UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADO EN INGENIERÍA DE LA SALUD

VALIDACIÓN DE MODELOS DE CLASIFICACIÓN DE COBERTURA TERRESTRE UTILIZANDO EL SISTEMA DE INFORMACIÓN GEOGRÁFICA DE PARCELAS AGRÍCOLAS Y CATASTRO

VALIDATION OF LAND COVER CLASSIFICATION MODELS USING GEOGRAPHIC INFORMATION SYSTEMS FOR AGRICULTURAL PARCELS AND CADASTRE

Realizado por
Jesús Aldana Martín

Tutorizado por
Cristóbal Barba González
Antonio Manuel Burgueño Romero

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, MARZO, 2024

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA DE LA SALUD

VALIDACIÓN DE MODELOS DE CLASIFICACIÓN DE COBERTURA TERRESTRE UTILIZANDO EL SISTEMA DE INFORMACIÓN GEOGRÁFICA DE PARCELAS AGRÍCOLAS Y CASTASTRO

VALIDATION OF LAND COVER CLASSIFICATION MODELS USING GEOGRAPHIC INFORMATION SYSTEMS FOR AGRICULTURAL PARCELS AND CADASTRE

Realizado por:

Jesús Aldana Martín

Tutorizado por:

Cristóbal Barba González, Antonio Manuel Burgueño Romero

Departamento:

Lenguajes y Ciencias de la Computación

MÁLAGA, MARZO, 2024

Fecha defensa:

Fdo.: El Secretario del Tribunal

Resumen

Este trabajo de fin de grado tiene como propósito la creación de una herramienta de software que permita validar modelos de clasificación de cobertura terrestre en la península ibérica. Con el fin de supervisar y monitorizar la distribución de campos de cultivo, se emplearán los datos del Sistema de Información Geográfica de Parcelas Agrícolas (SIGPAC), suministrados por el Ministerio de Agricultura, Pesca y Alimentación y de acceso público. Con la ayuda de estos datos e imágenes satélites de Sentinel-2 proporcionadas por la agencia espacial europea tomadas mediante técnicas de Teledetección. Con las imágenes satélite procesadas se ejecutará un script de Python que obtenga un raster de salida con los nuevos datos de entrada y métricas que nos permitan medir y validar el rendimiento del modelo de clasificación. Para poner a prueba la herramienta se analizará la provincia de Málaga y la comunidad autónoma de Andalucía. Obteniendo distintos raster de análisis y métricas estadísticas para la validación.

Palabras clave: Teledetección, Validación, Python, Clasificación, SIGPAC.

Abstract

This bachelor's thesis aims to develop a software tool for validating land cover classification models in the Iberian Peninsula. In order to supervise and monitor the distribution of croplands, data from the Geographic Information System for Agricultural Parcels (SIGPAC), provided by the Ministry of Agriculture, Fisheries and Food and publicly accessible, will be utilized. With the assistance of remote sensing data and processed satellite images provided by the european space agency, a Python script will be executed to generate an output raster containing new input data and metrics enabling us to measure and validate the classification model's performance. To test the tool, an analysis will be conducted on the province of Málaga and the autonomous community of Andalusia, obtaining various analysis rasters and statistical metrics for validation.

Keywords: Remote Sensing, Validation, Python, Classification, SIGPAC.

Índice general

1 Introducción

1.1. Contexto y motivación del estudio

En los últimos años, gracias a la aparición de nuevas herramientas de captura de imágenes multiespectrales y el avance del desarrollo software han surgido nuevas áreas de conocimiento e investigación. La necesidad de abordar dichos desafíos de manera efectiva nos ha llevado a la búsqueda de soluciones innovadoras y a la aplicación de nuevas tecnologías. Entre estas nuevas tecnologías aparece la agricultura de precisión, que se refiere a la aplicación de técnicas y herramientas modernas que mejoran la productividad agrícola al mismo tiempo que minimizan el impacto ambiental. Estas tecnologías permiten a los agricultores tomar decisiones más precisas y estratégicas, al proporcionar información detallada sobre la salud de los cultivos y la identificación precisa del tipo de cobertura terrestre. En este trabajo de fin de grado se pretende validar un modelo de clasificación de cobertura terrestre haciendo uso de tecnologías avanzadas como imágenes multiespectrales, datos raster, datos vectoriales y tecnologías de desarrollo de software.

