   a. Imagen: se carga desde la parte superior de la pantalla.
   b. Cadena de Procesos: se carga en la primera fase del proceso.
   c. Modelo: se selecciona de entre los modelos existentes en la fase de Training.
   d. Conjunto de Datos: se completa o carga en el cuadro de datos de la fase de Training.
   e. Algoritmo: se selecciona el clasificador en la zona 3 del formulario actual.

Una vez se ha cargado todo, el estado del formulario será que se muestra en la imagen de la figura 3.34:

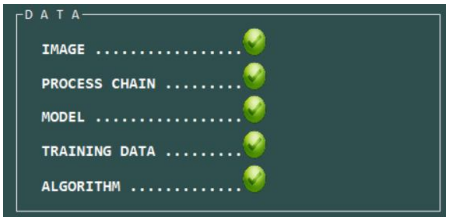


Figura 3.34 – Comprobaciones Previas a la Inspección

3. Selección de tipo de Test y Clasificador:

Esta es la zona más importante del formulario actual, pues permite al usuario elegir el clasificador que va a ser objeto de:

   a. Las pruebas previas: para lo cual será necesario seleccionar el tipo de prueba a realizar (Curva ROC o Validación Cruzada), y posteriormente clicar en el botón *Test.*
   b. La inspección: para lo cual habrá que clicar en el botón *Inspect.*

Además, se puede llevar a cabo un proceso de reducción de dimensiones. Para ello el usuario puede seleccionar el tipo de algoritmo que desee emplear, o bien seleccionar *None* para no llevar a cabo dicha reducción.

En el capítulo 2 de esta memoria se explica el funcionamiento tanto de los clasificadores como de las pruebas y la reducción de dimensionalidad.

El clasificador se seleccionará de entre las opciones que se muestran en la figura 3.35:

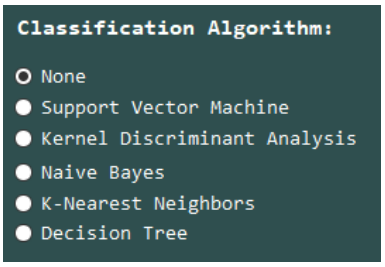


Figura 3.35 – Tipos de Clasificadores Disponibles

Para los algoritmos de reducción de dimensionalidad y los tipos de pruebas las opciones son las que se muestran en la figura 3.36:

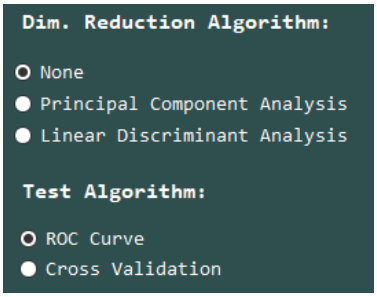


Figura 3.36 – Algoritmos de Reducción de Dimensionalidad Y Pruebas Disponibles

4. Resultados:

En el cuadro de texto de la derecha el usuario podrá visualizar los resultados tanto de las pruebas como de la propia inspección. A continuación, se muestran dos ejemplos de resultados durante la puesta a prueba (figura 3.37) y la inspección (figura 3.38):

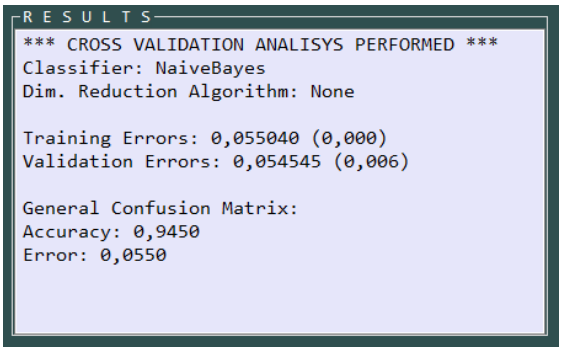


Figura 3.37 – Resultado de Prueba de Validación Cruzada sobre Clasificador Naive Bayes

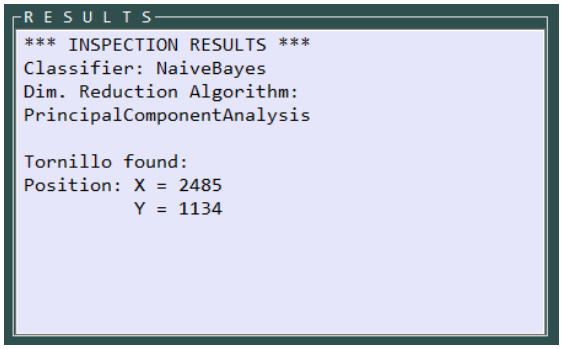


Figura 3.38 – Resultado de Inspección con Clasificador Naive Bayes y Reducción de Dimensionalidad mediante PCA

Por último, se muestra el flujo de trabajo para la última fase de inspección en la figura 3.39:

Figura 3.39 – Proceso de Test e Inspección

3.1.5. Otras Funcionalidades

En este capítulo se habla de las características extra del programa que complementan o amplían las capacidades del software más allá de la propia inspección.

- Creación de Modelos

Como ya se ha explicado, los modelos deben ser creados por el usuario para representar la realidad dentro del programa. Un modelo se crea mediante la asignación de un nombre y un pequeño texto informativo sobre la pieza que se va a inspeccionar. Además, cada modelo tiene asignado el conjunto de características, aunque este pase se realiza directamente desde la fase de entrenamiento. El formulario de gestión de modelos se muestra en la figura 3.40.

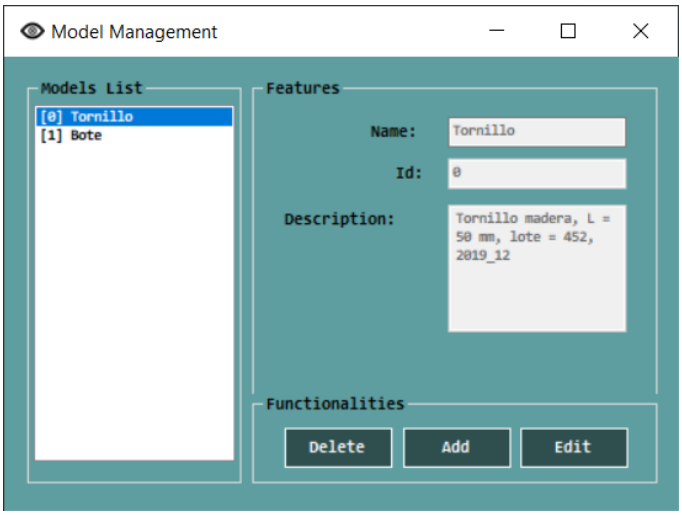


Figura 3.40 – Formulario de Creación de Modelos

Estos modelos se guardan en un archivo llamado *Models* de formato .XML, el cual se modifica al añadir modelos, editarlos o añadir el conjunto de características deseado. En la figura 3.41 se muestra un ejemplo de este archivo:

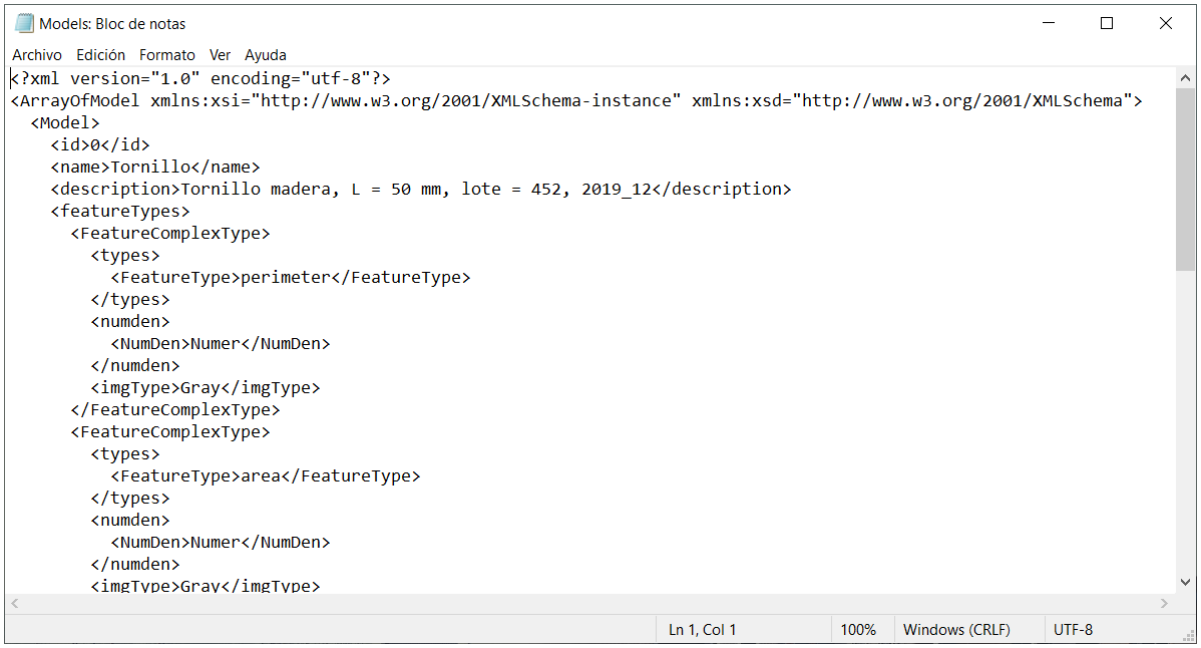


Figura 3.41 – Archivo XML con Información de los Modelos

- Log

Una de las partes contempladas en los objetivos de este proyecto era proporcionar la funcionalidad de llevar a cabo un log de cada acción y resultado derivado del uso del programa. Para ello, se da la posibilidad de visualizar y exportar hasta tres archivos de texto en los que se van registrando cada acción y resultado tomado.

Para la primera parte del programa, la de edición y procesamiento de imágenes, el log guarda un registro de cada transformación realizada sobre la imagen. Además, se guardan las acciones relacionadas con las cadenas de procesamiento: importación, exportación o

realización de las mismas. A continuación, en la figura 3.42, se muestra una captura del archivo derivado de esta parte, donde se aprecian algunas de estas instrucciones guardadas con su fecha y hora.

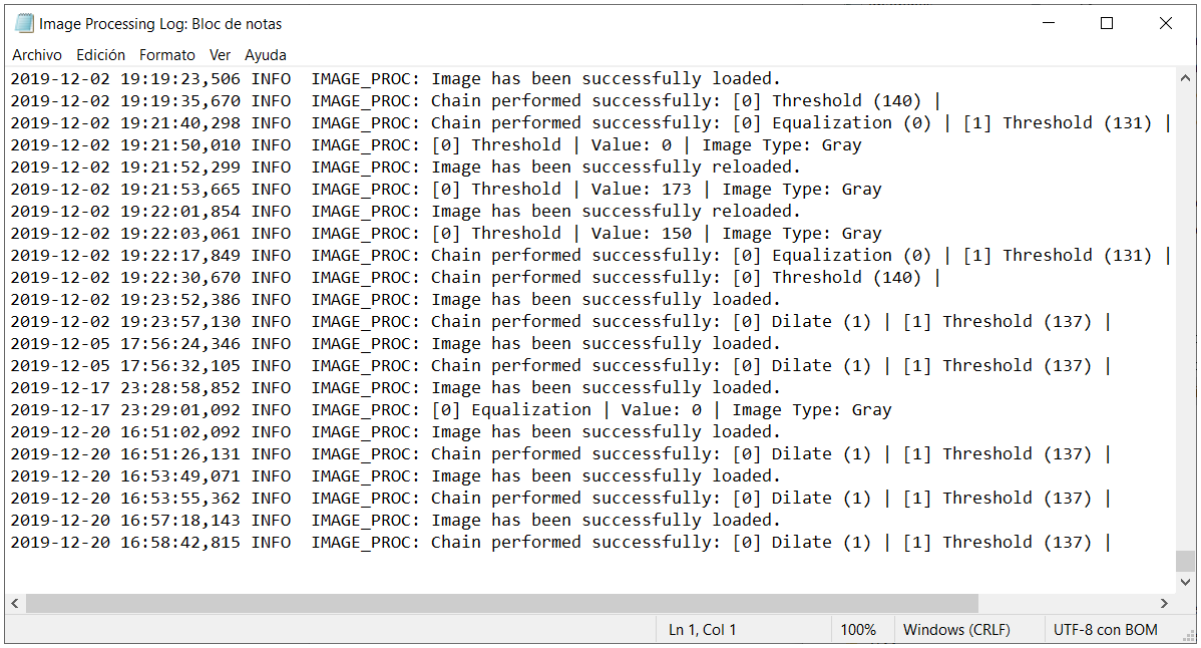


Figura 3.42 – Archivo XML con el Log de la Fase de Procesamiento

Del mismo modo, en el caso de la segunda fase, la de entrenamiento, se tendría constancia de la búsqueda de contornos, así como de la carga y guardado del conjunto de entrenamiento. En la figura 3.43 se muestra cómo quedaría el archivo de texto derivado de esta etapa.