Este proyecto de fin de grado se ha llevado a cabo con investigadores del grupo de investigación Khaos [1] de la Universidad de Málaga y sigue una línea de investigación del grupo en relación a la cobertura terrestre y la agricultura de precisión. Concretamente busca conseguir validar de cierta forma el modelo y flujo de trabajo que se ha desarrollado en el grupo, que explican los propios autores en el siguiente artículo [2]. Este proyecto por tanto continúa esta línea de investigación del grupo y, pretende de cierta forma replicar con mayor precisión el proyecto europeo **CORINE** [3]. Este proyecto es considerado en la literatura por muchos investigadores como el líder a nivel europeo respecto al mapeo de la ocupación del suelo de la superficie terrestre. El proyecto CORINE (Coordination of Information of the Environment)

inició en el año 1985 a raíz de una decisión del Consejo de Ministros de la Unión Europea que, posteriormente en 1995 pasó a ser responsabilidad de la Agencia Europea de Medio Ambiente (AEMA). Este proyecto ha conseguido clasificar la cobertura del suelo de toda la superficie terrestre en nueve clases inspirando a muchos investigadores a aplicar este tipo de herramientas en sus proyectos acuñando el término de **'land cover'** (Cobertura terrestre). La figura 1.1 muestra el proyecto CORINE y como la superficie terrestre se ha clasificado en las nueve clases observables abajo a la derecha que, en orden descendente son; Bosques, pastizal, vegetación herbácea, humedales, musgo y liquen, tierra descubierta, cultivos, edificaciones, hielo y nieve y por ultimo reservas permanentes de agua.

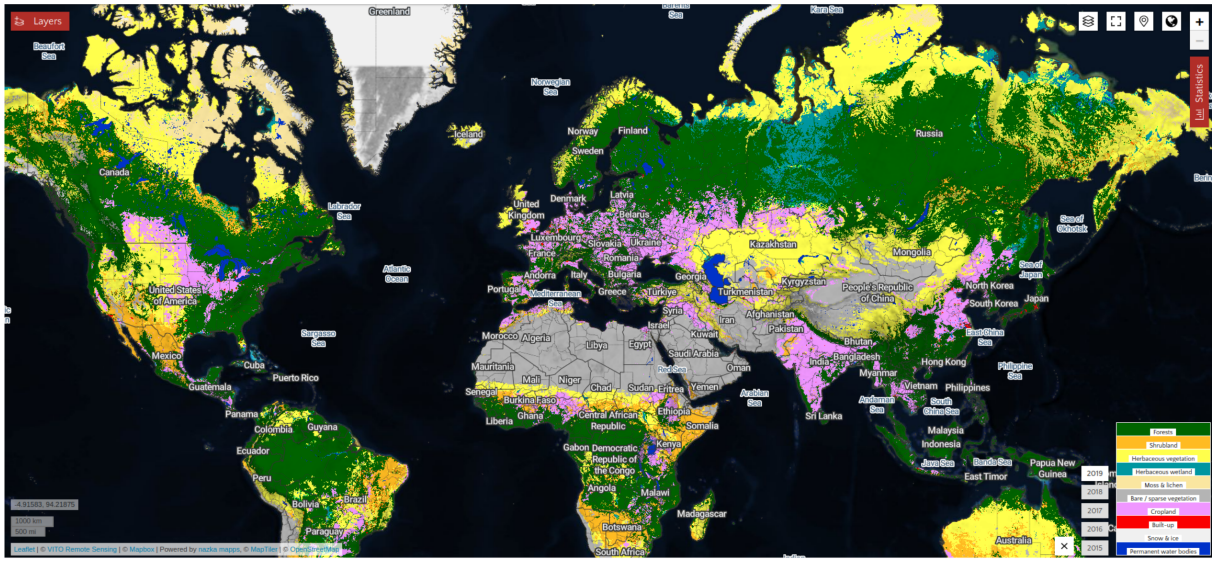


Figura 1.1: Servicio web del proyecto CORINE. Fuente: [3]