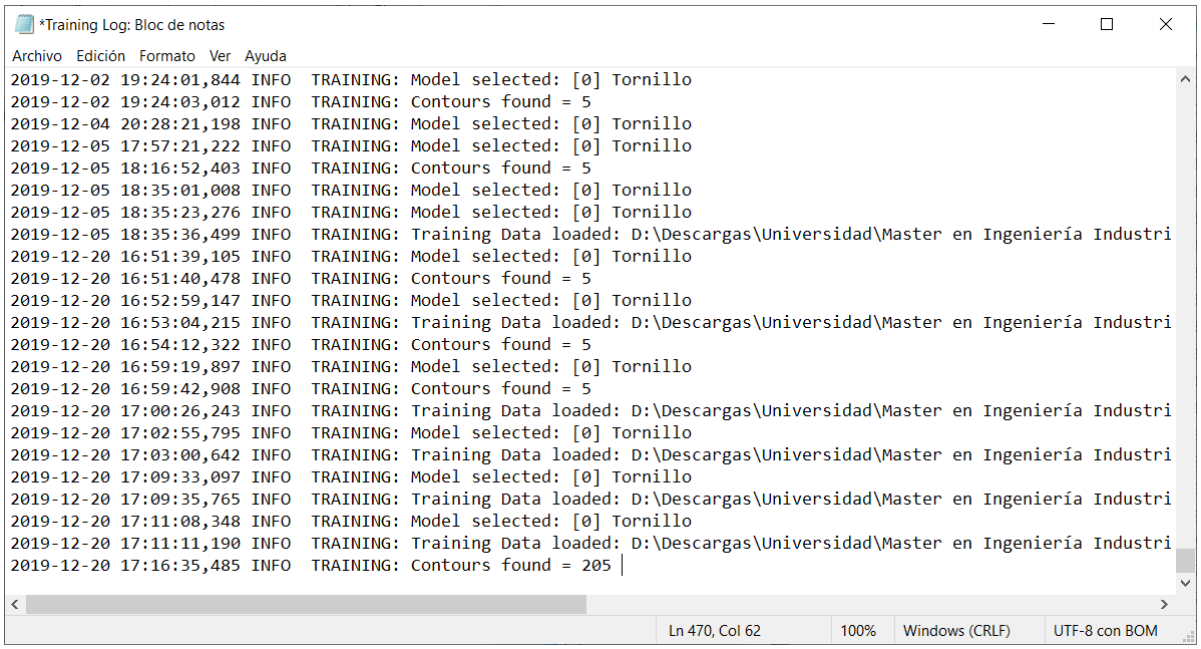


Figura 3.43 – Archivo XML con el Log de la Fase de Entrenamiento

Por último, durante la fase de inspección se guardaría constancia de cada resultado obtenido, ya sea durante la realización de pruebas (curva ROC o Validación Cruzada) como durante la fase de inspección, tal y como se muestra en la figura 3.44.

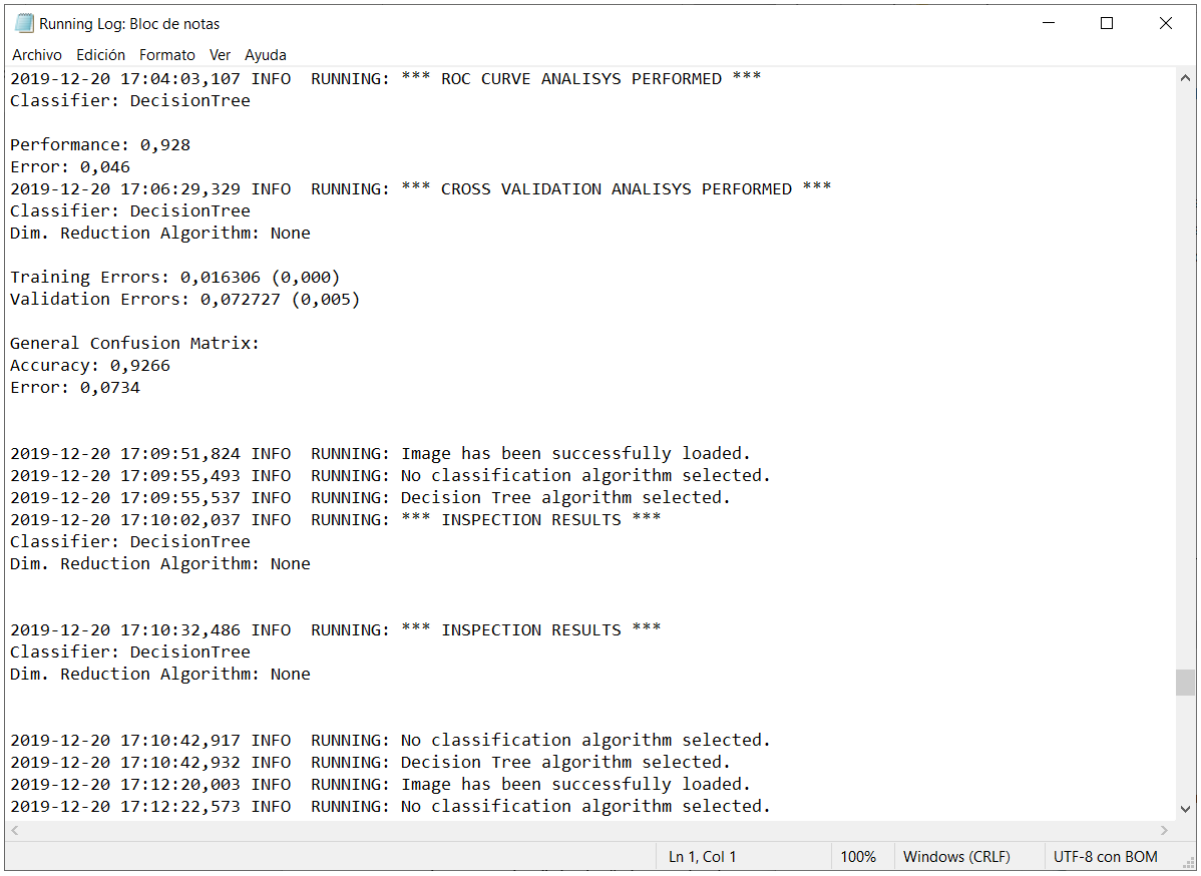


Figura 3.44 – Archivo XML con el Log de la Fase de Inspección

- Parámetros Configurables

Como es normal, la cantidad de parámetros que se pueden modificar a lo largo del funcionamiento del programa es relativamente alta, y en un programa que se caracterice por su sencillez y su carácter intuitivo no resulta nada apropiado introducir cuadros de texto y botones para introducir todos estos parámetros en la misma pantalla principal de cada fase.

Es por esto por lo que resulta de gran utilidad un formulario extra donde todos estos parámetros están debidamente ordenados y puedan modificarse de una vez. Y es para ello que se ha creado el formulario que se muestra en la figura 3.45.

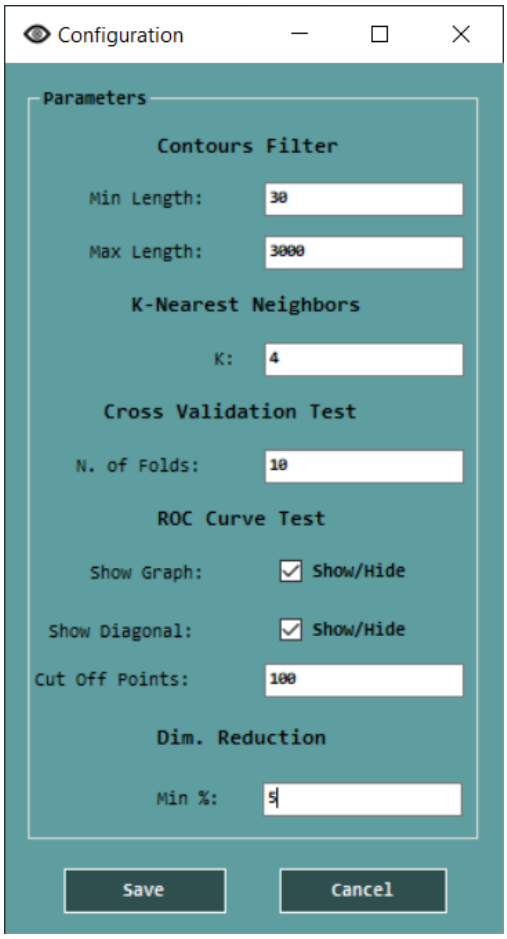


Figura 3.45 – Formulario de Configuración de Parámetros

Los parámetros que pueden ser configurados son los siguientes:

- Filtro de contorno: permite establecer un valor mínimo y máximo para los contornos encontrados en la imagen, lo que dota de mayor limpieza al proceso de búsqueda de blobs.
- Parámetro K del clasificador KNN: establece cuántos valores vecinos se tienen en cuenta en el proceso de clasificación de los K Valores Vecinos, tal y como se explica en el capítulo de fundamento teórico.
- Parámetro de Validación Cruzada: establece el número de “folds” para dicha prueba, también definido y explicado en la parte teórica del proyecto.
- Parámetros de la Curva ROC: establece el número de puntos que forman la curva, así como si se desea o no mostrar la curva, además de mostrar el resultado numérico (área bajo la misma).
- Porcentaje de influencia: establece un valor umbral por debajo del cual un parámetro es eliminado del conjunto de características a tener en cuenta antes del entrenamiento del clasificador que se está utilizando.

3.2. Algoritmia

El objetivo de este capítulo es arrojar luz sobre ciertas partes del código fuente que hay tras el software Spector, evitando copiar partes grandes del mismo, pero explicando aquellas que resulten de cierto interés por su relación con los conceptos teóricos desarrollados en el capítulo 2.

3.2.1. Estructuras y Clases

El lenguaje de programación C# es un lenguaje que entra dentro de la definición de la programación orientada a objetos. Este tipo de paradigma permite al programador establecer una representación de la realidad mediante la creación de entidades denominadas "objetos", los cuales tienen diversos comportamientos (métodos) y estados (atributos).

Las clases en C# son los tipos más importantes, y de entre sus muchas características está la posibilidad de aplicarles herencia y polimorfismo, lo que permite expandir y especializar clases base (padre). Precisamente siguiendo este fundamento se han implementado en Spector diversas clases para gestionar algunos elementos del programa, que son las que se desarrollan a continuación:

- Cadena de Procesos

El objetivo es tener una clase serializable que permita exportar los datos a un archivo .XML de manera que pueda ser de nuevo importado por el usuario cuando este lo desee. Esta clase contaría con 4 atributos: un número identificador que indica el orden de los procesos en la cadena (de tipo entero), el tipo de proceso en cuestión (enumerado), el tipo de imagen al que afecta el proceso (la de blanco y negro o la de color, de nuevo de tipo enumerado) y el valor, si lo hubiere, relacionado con el proceso (double). Este último atributo depende del tipo de proceso. Si se tiene, por ejemplo, un proceso de umbralización, el valor sería el límite de la propia umbralización. Para el caso de otros procesos donde no se aplica un valor concreto, como la ecualización o la conversión a ByN, dicho parámetro puede quedar como 0.

En la figura 3.46 se muestra un esquema donde quedaría representado la estructura comentada justo arriba:

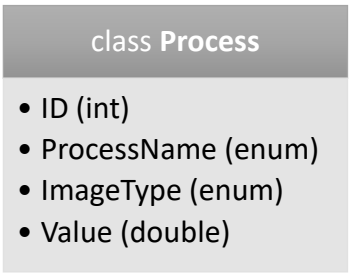


Figura 3.46 – Clase "Process"

- Modelos y Características

La clase "Model" es una de las clases más importantes del software, pues cuando crea un modelo tiene que contener toda la información relacionada con el mismo. Esto incluye desde el nombre y la información de lo que se quiere modelizar, hasta las características que van a extraerse de cada imagen.

Con respecto a las características cabe destacar que la posibilidad de seleccionarlas de tipo simple y compuesto hace un poco más complicado la definición de la estructura de clases. En este punto, se hace necesario el uso de la herencia. En la figura 3.47 se muestra el esquema que define dicha estructura:

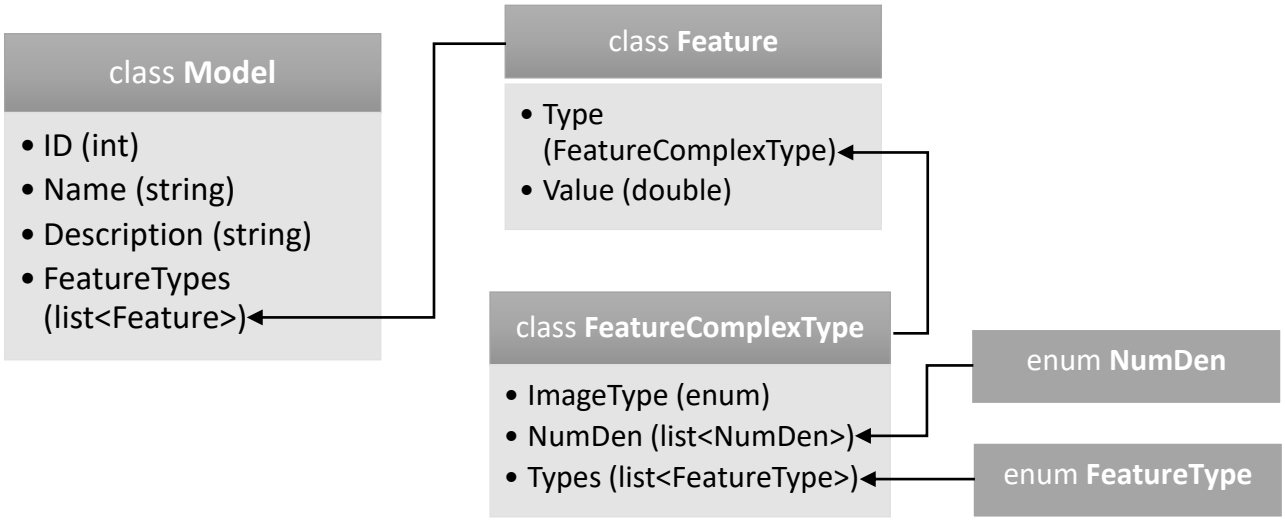


Figura 3.47 – Clases de Modelos y Características

Como se mencionaba anteriormente, la estructura de clases que definen a las características (features) está relacionada con la existencia de las características complejas. El enumerado "FeatureType" hace referencia al tipo de característica: altura, anchura, área, perímetro, etc. La clase "FeatureComplexType" es la característica creada por el usuario como combinación de características, por lo que es necesario conocer la lista de características simples que la componen y si éstas van multiplicando o dividiendo (de ahí el enumerado "NumDen", que hace referencia a si la característica va en el numerado o en el denominador de la fórmula creada). Por último, la clase "Feature" contiene el objeto que define la fórmula de características sencillas y el valor (double), resultado de realizar la operación.

Esta forma de programar la gestión de las características hace que sea relativamente sencillo expandir las posibilidades del software. Añadir nuevas características sencillas sería tan fácil como aumentar el tamaño del enumerado "FeatureType" y programar aquellas funciones que extraigan dichas características a partir de los contornos.

- Conjunto de Entrenamiento

La clase serializable "TrainingData" es la que permite definir el conjunto de entrenamiento, y estaría definida según el esquema de la figura 3.48:

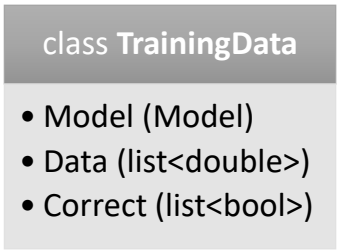


Figura 3.48 – Clase "Training Data"

Lo que se necesita guardar con cada conjunto de entrenamiento son la matriz de datos, el vector de resultados (etiquetas) y el modelo que contiene la información de qué es cada columna de la matriz. La matriz de datos no es más que un array de dos dimensiones (de valores tipo double) cuyo número de columnas es el número de características más la cantidad de medidas extraídas del perfil del contorno (si se diera el caso), mientras que el número de filas es el número de muestras del conjunto de entrenamiento.

3.2.2. Funciones de Librerías

En el desarrollo del software Spector se utilizan fundamentalmente dos librerías: OpenCV y Accord. Una librería no es más que un conjunto de funciones disponibles para cada lenguaje de programación sobre un campo, de manera más o menos específica.

La primera mencionada, OpenCV, no se encuentra disponible para C#, sino que está programada en C++. Esto hace necesario la utilización de un "wrapper", una especie de traductor que permite llamar a las funciones de la librería desde un entorno de programación en C#, conocido como Emgu. Gracias a esta librería se pueden usar gran cantidad de funciones dirigidas al campo de la visión por computador.

En la figura 3.49 se muestra una tabla donde se nombran y explican brevemente las funciones utilizadas de dicha librería:

Emgu CV (OpenCV)	
CvInvoke.Threshold()	Función para realizar la umbralización. Acepta imágenes en color y en blanco y negro. Lleva a cabo el proceso a partir de un valor dado, o bien permite seleccionar diversos métodos de umbralización típicos como Otsu.
CvInvoke.Erode()	Función que realiza una erosión de la imagen un número determinado de veces (iteraciones) determinado. Acepta imágenes en ByN o a color, donde realiza la erosión para cada color.
CvInvoke.Dilate()	Función para realizar la dilatación de una imagen, con las mismas observaciones que en el caso de la erosión.
CvInvoke.EqualizeHist()	Función para llevar a cabo la normalización del histograma: normalización del brillo e incremento del contraste de la imagen.

CvInvoke.Split()	Función para realizar la separación de canales de una imagen a color, de manera que se tienen tres matrices, una para cada uno de los canales R, G y B. Cada una de las matrices pasa a comportarse como una imagen en ByN, es decir, de un solo canal.
CvInvoke.FindContours()	Esta función permite obtener los contornos de una imagen binaria, devolviendo además la cantidad de ellos obtenida. Además, permite clasificar los contornos en función a si estos son internos o externos.
CvInvoke.cvCopy()	Permite copiar una imagen, o parte de ella, a otra de destino del tamaño de la porción seleccionada o de la imagen completa.
CvInvoke.ContourArea()	Función para obtener el área de un contorno, es decir, el número de píxeles contenidos dentro del mismo.
CvInvoke.MinAreaRect()	Función para obtener el mínimo rectángulo que contiene a un contorno. De esta manera se puede obtener la altura y anchura de la pieza.

Figura 3.49 – Tabla Resumen de Funciones de OpenCV

La segunda mencionada, Accord, es una librería que está dirigida principalmente al Machine Learning. Esta librería tiene un nivel adecuado para aplicaciones profesionales, incluyendo funciones de visión por computador, procesamiento de señal o aplicaciones estadísticas, entre otras.

Las funciones utilizadas de esta librería son aquellas relacionadas con los clasificadores, los tests y los algoritmos de reducción de dimensionalidad. A continuación, se explican los códigos programados en torno a cada uno de estos procesos.

Para el caso de los clasificadores, se han programado funciones que toman como entrada los datos de entrenamiento (inputs), las etiquetas (outputs) así como los nuevos datos a analizar (newData).

- Support Vector Machine

```
public bool[] trainSVM(double[][] inputs, int[] outputs, double[][] newData)
        {
            bool[] answers = new bool[newData.Rows()];

            // Se crea el objeto que aprenderá los datos
            var learn = new SequentialMinimalOptimization<Gaussian>()
            {   UseComplexityHeuristic = true,
                UseKernelEstimation = false
            };

            // Se entrena el clasificador mediante el método Learn
            SupportVectorMachine<Gaussian> svm = learn.Learn(inputs, outputs);

            // Se clasifican los nuevos datos para obtener una solución (0 o 1)
            answers = svm.Decide(newData);
            return answers;
        }
```

- Naive Bayes

```
public bool[] trainNB(double[][] inputs, int[] outputs, double[][] newData)
        {
            bool[] answers = new bool[newData.Rows()];
            int[] answersInt = new int[newData.Rows()];

            // Se crea el Algoritmo Naive Bayes con Distribución Normal
            var teacher = new NaiveBayesLearning<NormalDistribution>();

            // Se entrena el algoritmo
            var nb = teacher.Learn(inputs, outputs);

            // Se clasifican las muestras, que dan como resultado un array de int
            answersInt = nb.Decide(newData);

            // Se transforman los resultados a un array de bool
            for(int i = 0; i < answers.Length; i++)
            {
                if (answersInt[i] < 0.5) answers[i] = false;
                else if (answersInt[i] >= 0.5) answers[i] = true;
            }
            return answers;
        }
```

- K-Nearest Neighbors

```
public bool[] trainKNN(double[][] inputs, int[] outputs, double[][] newData)
        {
            bool[] answers = new bool[newData.Rows()];
            int[] answersInt = new int[newData.Rows()];

            // Se crea el algoritmo de KNN con K igual al parámetro introducido
            // por el usuario en el formulario de configuración
            var knn = new KNearestNeighbors(k: _configuration.K_KNN);

            // Se entrena el algoritmo
            knn.Learn(inputs, outputs);

            // Se clasifican los nuevos datos
            answersInt = knn.Decide(newData);

            for (int i = 0; i < answers.Length; i++)
            {
                if (answersInt[i] < 0.5) answers[i] = false;
                else if (answersInt[i] >= 0.5) answers[i] = true;
            }
            return answers;
        }
```

- Árbol de Decisión

```
public bool[] trainDecisionTree(double[][] inputs, int[] outputs, double[][] newData)
        {
            bool[] answers = new bool[newData.Rows()];
            int[] answersInt = new int[newData.Rows()];

            // Se opta por el algoritmo de Árbol de Decisión C4.5:
```

```
            C45Learning teacher = new C45Learning();

            // Se entrena el algoritmo de clasificación
            var tree = teacher.Learn(inputs, outputs);

            // Se clasifican los nuevos datos
            answersInt = tree.Decide(newData);

            for (int i = 0; i < answers.Length; i++)
            {
                if (answersInt[i] < 0.5) answers[i] = false;
                else if (answersInt[i] >= 0.5) answers[i] = true;
            }
            return answers;
        }
```

- Kernel Discriminant Analysis

```
public bool[] trainK(double[][] inputs, int[] outputs, double[][] newData)
        {
            bool[] answers = new bool[newData.Rows()];

            // Se crea el objeto de KDA
            var kda = new KernelDiscriminantAnalysis()
            {
                Kernel = new Linear() //Seleccionar cualquier función de Kernel
            };

            // Se crea el clasificador
            var classifier = kda.Learn(inputs, outputs);

            // Se proyectan los datos en un nuevo espacio
            double[][] projection = kda.Transform(inputs);

            // Se lleva a cabo la clasificación
            int[] answersInt = classifier.Decide(newData);

            for (int i = 0; i < answers.Length; i++)
            {
                if (answersInt[i] < 0.5) answers[i] = false;
                else if (answersInt[i] >= 0.5) answers[i] = true;
            }
            return answers;
        }
```

Para el caso de las pruebas se llevan a cabo dos tipos de procedimientos, que resumidamente quedarían programados como se muestra a continuación.