Actualmente el 'land cover' o cobertura de suelo es una práctica muy común que se basa en la clasificación y la categorización de los diferentes terrenos que cubren la superficie de la Tierra como los bosques, cultivos, pastizales, cuerpos de agua, áreas urbanas, etc. Estas aproximaciones utilizan entre otros enfoques datos de imágenes multiespectrales y datos geoespaciales para clasificar el terreno con el objetivo de proporcionar ayudas e información en disciplinas como la ecología, planificación urbana, gestión y monitorización ambiental, agricultura de precisión, evaluación de riesgos naturales y gestión de recursos [4]. Uno de los métodos más usados en la clasificación de 'land cover' es la clasificación orientada a objetos. A diferencia de los métodos de clasificación tradicionales que tratan cada píxel de forma individual, la clasificación orientada a objetos utiliza la información espacial y contextual para

agrupar los píxeles en objetos coherentes. Puesto que este tipo de objetos de una imagen, como edificios, cultivos o agua, tienen características y formas distintivas que pueden ser identificadas y clasificadas. Al analizar los objetos en lugar de los píxeles individuales, la clasificación orientada a objetos proporciona resultados más precisos y semánticamente más significativos, lo que la convierte en una herramienta valiosa para diferentes aplicaciones [5].

Pese a todas las ventajas ya mencionadas. El gran problema que tiene está práctica es la validación de los modelos una vez clasificados. La semántica del proyecto dificulta mucho la validación de los resultados de forma automática y por ello se recurre en la literatura a la validación de este tipo de raster de forma manual, conociendo la zona de clasificación con planos catastrales y geográficos, no obstante en este proyecto se pretende proporcionar una estrategia para la validación de este tipo de clasificadores mediante los datos del Sistema de Información Geográfica de Parcelas Agrícolas y Catastro (SIGPAC). Este tipo de algoritmo que permita el análisis de los resultados puede dar problemas debido a la gran cantidad de información que se clasificada. Por ejemplo en la figura 1.1 cada píxel de esa imagen tiene una resolución espacial de 100 metros y ocupan toda la superficie terrestre por lo que hablamos de miles de millones de píxeles de estudio.

1.2. Objetivo del estudio

Este proyecto busca ofrecer un algoritmo estandarizado que valide automáticamente un raster dado con información geográfica del territorio español. Como objetivos secundarios se busca crear algoritmos y métodos que faciliten el trabajo con grandes cantidades de datos raster y vectoriales, así como una implementación optimizada que ofrezca resultados en el menor tiempo posible. Lo anterior comentado se automatizará en un script de bash que, recibiendo las imágenes satélite clasificadas y los datos del gobierno del sistema de información geográficas de parcelas agrícolas y catastro como parámetros de entrada, genere imágenes rasterizadas en formato .tiff de salida con los datos SIGPAC así como métricas estadísticas y gráficas con las que medir el desempeño del clasificador para posteriormente validar un modelo de clasificación con esos resultados y medir la calidad del modelo así como para discernir el tipo de cultivo o terreno más propenso a fallar o peor clasificado. La herramienta funcionará para modelos que se realicen en el territorio Español ya que es la zona que nuestros datos de referencia

SIGPAC nos aporta pero de conseguir nuevos datos será perfectamente aplicable a otras zonas.

Cabe destacar que todos estos objetivos son los establecidos antes del desarrollo del proyecto por lo que es posible que haya adiciones y sustituciones a medida que se profundice en la materia y el código. A continuación se definen los objetivos del proyecto de manera secuencial en lo referente al desarrollo del código:

1. Script de conversión y pre-procesamiento de datos raster y vectoriales.
    a) Funciones de transformación y proyección de sistemas de coordenadas.
    b) Funciones de conversión de datos vectoriales a raster.
    c) Creación de nuevas imágenes tiff con los datos SIGPAC.
2. Script de validación.
    a) Creación de rasters de comparación de las imágenes clasificadas y las nuevas imágenes con los datos SIGPAC.
    b) Dado el raster anterior obtener matriz de confusión de aciertos y fallos.
    c) Grafos y análisis de los resultados anteriores.
3. Optimización y edición del clasificador.
    a) Realizar pruebas con los hiper-parámetros del modelo y validar de nuevo los resultados con el script anterior.
4. Script de bash que automatice los objetivos 1 y 2 para facilitar el uso del proyecto.