- Validación Cruzada

Se muestra el código para el caso de probar el clasificador SVM, siendo igual para el resto de los clasificadores:

```
var crossvalidation = new CrossValidation<SupportVectorMachine<Gaussian, double[]>,
double[]>()
{
      K = _configuration.K_Folds, // Se toma la K introducida en el form de config.
```

```
        // Características del clasificador
        Learner = (s) => new SequentialMinimalOptimization<Gaussian, double[]>()
        {
                UseComplexityHeuristic = true
        },

        // Indicar cómo se mide la actuación de los modelos
        Loss = (expected, actual, p) => new ZeroOneLoss(expected).Loss(actual),

        Stratify = false, // no forzar el equilibrio de clases
};

// Grado de paralelización
crossvalidation.ParallelOptions.MaxDegreeOfParallelism = 1;

// Se computa la validación cruzada
var result = crossvalidation.Learn(inputs, outputs);

// Obtención de medidas: medias
CVtrainingErrors = result.Training.Mean;
CVvalidationErrors = result.Validation.Mean;
// Obtención de medidas: varianzas
CVtrainingErrorVar = result.Training.Variance;
CVvalidationErrorVar = result.Validation.Variance;

// Matriz de Confusión
GeneralConfusionMatrix gcm = result.ToConfusionMatrix(inputs, outputs);
GCMaccuracy = gcm.Accuracy;
GCMerror = gcm.Error;
```

- Curva ROC

Se muestra el código para el caso de probar el clasificador SVM, al igual que en el caso anterior:

```
int[] answers = new int[inputs.Length];

// Valores obtenidos del entrenamiento (resultados)
answers = trainSVM_int(inputs, outputs, inputs);

// Valores esperados (etiquetas)
bool[] expected = new bool[answers.Length];
for (int j = 0; j < answers.Length; j++)
{
        expected[j] = Convert.ToBoolean(outputs[j]);
}

// Crear la curva ROC
var roc = new ReceiverOperatingCharacteristic(expected, answers);
roc.Compute(_configuration.CutOffPoints); // Crear curva con número de puntos
determinado por la configuración del usuario

//Extracción de parámetros de la curva
double ROC_Area = roc.Area;
double ROC_Error = roc.StandardError;
```

4. Caso Práctico

En el presente capítulo se presenta un ejemplo de uso completo del software Spector. Como se comenta anteriormente en esta memoria, es necesario contar con una galería de imágenes de las piezas que se pretenden inspeccionar, teniendo en cuenta que se deben tener tanto imágenes de piezas correctas como una buena cantidad de aquellas que presenten los defectos que se pretenden identificar en la línea de producción en la que esté montado el sistema.

Para la siguiente puesta a prueba se cuenta con 96 imágenes de tornillos correctos, lo que en principio es una muestra pequeña pero que para esta prueba en concreto será suficiente. Se trata de tornillos de 4,5mm de grosor y de 40mm de longitud, como los que se muestran en la figura 4.1:

Figura 4.1 – Tornillos 4,5x40

Sobre las condiciones del experimento caben destacar la utilización de una mesa de luz similar a la de la figura 4.2. Por otro lado, con un soporte y una cámara a distancia fija de la superficie de la mesa se han tomado todas las imágenes de los tornillos.

Figura 4.2 – Mesa de Luz para las Pruebas

Antes de entrar en la prueba del software, se muestran las imágenes tomadas de los tornillos. En primer lugar, se muestran dos imágenes en la figura 4.3 en las que los tornillos son correctos:

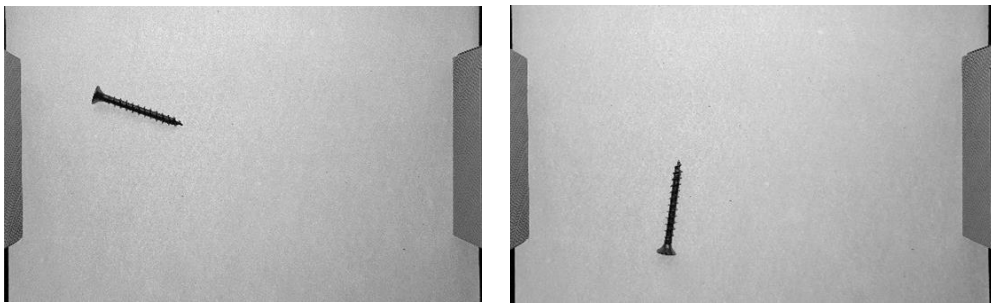


Figura 4.3 – Imágenes Correctas de Tornillos

Por otro lado, algunas imágenes con tornillos incorrectos son las que se muestran en la figura 4.4. En este caso, se contempla la posibilidad de que el tornillo sea de un tipo diferente, que esté incompleto o tenga algún defecto de forma.

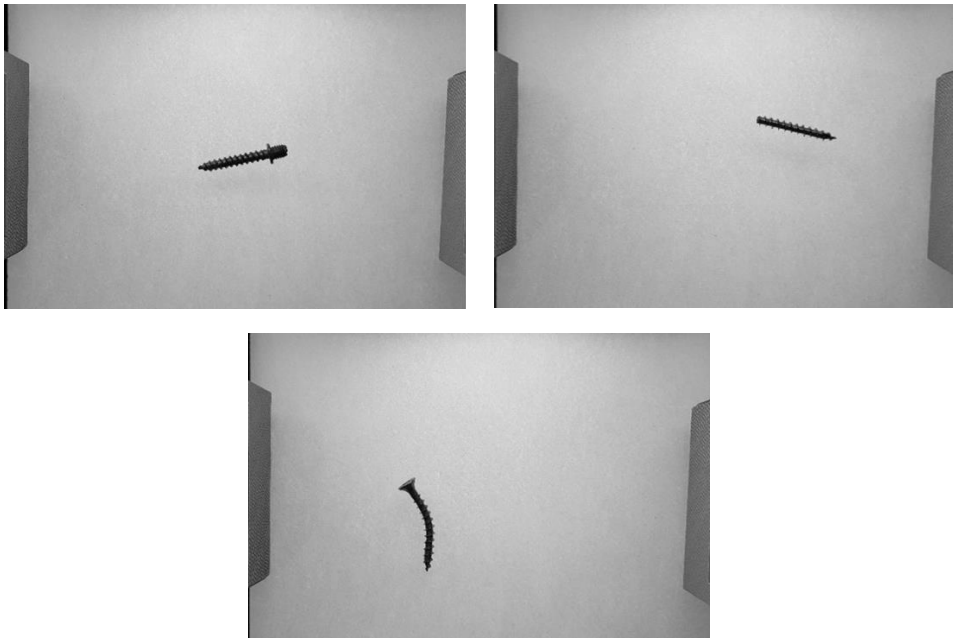


Figura 4.4 – Imágenes Incorrectas de Tornillos

Conocidas las imágenes, en los siguientes apartados se pasa a describir el proceso de preparación, entrenamiento e inspección mediante el software paso a paso, tal y como se ha descrito en el apartado anterior.

1) Procesamiento de las Imágenes

Tras realizar varias pruebas, se ha optado por llevar a cabo dos procesos bastante básicos para el procesamiento en esta fase de pruebas.

Al cargar una imagen de la base de datos, ésta aparece en color y en ByN, tal y como se muestra en la figura 4.5.

Figura 4.5 – Muestra de Imagen en el Control dedicado a ello

Sobre las dos ediciones comentadas anteriormente, se tendría por un lado un proceso de dilatación que permite eliminar el ruido en la imagen. Posteriormente se llevaría a cabo un proceso de Threshold con un valor constante de 137. Para llegar a este valor puede partirse del valor arrojado por el método de Otsu y modificarlo hasta encontrar aquel valor límite que separe de manera más adecuada el objeto en cuestión. Estos procesos quedan reflejados en la caja de la figura 4.6.

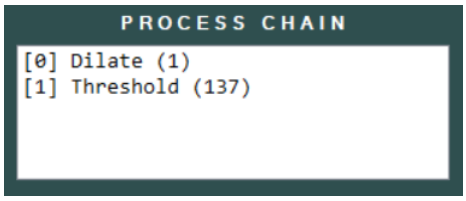


Figura 4.6 – Cadena de Procesamiento Creada

La imagen final, umbralizada, quedaría como se muestra en la figura 4.7:

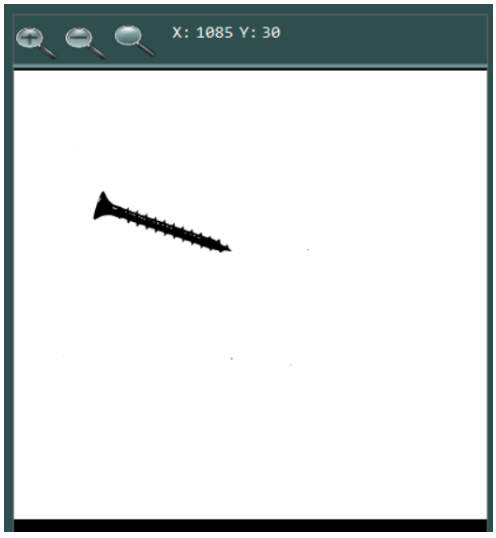


Figura 4.7 – Cadena de Procesamiento Creada

2) Datos de Entrenamiento

Siguiendo el flujograma correspondiente, la siguiente fase del trabajo sería la de crear el conjunto de datos necesario para entrenar el clasificador, pero antes de eso se debe seleccionar un modelo (bien ya creado, o bien creándolo previamente).

Así, en la figura 4.8 se aprecia cómo quedaría la parte de la interfaz dedicada a ello: tras pulsar en la estrella y seleccionar el modelo "Tornillo", se muestra cierta información sobre la pieza en sí. En la parte derecha se muestran las características seleccionadas para crear el conjunto: perímetro del contorno, área del mismo, altura y anchura de la pieza (seleccionadas a partir del rectángulo menor que encierra el contorno) y coeficiente de circularidad.

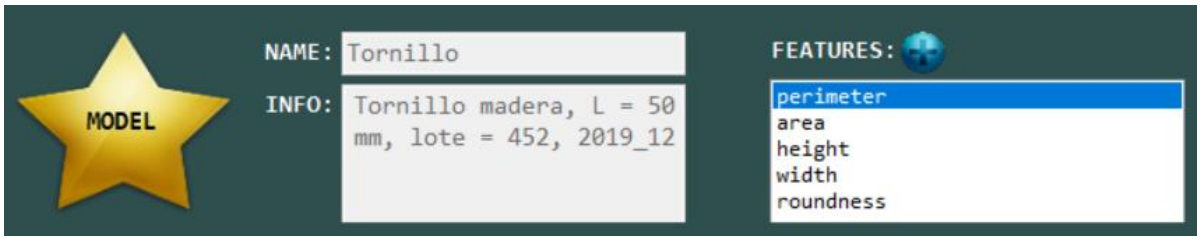


Figura 4.8 – Modelo Seleccionado y Características a Extraer

Con estas selecciones previas, el siguiente paso sería llevar a cabo la búsqueda de contornos y selección de este. Cuando se pulsa en el botón correspondiente y se selecciona el contorno adecuado, los valores para cada característica se muestran en la tabla y justo debajo se ve una visualización del contorno aislado.

Si el contorno corresponde a una pieza correcta, se debe dejar el botón de "CORRECT" tal cual, mientras que, en el caso de una pieza incorrecta, se pulsará sobre el mismo para cambiar su estado a "FAIL". El botón se sitúa tal y como aparece en la figura 4.9.

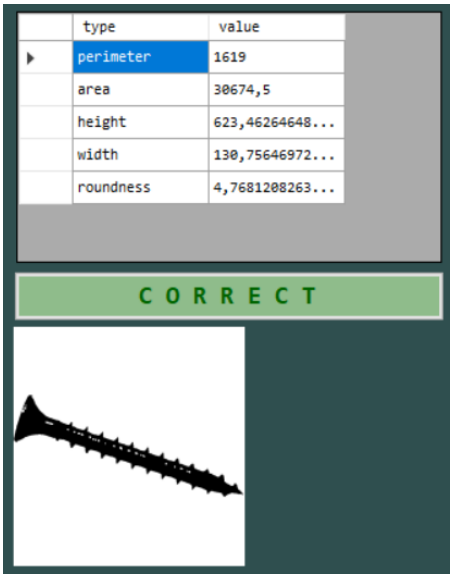
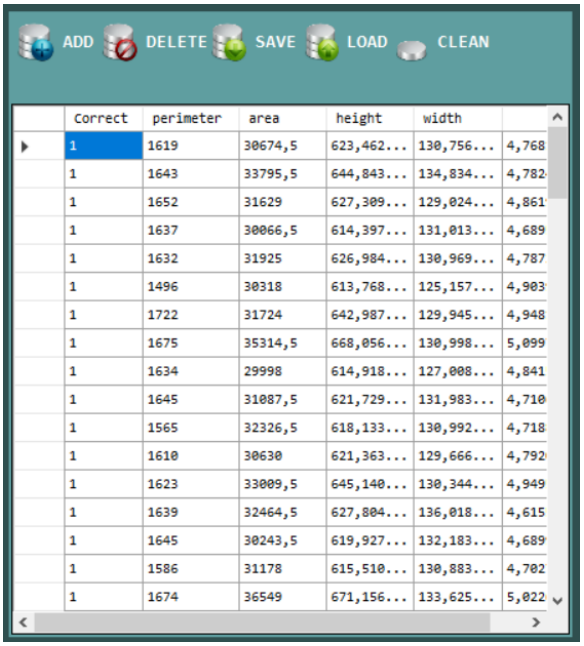


Figura 4.9 – Características de la Pieza y Contorno de Esta

Con esto, lo único que quedaría sería añadir los datos al conjunto, mediante el botón “ADD” que se encuentra en la parte superior de la tabla del conjunto. En la figura 4.10 se muestra esta tabla completa. Si se quiere eliminar el dato recién introducido, basta con pulsar en “DELETE”. Es importante guardar los datos mediante el botón “SAVE”, pudiendo cargarlos en otro momento con el botón “LOAD”.



	Correct	perimeter	area	height	width	
▶	1	1619	30674,5	623,462...	130,756...	4,768
	1	1643	33795,5	644,843...	134,834...	4,782
	1	1652	31629	627,309...	129,024...	4,861
	1	1637	30066,5	614,397...	131,013...	4,689
	1	1632	31925	626,984...	130,969...	4,787
	1	1496	30318	613,768...	125,157...	4,903
	1	1722	31724	642,987...	129,945...	4,948
	1	1675	35314,5	668,056...	130,998...	5,099
	1	1634	29998	614,918...	127,008...	4,841
	1	1645	31087,5	621,729...	131,983...	4,710
	1	1565	32326,5	618,133...	130,992...	4,718
	1	1610	30630	621,363...	129,666...	4,792
	1	1623	33009,5	645,140...	130,344...	4,949
	1	1639	32464,5	627,804...	136,018...	4,615
	1	1645	30243,5	619,927...	132,183...	4,689
	1	1586	31178	615,510...	130,883...	4,702
	1	1674	36549	671,156...	133,625...	5,022

Figura 4.10 – Características de la Pieza y Contorno de Esta

1) Pruebas e Inspección

En esta última fase del proyecto llega la parte de realizar las pruebas de clasificación. Para ello es necesario seleccionar el clasificador, el algoritmo de reducción de dimensionalidad que se quiera y el método de test deseado. Las distintas posibilidades que ofrece el software Spector hacen que se puedan realizar hasta una treintena de verificaciones distintas, lo que ha llevado a la selección de una combinación de posibilidades para mostrar en este ejemplo de uso.

El algoritmo de clasificación seleccionado es el del Árbol de Decisión, además se ha decidido dejar fuera la reducción de dimensionalidad al no ofrecer una variación en los resultados apreciable para el caso sencillo que se muestra como ejemplo.

Dicho esto, sí que se muestran los resultados obtenidos para los dos tipos de pruebas de verificación. La primera de ellas, la Validación Cruzada, arroja estos resultados, tal y como se muestran en la parte de resultados de la figura 4.11:

```
R E S U L T S
*** CROSS VALIDATION ANALISYS PERFORMED ***
Classifier: DecisionTree
Dim. Reduction Algorithm: None

Training Errors: 0,016306 (0,000)
Validation Errors: 0,072727 (0,005)

General Confusion Matrix:
Accuracy: 0,9266
Error: 0,0734
```

Figura 4.11 – Resultados de la Validación Cruzada

De estos resultados se muestra que el algoritmo presenta una precisión del 92,66%, o lo que es lo mismo, un error del 7,34%.

Si pasamos a la prueba de la curva ROC, dicha curva presenta la forma que se aprecia en la figura 4.12:

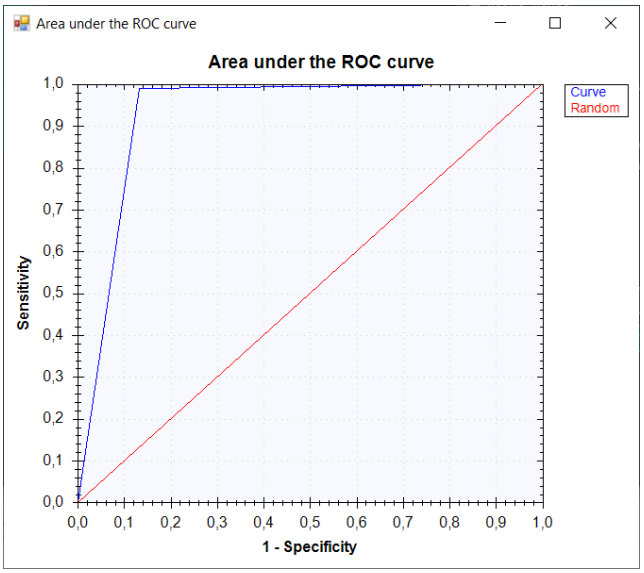


Figura 4.12 – Curva ROC

En la zona de resultados, además, puede observarse en la figura 4.13 que el método arroja los siguientes resultados: precisión del 92,8% (área encerrada bajo la curva), y un error del 4,6%.

```
R E S U L T S
*** ROC CURVE ANALISYS PERFORMED ***
Classifier: DecisionTree

Performance: 0,928
Error: 0,046
```

Figura 4.13 – Resultados de la Curva ROC

La última parte del uso del software es para la que realmente se ha diseñado el programa: inspeccionar otras piezas. Así bien, con la cadena de procesamiento creada en la primera fase y cargada, el modelo y conjunto de entrenamiento creado en la fase segunda y los algoritmos adecuados seleccionados en la presente fase, solo faltaría cargar nuevas imágenes e inspeccionar.

En la figura 4.14 se muestra la inspección de una pieza que es correcta. Como se puede comprobar, tras llevar a cabo la inspección, el software a identificado un tornillo de características geométricas correctas en la posición [1336, 448], tal y como se muestra en la zona de resultados.

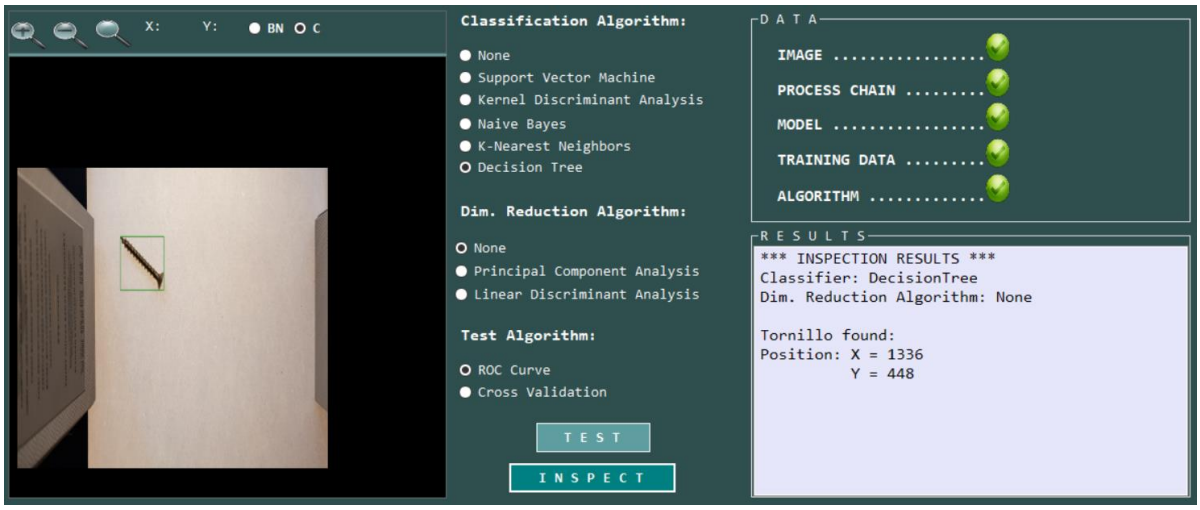


Figura 4.14 – Inspección de una Pieza Correcta

Por último, se muestra el ejemplo de inspección de una pieza que no es correcta en la figura 4.15. Como resultado de la inspección, los blobs identificados que no corresponden con las características geométricas adecuadas aparecen recuadrados en rojo. En el caso anterior dichos blobs son recuadrados en verde. Cabe destacar la posibilidad de inspeccionar diversas piezas a la vez, es decir, en la misma imagen.