Como objetivos transversales al proyecto para ofrecer un enfoque más realista y presentar mejor la utilidad de la herramienta se buscan definir lo siguientes casos de uso: El primero se basa en la detección de fraudes a la hora de la declaración de terreno que tienen que hacer los poseedores de tierras en SIGPAC, para ello debemos obtener muy buenos resultados de validación del clasificador. Y en segundo lugar se proporcionarán métricas de validación para detectar el porcentaje de acierto de cada tipo de cultivo permitiendo a los expertos agrícolas detectar qué cultivos concretos dificultan la clasificación.

1.3. Estructura del documento

El presente documento se encuentra divido en seis capítulos y un apéndice con sus respectivas secciones y sub-apartados que recaban toda la información relacionada con el desarrollo

de este proyecto.

Tras este capítulo 1 introductorio donde se presenta la materia de estudio comienza el capítulo 2 donde se exponen los conceptos previos y el estado del arte actual en las distintas ramas de conocimiento que se van a tratar durante el desarrollo, así como algunos conceptos básicos acerca de la materia de estudio como el manejo de imágenes y datos geo-espaciales.

En los capítulos 3 y 4 se va a comentar la metodología de desarrollo software usada y el diseño e implementación de la herramienta. El capítulo 3 tendrá un enfoque mucho más teórico acerca de las fases del proyecto, la organización y las tecnologías. Se comentará brevemente como fue el día a día trabajando en el proyecto.

Ya en el capítulo 4 en la implementación se tratarán temas mucho más prácticos sobre el desarrollo del código fuente y el manejo de las herramientas creadas destacando el algoritmo principal de conversión a imágenes SIGPAC.

El capítulo 5 es uno de los más importante de la memoria puesto que expondrá los desafíos encontrados durante el desarrollo y sus respectivas soluciones además de exponer los casos de uso reales que esta herramienta ofrece a los agricultores y a las personas que se dediquen al desarrollo de software con sistemas de información geográfica.

En el capítulo 6 se concluye con el futuro del 'land cover' y su validación además de una breve observación acerca de la información terrestre en general. También se habla del futuro de esta herramienta en específico y cómo se podría mejorar.

Finalmente en el apéndice A se explica al usuario la instalación de la herramienta y el manejo de la misma.

2 Conceptos previos y estado del arte

En este proyecto se hace uso de diferentes fuentes de datos con tipologías muy distintas entre sí, además de numerosas tecnologías y técnicas para el desarrollo de código y modelos de inteligencia artificial. Un buen conocimiento de estas aptitudes nos permitirá obtener un mejor resultado final, que, en nuestro caso será una mejor validación del modelo de predicción de cobertura terrestre. Es por eso que este capítulo comienza hablando sobre nuestro primer tipo de dato que son las imágenes multiespectrales, continuaremos con los sistemas de información geográficas y los tipos de datos con los que se trabaja para terminar en el apartado 2.3 con una breve explicación de los distintos modelos de aprendizaje que se aplican en la cobertura terrestre. El capítulo finaliza con una síntesis de las distintas líneas de investigación actuales respecto al campo de estudio para poder entender mejor el problema al que nos enfrentamos.

2.1. Imágenes multiespectrales

La primera cuestión que se nos puede venir a la mente a la hora de comenzar a trabajar en un proyecto relacionado con la geografía del planeta como la cartografía, es el tipo de dato que se emplea y para ello se usan las imágenes multiespectrales. El origen de estas imágenes nos lleva a 1957 con el lanzamiento del satélite de la unión soviética Sputnik1. No fue hasta dos años después, en 1959, que se tomaría la primera foto del planeta tierra desde el espacio a manos del satélite estadounidense Explorer-6. En la actualidad más de 4000 satélites orbitan la tierra realizando funciones de observación, navegación y comunicación principalmente. En lo que al proyecto respecta según la NASA(National Aeronautics and Space Administration) y los estatutos de libre acceso de datos existen satélites cuya única función es obtener fotos

del planeta cada diez minutos. Para este proyecto se ha trabajado con el satélite Sentinel-2 perteneciente a la ESA [1] que lleva desde 2015 capturando imágenes de la tierra y las costas del planeta [6] así como con el programa ASTER de la NASA [2].

2.1.1. Modelo de elevación digital global

El Modelo de Elevación Digital Gloabl (GDEM) es una representación digital de la superficie terrestre en tres dimensiones. Se trata de un modelo desarrollado por la NASA y de acceso público fundamental en las disciplinas como la cartografía o geomática.