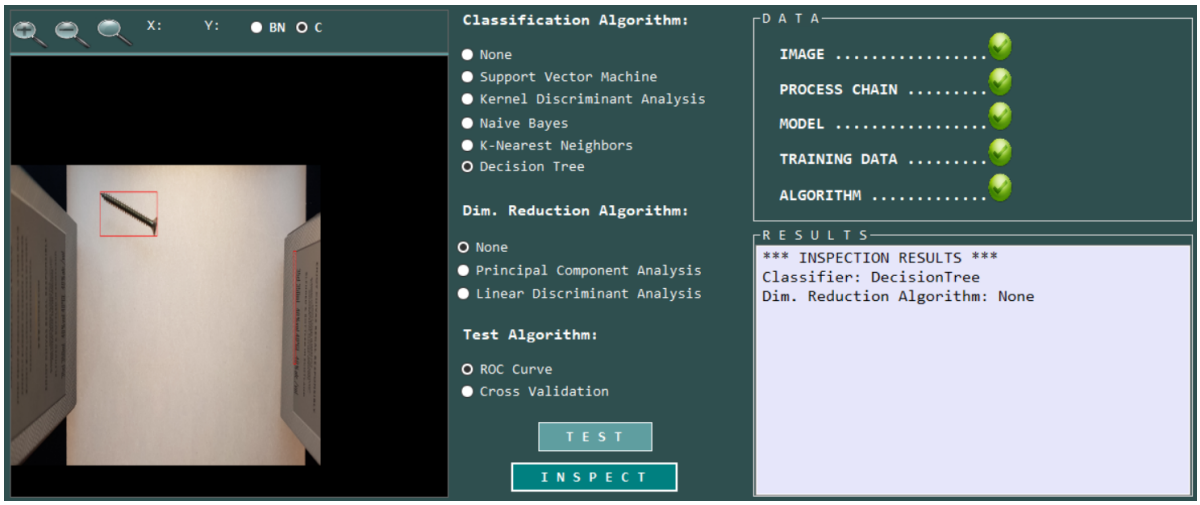


Figura 4.15 – Inspección de una Pieza Incorrecta

5. Conclusión y Trabajos Futuros

Este último capítulo está dedicado a las conclusiones que se han extraído de la realización del software Spector, así como a posibles vías de ampliación de este en caso de que sea usado como base para llegar a un programa más robusto o con nuevas funcionalidades.

5.1. Conclusiones

Las conclusiones que se extraen de la realización de este programa, así como de la utilización de las herramientas elegidas para el mismo, serían las siguientes:

- Uno de los puntos más importantes era la creación de una interfaz gráfica sencilla e intuitiva. El objetivo relacionado con esto era generar un software que pudiera ser usado por cualquier empleado dentro de un contexto industrial, reduciendo el coste de mano de obra cualificada y de formación de esta. Spector presenta una interfaz dividida en 3 pantallas principales desde las que se visualiza de forma clara la forma de proceder, dividida del mismo modo en 3 fases perfectamente definidas. Además, una combinación de colores oscuros y botones vistosos y coloridos permite intuir el funcionamiento del mismo con una dedicación de tiempo baja.

- Otro de los objetivos que se pretendían con este proyecto era establecer un primer contacto con el mundo del Machine Learning, el cual se encuentra en constante crecimiento e integración en la industria de nuestro país en los últimos años. Con este software se ha indagado en aquellos clasificadores más importantes, así como en otro tipo de algoritmos que resultan fundamentales para poder trabajar con los propios clasificadores: los algoritmos de test y los de reducción de dimensionalidad. Una de las cualidades que tiene este mundo, el de la programación y el software, es que a menudo resulta prescindible conocer al detalle la base matemática que sustenta a los algoritmos utilizados, todo esto gracias a la existencia de librerías y funciones que permiten hacer uso de estos métodos conociendo de forma general su funcionamiento.

- Lograr un software de utilidad en un contexto industrial conlleva algunas funcionalidades que resultan fundamentales en relación con el avance de la calidad y la gestión de la planta de producción. Spector permite la gestión de datos mediante la exportación e importación mediante el uso de archivos .XML, gracias al proceso conocido como serialización. Del mismo modo, el programa lleva un exhaustivo control de cada acción realizada, así como de cada resultado obtenido, generando 3 archivos de seguimiento del software, donde cada acción es registrada de manera clara incluyendo la hora y día en que se ha realizado.

- Si bien este proyecto no se ha creado con la intención de ser implementado en una industria real, otro paso necesario en su creación es la realización de un Manual de Usuario. El manual creado, además de sencillo y claro, contiene la suficiente información para que el operario a cargo de su uso pueda hacer uso pleno de todas las funcionalidades implementadas, teniendo una idea de cómo deben ser los resultados a extraer del software.

- Una de las intenciones que se pretendían desde el principio no era otra que la de crear un software versátil, que tuviera capacidad de resolver más de un problema de inspección sin que fuera necesario adaptarlo. Prueba de ello es el ejemplo de aplicación, en el que se ha utilizado para discernir diversos fallos o defectos en tornillos de métrica 4,5x40. Si se deseara, por ejemplo, comprobar varias dimensiones de cualquier otra pieza, el modo de proceder sería el mismo que se ha descrito en el capítulo 4 de esta memoria, sin tener que realizar ninguna modificación en el código del programa.

5.2. Trabajos Futuros

Para finalizar esta memoria, se concluye cuáles serían algunas vías de mejora para este software:

- La exportación a otros Sistemas Operativos, pues por el momento solo estaría listo para ser ejecutado en Windows, sería una posible mejora para realizar, dotando al programa de mayor versatilidad y posibilidades de uso.

- Uno de los aspectos que se ha decidido dejar fuera es la calibración. Si bien existen dimensiones numéricas, estas corresponden a los píxeles de la imagen. La única manera de establecer una relación entre píxeles y dimensiones reales sería mediante una calibración (extrínseca e intrínseca). Para simplificar el proyecto, se ha optado por saltar este paso. Esto no hace que el software sea menos válido, pues si se mantiene la cámara fija (a una distancia constante de la mesa) los resultados resultan igualmente adecuados.

- Otro de los aspectos a añadir al software es la conexión con una cámara industrial. Si bien se trata de un procedimiento sencillo, el carácter de este proyecto siempre fue limitar el trabajo a la parte software, dejando de lado el hardware. Es por esto por lo que se ha optado por la carga de imágenes desde archivo. Sin embargo, añadir un formulario extra para establecer la conexión con la cámara no sería complicado, si bien muy necesario cuando el software se utiliza directamente en la línea de producción. Si, por el contrario, el software se utiliza en alguna oficina más dedicada a la calidad, es posible que no sea necesario esta parte, siendo suficiente con la carga de las imágenes de las piezas desde una carpeta una vez se han tomado las fotos.

6. Bibliografía

La bibliografía consultada en el desarrollo de este proyecto es la siguiente:

[1] "Software de imagen - Productos de Visión Artificial - Infaimon." [Online]. Available: https://www.infaimon.com/categoria-producto/software-de-imagen/. [Accessed: 19-Nov-2019].

[2] "Matrox Imaging - About." [Online]. Available: https://www.matrox.com/imaging/en/about/. [Accessed: 19-Nov-2019].

[3] S. Sastoque, C. Narváez, and G. Garnica, "Metodología para la construcción de Interfaces Gráficas Centradas en el Usuario."

[4] R. D. Pea, "User Centered System Design: New Perspectives on Human-Computer Interaction," 1987.

[5] T. Brown and C. Funk, "Design Thinking," 2012.

[6] "ISO - ISO 9241-210:2010 - Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems." [Online]. Available: https://www.iso.org/standard/52075.html. [Accessed: 05-Nov-2019].

[7] "ISO - ISO 9241-11:1998 - Ergonomic requirements for office work with visual display terminals (VDTs) — Part 11: Guidance on usability." [Online]. Available: https://www.iso.org/standard/16883.html. [Accessed: 05-Nov-2019].

[8] T. S. Huang, "Computer Vision: Evolution and Promise," 1996.

[9] D. Marr, "Understanding Complex Information-processing Systems," *W. H. Free. Company, Used with Permis.*, no. 1956, pp. 69-?, 1982.

[10] A. Ros García, "Sistema de detección de objetos mediante cámaras de tiempo de vuelo (ToF). Aplicación en conducción autónoma," 2017.

[11] J. A. Cortés Osorio, F. A. Medina Aguirre, and M. S. J. Mendoza Vargas, "Sistema De Visión Por Computador Para El Control De Calidad En La Producción," *Sci. Tech.*, no. 45, pp. 130–134, 2010.

[12] J. F. Vélez Serrano, A. B. Moreno Díaz, Á. Sánchez Calle, and J. L. Esteban Sánchez-Marín, "Visión por Computador."

[13] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, "Real-Time Computer Vision with OpenCV," 2012.

[14] G. Bradsky and A. Kaehler, *Learning OpenCV*, vol. 16, no. 3. 2008.

[15] P. Huamaní Navarrete, "Umbralización múltiple utilizando el método de Otsu para reconocer la luz roja en semáforos," *Scientia*, vol. 17, no. 17, pp. 247–262, 2016.

[16] G. Aguilar, "Procesamiento Digital De Imágenes Utilizando Filtros Morfológicos," p. 347, 1995.

[17] "Transformaciones Morfológicas con Visión Artificial - IEEE UCSA." [Online]. Available: http://ucsa.ieeeparaguay.org/2019/05/15/transformaciones-morfologicas-con-

vision-artificial/. [Accessed: 23-Oct-2019].

[18] G. Xie and W. Lu, "Image Edge Detection Based On Opencv," *Int. J. Electron. Electr. Eng.*, vol. 1, no. 2, pp. 104–106, 2013.

[19] J. Seo, S. Chae, J. Shim, D. Kim, C. Cheong, and T. D. Han, "Fast contour-tracing algorithm based on a pixel-following method for image sensors," *Sensors (Switzerland)*, vol. 16, no. 3, Mar. 2016.

[20] Y. He, B. Gao, A. Sophian, and R. Yang, "Coil-Based Rectangular PEC Sensors for Defect Classification," in *Transient Electromagnetic-Thermal Nondestructive Testing*, Elsevier, 2017, pp. 55–90.

[21] "Aprendizaje automático: Qué es y por qué es importante | SAS." [Online]. Available: https://www.sas.com/es_es/insights/analytics/machine-learning.html. [Accessed: 31-Oct-2019].

[22] R. Prieto, A. Herrera, J. L. Pérez, and A. Padrón, "El Modelo Neuronal de McCulloch y Pitts."

[23] M. Kaytan and I. B. Aydilek, "A review on machine learning tools," no. March, pp. 1–4, 2017.

[24] T. M. Cover and P. E. Hart, "This Week's Citation Classic: Cover T.M. & Hart P.E. Nearest neighbor pattern classification," *Curr. Contents*, vol. 13, no. 1, p. 20, 1982.

[25] E. Fernández, "Análisis de Clasificadores Bayesianos."

[26] G. Shobha and S. Rangaswamy, "Computational Analysis and Understanding of Natural Languages: Principles, Methods and Applications," *Handbook of Statistics*, 2018. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/supervised-learning/pdf. [Accessed: 07-Nov-2019].

[27] M. Cardenas, R. Medel, J. Castillo, J. C. Vázquez, and O. Casco, "Modelos de Aprendizaje Supervisados: aplicaciones para la predicción de incendios forestales en la provincia de Córdoba."

[28] M. R. M. Talabis, R. McPherson, I. Miyamoto, J. L. Martin, and D. Kaye, "Analytics Defined," in *Information Security Analytics*, Elsevier, 2015, pp. 1–12.

[29] J. Park, T. Kim, and S. Seong, "Providing support to operators for monitoring safety functions using reinforcement learning," *Prog. Nucl. Energy*, 2020.

[30] Y. Li, C. Lin, and W. Zhang, "Improved sparse least-squares support vector machine classifiers," *Neurocomputing*, vol. 69, no. 13–15, pp. 1655–1658, 2006.

[31] J. M. Keller and M. R. Gray, "A Fuzzy K-Nearest Neighbor Algorithm," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-15, no. 4, pp. 580–585, 1985.

[32] C. Malagón Luque, "Clasicadores bayesianos. El algoritmo Naïve Bayes."

[33] J. Orellana Alvear, "Arboles de decision y Random Forest," 2018. [Online]. Available: https://bookdown.org/content/2031/arboles-de-decision-parte-i.html#que-son-los-arboles-de-decision. [Accessed: 12-Nov-2019].

[34] L. Li, S. Liu, Y. Peng, and Z. Sun, “Overview of principal component analysis algorithm,” *Optik (Stuttg).*, vol. 127, no. 9, pp. 3935–3944, May 2016.

[35] D. Garcia-Alvarez and M. J. Fuente, “Estudio comparativo de técnicas de detección de fallos basadas en el análisis de componentes principales (PCA),” *RIAI - Rev. Iberoam. Autom. e Inform. Ind.*, vol. 8, no. 3, pp. 182–195, 2011.

[36] S. Wang, J. Lu, X. Gu, H. Du, and J. Yang, “Semi-supervised linear discriminant analysis for dimension reduction and classification,” *Pattern Recognit.*, 2016.

[37] L. Pérez-Planells, J. Delegido, J. P. Rivera-Caicedo, and J. Verrelst, “Análisis de métodos de validación cruzada para la obtención robusta de parámetros biofísicos,” vol. 44, pp. 55–65, 2015.

[38] A. P. Bradley, “The use of the area under the ROC curve in the evaluation of machine learning algorithms,” *Pattern Recognit.*, vol. 30, no. 7, pp. 1145–1159, 1997.

ANEXO 1. Manual de Uso del Software “Spector”

Índice

1. INTRODUCCIÓN

Con este manual se pretende describir de la forma más sencilla y concisa posible el funcionamiento del software de inspección de piezas en entornos industriales *Spector*. A lo largo de este documento se exponen los pasos que hay que dar para la preparación y puesta en ejecución del programa, los cuales se resumen en 3 fases, fundamentalmente:

- Fase de procesamiento de las imágenes.
- Fase de extracción y creación de los datos.
- Fase de pruebas e inspección.

Además, se describen diversas características del software que incluyen la creación, guardado o carga de modelos, conjuntos de datos o historiales de acciones e inspecciones llevadas a cabo.

De manera complementaria al uso de *Spector* resulta de vital importancia la adquisición de una cámara de uso industrial con características suficientes como para generar imágenes de calidad. Si bien el software no incluye la funcionalidad extra de conexión con la cámara montada en el lugar de inspección, puede plantearse su implementación en la fase de puesta en marcha del programa.

Esquema 1 – Flujo de Información

Algunas consideraciones que se deben tener en cuenta durante la preparación de la zona de inspección son las que se recogen a continuación:

- El entorno de captura de imágenes debe ser adecuado en cuanto a iluminación, de modo que el contraste entre las piezas a inspeccionar y el fondo sea suficiente. Esto reducirá la complejidad del procesamiento y edición de las imágenes en cuestión.

- Si bien el manejo del software es sencillo e intuitivo, se requieren ciertas capacidades que permitan discernir qué características de las piezas son las más adecuadas para que los clasificadores obtengan los resultados más fiables. Esto implica un conocimiento previo del proceso, de la línea de producción y de los objetos con los que se va a trabajar.

2. DESCRIPCIÓN GENERAL DEL SOFTWARE

En este capítulo se describen todas las funcionalidades implementadas en el programa. Desde la pantalla de inicio se pueden identificar 4 zonas principales:

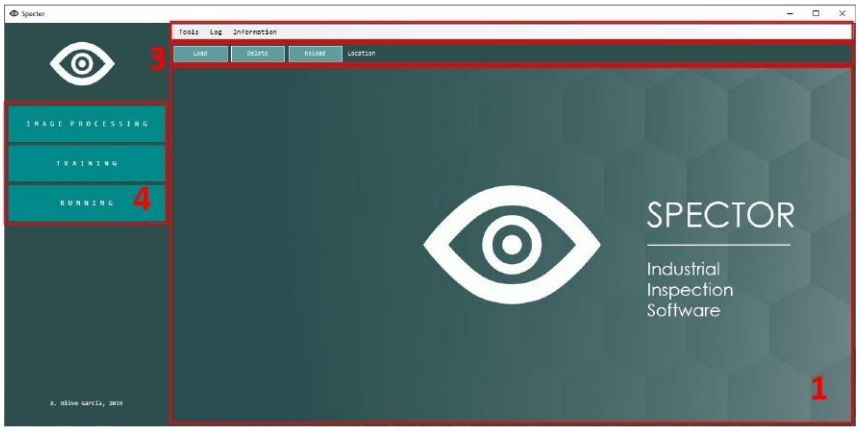


Imagen 1 – Pantalla de Inicio

Dichas zonas son:

1- **Zona de trabajo:** a medida que se avance en las distintas fases de trabajo con el software, las acciones principales se realizarán aquí.
2- **Menú de herramientas y opciones:** desde aquí se permite configurar diversos parámetros que serán explicados más adelante en este manual.
3- **Gestión de imágenes:** permite la carga, recarga y limpieza de imágenes. Dichas imágenes se muestran en la zona de trabajo.
4- **Menú de fases:** para la selección de la fase de trabajo, lo que permite visualizar en la zona de trabajo los formularios correspondientes.

La selección de cada fase de trabajo depende de en qué punto se encuentre de la inspección. El flujo de trabajo completo sería el que se muestra en el flujograma 1:

Flujograma 1 – Proceso Completo de Trabajo

2.1. Procesamiento de Imágenes

En este capítulo se detallan las diferentes acciones que pueden realizarse desde el formulario dedicado al procesamiento y edición de las imágenes.

El objetivo que se pretende en esta fase de trabajo es el de llegar a una **cadena de procesos** que pueda llevarse a cabo con todas las imágenes que desean procesarse, ya sea con intención de entrenar al software, como en la fase de funcionamiento normal, para inspeccionar nuevas piezas.

Al igual que en la pantalla inicial, la zona de trabajo puede dividirse en 3 partes claramente diferenciadas e implementadas para las 3 funciones principales que hay que tener en cuenta:

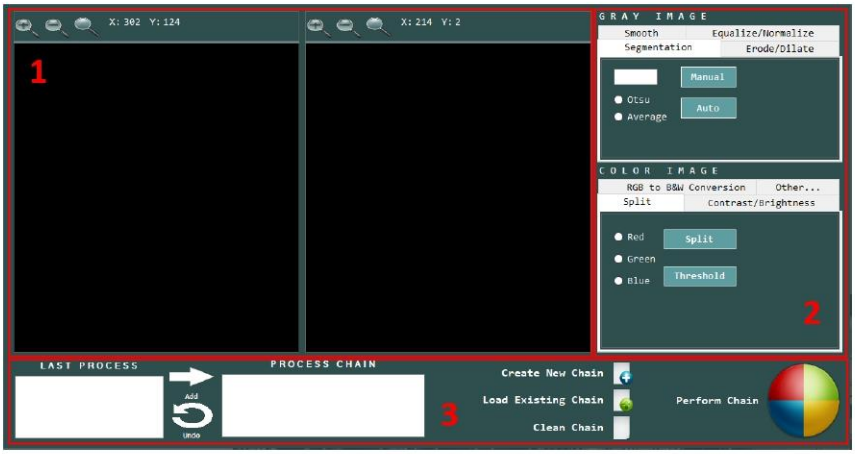


Imagen 2 – Formulario 1: Procesamiento de Imágenes

En cada área se recogen las acciones dedicadas a:

1- **Visionado de Imágenes:** si bien las imágenes se cargan gracias a los botones de la parte superior, presentes en todo momento, en este formulario se permite la visualización de dichas imágenes en blanco y negro y a todo color, lo que permite ir comprobando cada modificación llevada a cabo sobre ellas.

2- **Opciones de Edición:** se provee de diversas herramientas para llevar a cabo la modificación de las imágenes con el fin de aislar y obtener una imagen binarizada de la que se pueda extraer de manera limpia y clara el blob de la pieza que se desea analizar. Las posibilidades de edición que permite el software son:
   a. **Imagen en ByN:**
      i. Binarización (valor manual, algoritmo OTSU o de las medias)
      ii. Erosión o Dilatación
      iii. Ecualización o Normalización
      iv. Suavizado
   b. **Imagen a color:**
      i. Split (aislado de capa R, G o B)

ii. Contraste y Brillo
iii. Ecualización de Histograma
iv. Conversión a ByN

3- **Gestión cadenas de procesos**: cada vez que se lleve a cabo una edición sobre la imagen, el cuadro de texto "Last Process" mostrará dicha acción. Por medio de los dos botones "Add" y "Undo" se puede añadir acciones a la cadena de proceso, de manera que el usuario puede diseñar una cadena de acciones y guardarla. De esta manera, cuando se trabaje con imágenes similares no será necesario llevar a cabo la edición de nuevo, sino que se podrá cargar la cadena ya diseñada y guardada. Mediante el botón inferior derecho se puede transformar la imagen según la cadena creada/cargada.

Imagen 3 – Ejemplo de Cadena de Procesos

El flujo de trabajo en esta fase sería el siguiente:

Flujograma 2 – Proceso de Procesamiento de Imágenes

2.2. Extracción y Creación de Datos de Entrenamiento

En este capítulo se detallan las diferentes acciones que pueden realizarse desde el formulario dedicado a la extracción de características y creación de la matriz de datos de entrenamiento.

En esta fase de trabajo se busca crear el conjunto de datos necesario para entrenar el clasificador. Para ello, será necesario definir el modelo, las variables a extraer e ir analizando cada imagen para obtener los datos.

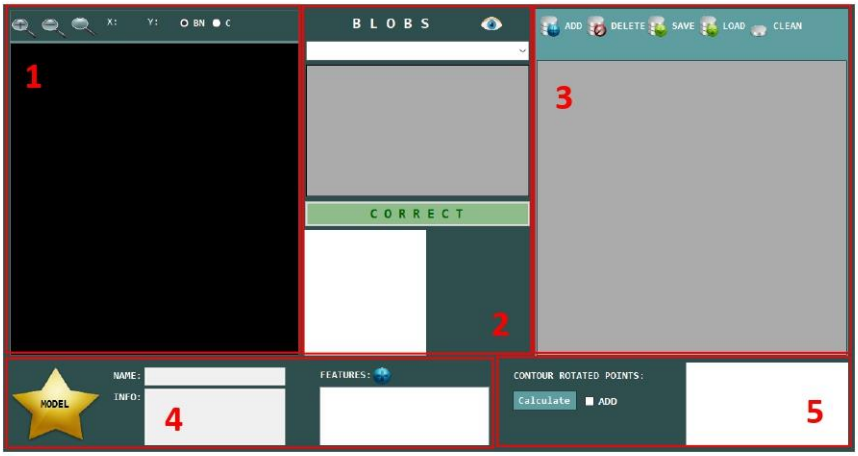


Imagen 4 – Formulario 2: extracción de datos

Las 5 zonas que hay señaladas en la imagen superior son:

1. **Visionado de Imágenes:** para reducir el área de esta zona se ha añadido un *Radio Button* en la parte superior del control que permite cambiar entre la imagen en ByN y la imagen a color. La imagen mostrada es la que ya ha sido editada en el paso anterior.

2. **Obtención de Blobs:** en este paso se deberá pulsar en el botón superior () para analizar la imagen en ByN en busca de contornos. Dichos contornos se añadirán al menú, y al seleccionar uno de ellos, los datos referentes a dicho blob se muestran en la tabla que se sitúa justo debajo. El botón "Correct/Fail" permite señalar si el contorno es correcto o falso. Además, en el cuadro inferior se muestra el contorno seleccionado recortado.