Este modelo ASTER se calcula procesando imágenes captadas con el sensor de Radiómetro de Emisión y Reflexión Térmica Avanzado (ASTER) y aplicando trigonometría a las imágenes resultantes. Gracias a este sensor se obtienen imágenes de alta resolución de la Tierra en 14 longitudes de onda diferentes del espectro electromagnético, que van desde la luz visible hasta la infrarroja térmica. Los científicos utilizan los datos de ASTER para crear mapas detallados de la temperatura de la superficie terrestre, la emisividad, la reflectancia y la elevación.

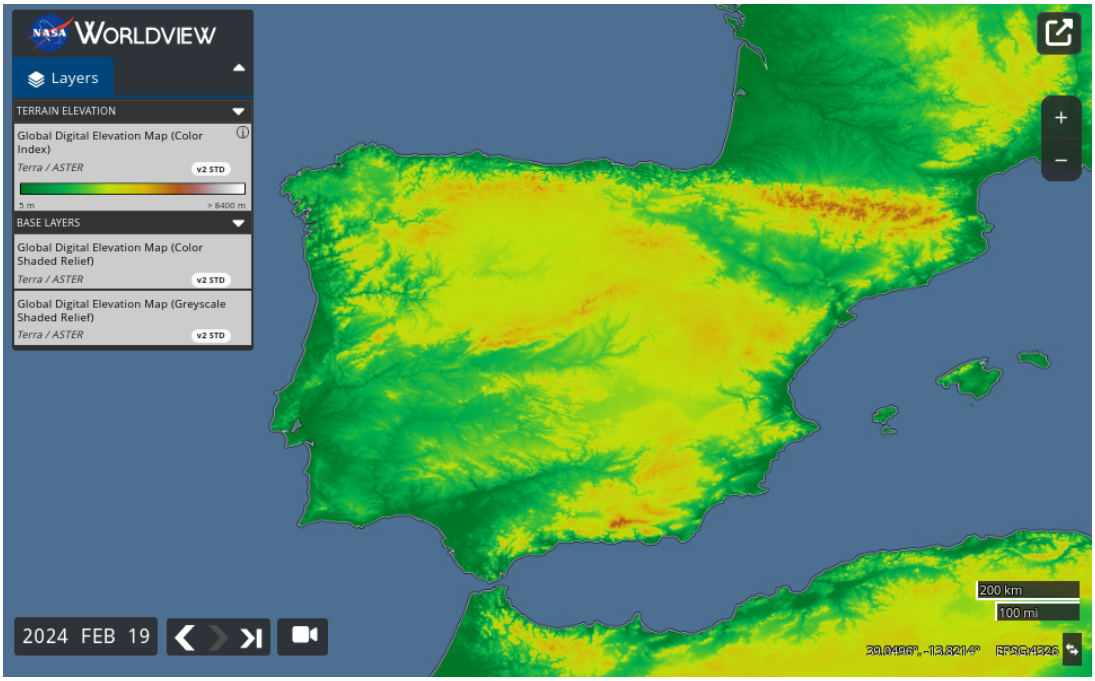


Figura 2.1: Sensor ASTER de la NASA. Fuente: [7]

El ASTER GDEM, en particular, ha sido ampliamente utilizado en la investigación científi-

[1] ESA: https://www.esa.int/
[2] ASTER: https://www.earthdata.nasa.gov/sensors/aster

ca y en aplicaciones prácticas, como la planificación de infraestructuras, la gestión de riesgos naturales y la conservación del medio ambiente.

2.1.2. Sentinel-2

La misión de este satélite del programa Copérnico costa de los satélites Sentinel-2A y Sentinel-2B [3]. Esta pareja de satélites orbitan con una diferencia de 180º respecto al ecuador realizando fotografías de zonas de interés. Esta misión genera datos geoespaciales a nivel local, regional e internacional de acceso libre para la investigación en los campos temáticos para la monitorización; medioambiental, acuática, vegetal y forestal, de reservas de carbono y recursos naturales, campos de cultivo y ordenación del territorio.