3. **Conjunto de Datos:** en la tabla de la zona de la derecha se muestran los datos que van a ser usados para entrenar los clasificadores. Los botones, de izquierda a derecha, permiten realizar las siguientes acciones:
    a. **Añadir Dato:** Los datos del cuadro de la zona 2 son añadidos al conjunto.
    b. **Eliminar Dato:** Se elimina el último dato introducido (fila).
    c. **Guardar Conjunto:** muestra un formulario de guardado, lo que permite guardar con el nombre que se desee el conjunto de datos. Paralelamente se guarda un archivo con los resultados (Correcto/Incorrecto).
    d. **Cargar Conjunto:** muestra un formulario de carga, lo que permite recuperar el conjunto de datos previamente guardado.
    e. **Limpiar Conjunto:** deja en blanco la tabla, eliminando el conjunto actual.

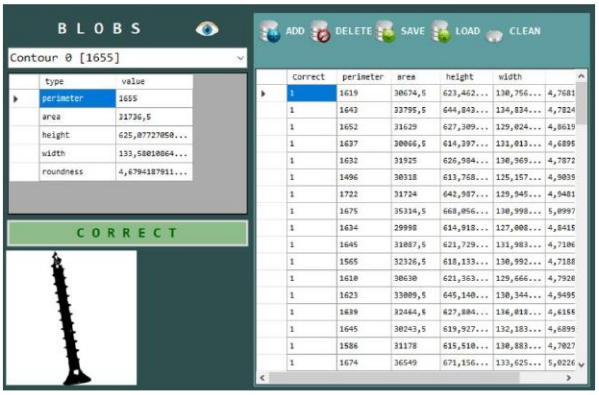


Imagen 5 – Ejemplo de Extracción de Características y Conjunto de Datos

4. **Gestión de Modelos:** En esta zona se selecciona el modelo con el que se está trabajando. Si bien existe un formulario para crear nuevos modelos, este será explicado más adelante. En esta zona se pulsa el botón ***Models*** () para seleccionar uno. Las características para extraer deberán seleccionarse antes de empezar a crear el conjunto de datos. Para ello se pulsa el botón  , lo que lleva al usuario a otro formulario, mostrado a continuación:

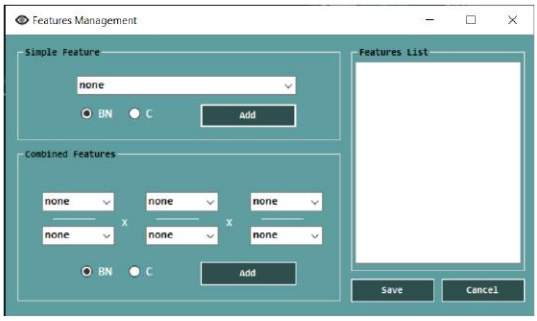


Imagen 6 – Gestión de Características

En este formulario se permite al usuario seleccionar las características (simples o compuestas) a extraer de cada contorno. Una **característica simple** puede ser una longitud, área, perímetro, etc., mientras que una **característica compleja** es la combinación de entre 2 y 6 características sencillas. Un ejemplo podría ser el área de la ROI (resultado del producto de la altura y la anchura de la ROI), o un coeficiente de rectangularidad (resultado del cociente del área del contorno y el área del mínimo rectángulo que lo encierra).

5. **Perfil del Contorno:** Como funcionalidad extra, se puede añadir el perfil de la pieza al conjunto de datos. Para ello habrá que calcularlo (botón *Calculate*) y activar la opción *ADD* antes de añadir los datos de la pieza actual al conjunto. En el cuadro de la derecha se muestra el contorno debidamente orientado y los puntos extraídos de su perfil.

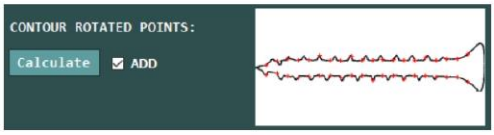
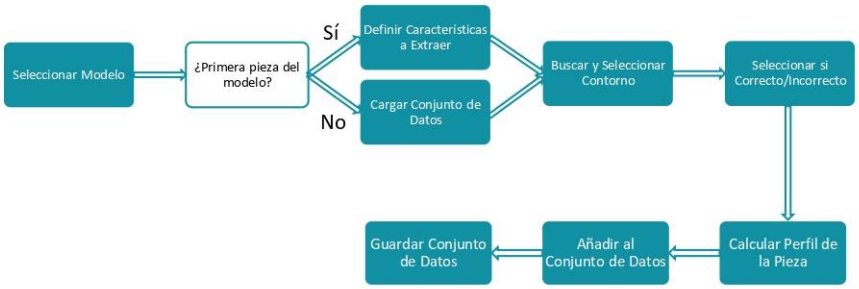


Imagen 6 – Perfil de la Pieza

Por último, se muestra a continuación el flujo de trabajo que define cómo proceder en esta fase de la inspección:



Flujograma 3 – Proceso del Conjunto de Datos

2.3. Pruebas e Inspección

En este capítulo se detallan las diferentes acciones que pueden realizarse desde el formulario dedicado a la selección, prueba y uso de los diferentes clasificadores.

En esta fase de trabajo se busca trabajar en la inspección de las piezas nuevas, seleccionando el clasificador que mejor funcione con el conjunto de datos del que se dispone.

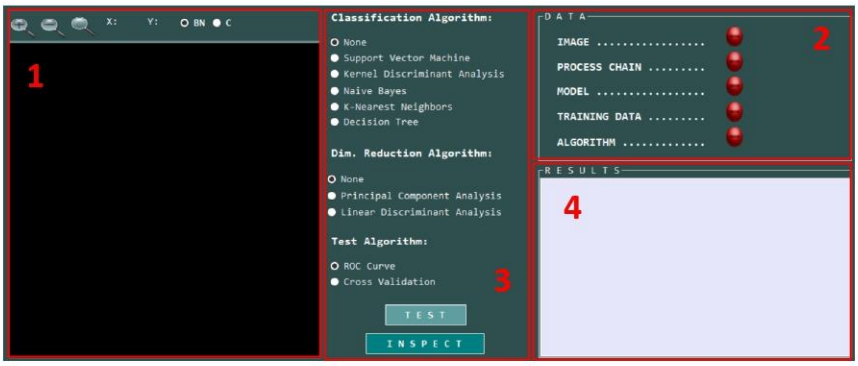


Imagen 7 – Formulario 3: Pruebas e Inspección

Las 4 zonas que hay señaladas en la imagen superior son:

1. **Visionado de Imágenes:** similar a la del formulario anterior.

2. **Comprobaciones Previas:** en esta parte se comprueba que todos los elementos necesarios para llevar a cabo el análisis han sido cargados correctamente. Dichos elementos son:
   a. **Imagen:** se carga desde la parte superior de la pantalla.
   b. **Cadena de Procesos:** se carga en la primera fase del proceso.
   c. **Modelo:** se selecciona de entre los modelos existentes en la fase de Training.
   d. **Conjunto de Datos:** se completa o carga en el cuadro de datos de la segunda fase (Training).
   e. **Algoritmo:** se selecciona el clasificador en la zona 3 del formulario actual.

   Una vez se ha cargado todo, el estado del formulario será que se muestra en la siguiente imagen:

Imagen 8 – Comprobaciones Previas a la Inspección

3. **Selección de tipo de Test y Clasificador:** esta es la zona más importante del formulario actual, pues permite al usuario elegir el clasificador que va a ser objeto de:
   a. **Las pruebas previas**: para lo cual será necesario seleccionar el tipo de prueba a realizar (**Curva ROC** o **Validación Cruzada**), y posteriormente clicar en el botón ***Test***.
   b. **La inspección:** para lo cual habrá que clicar en el botón ***Inspect***.

   Además, se puede llevar a cabo un proceso de **reducción de dimensiones**. Para ello el usuario puede seleccionar el tipo de algoritmo que desee emplear, o bien seleccionar *None* para no llevar a cabo dicha reducción.

4. **Resultados:** en el cuadro de texto de la derecha el usuario podrá visualizar los resultados tanto de las pruebas como de la propia inspección.

Por último, se muestra el flujo de trabajo para la última fase de inspección:

Flujograma 4 – Proceso de Test e Inspección

3. OTRAS FUNCIONALIDADES

En este capítulo se habla de las características extra del programa que complementan o amplían las capacidades del software más allá de la propia inspección.

3.1. Creación de Modelos

En este capítulo se detallan las diferentes acciones que pueden realizarse desde el formulario dedicado a la creación de modelos.

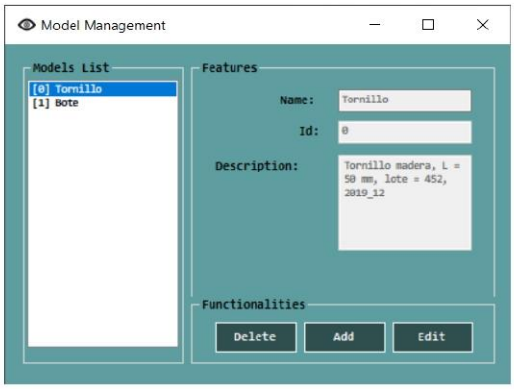


Imagen 9 – Formulario de Creación de Modelos

Los modelos deben ser creados por el usuario para representar la realidad mediante el formulario, en el que se rellenarán los siguientes apartados: nombre, número de identificación del modelo y descripción, donde se permite registrar algo de información acerca de la pieza que se pretende inspeccionar.

3.2. Parámetros de Configuración

En el software Spector se contempla la posibilidad de configurar ciertos parámetros que afecten al funcionamiento de los diferentes algoritmos implementados. El formulario es el que se muestra a continuación:

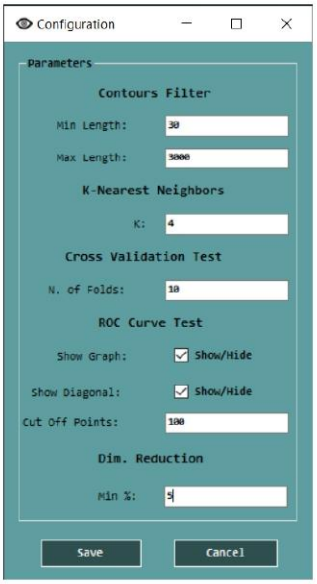


Imagen 10 – Formulario de Configuración de Parámetros

Los parámetros configurables son:

a) **Filtro de contorno:** permite establecer un valor mínimo y máximo para los contornos encontrados en la imagen, lo que dota de mayor limpieza al proceso de búsqueda de blobs.
b) **Parámetro K del clasificador KNN:** establece cuántos valores vecinos se tienen en cuenta en el proceso de clasificación de los K Valores Vecinos, tal y como se explica en el capítulo de fundamento teórico.
c) **Parámetro de Validación Cruzada:** establece el número de "folds" para dicha prueba.
d) **Parámetros de la Curva ROC:** establece el número de puntos que forman la curva, así como si se desea o no mostrar la curva, además de mostrar el resultado numérico (área bajo la misma).
e) **Porcentaje de influencia:** establece un valor umbral por debajo del cual un parámetro es eliminado del conjunto de características a tener en cuenta antes del entrenamiento del clasificador que se está utilizando.

3.3. Log

Mediante el apartado de Log se puede llevar a cabo un seguimiento de cada acción y resultado derivado del uso del programa, pudiendo generarse hasta tres archivos diferentes, uno para cada fase de funcionamiento del proyecto.