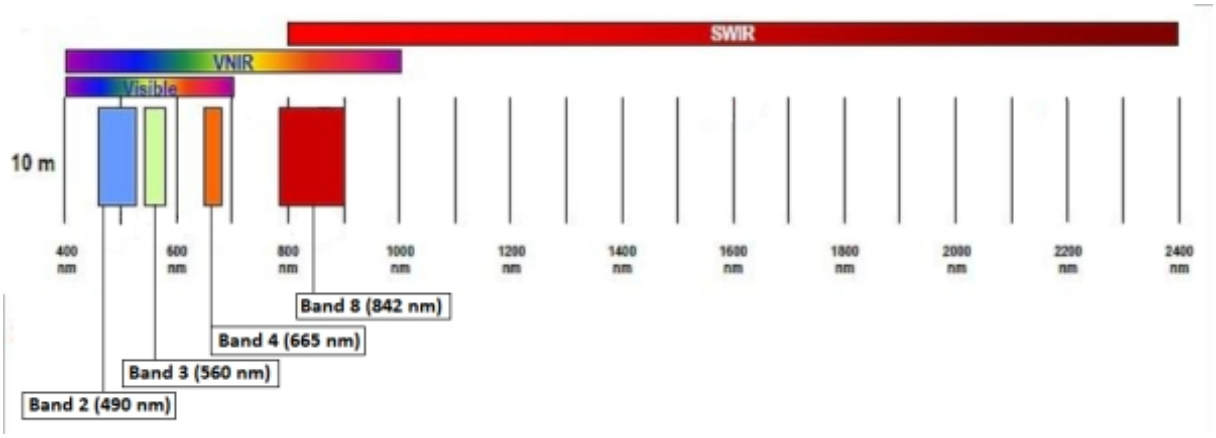
Las imágenes tomadas ofrecen información en trece bandas espectrales diferentes cada una con un tipo de resolución:

- **Espacial:** Es la representación en tierra de un detector del satélite. Esta indica el área espacial que cubre un píxel en la imagen resultante. Si una banda tiene una resolución espacial de 10m indicará que un píxel de la imagen representará 100m² de tierra (área del cuadrado = lado²).
- **Temporal:** Es la frecuencia de visita de un satélite a una ubicación particular. La frecuencia de Sentinel-2 es de diez días y la revisión de constelación combinada de cinco días.
- **Espectral:** La resolución espectral se define como la capacidad del instrumento para distinguir características en el espectro electromagnético. Cada banda puede medir en una escala diferente.
- **Radiométrica:** Esta resolución determina el nivel incremental de intensidad o reflectancia. A mayor resolución mayor precisión del satélite para detectar diferencias de intensidad.

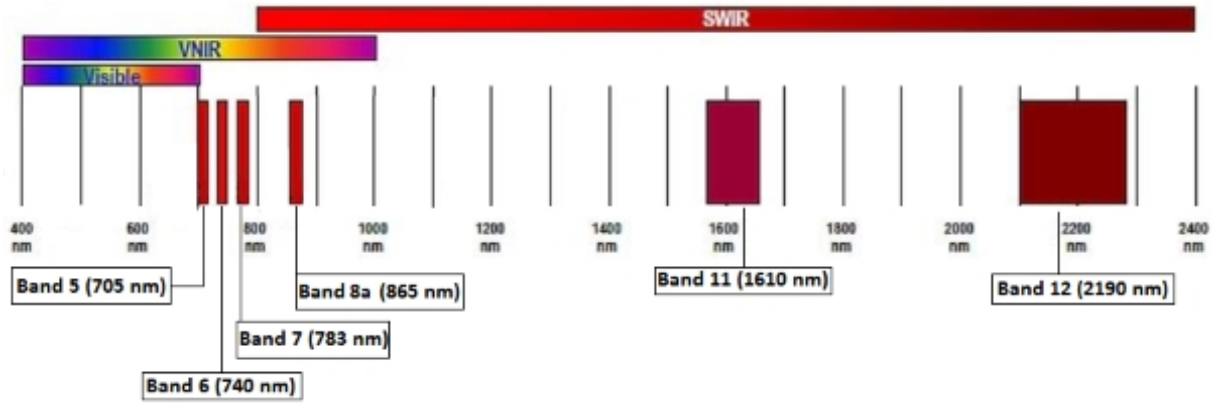
Entre estos tipos las resoluciones espaciales y espectrales son las más interesantes puesto que para detectar el tipo de tierra de la imagen espectral nos vamos a basar en la radiación que emite el cuerpo en el espectro electromagnético. Tal y como vemos en la Figura 2.2 según

[3] Datos Sentiel: https://dataspace.copernicus.eu/explore-data/data-collections/sentinel-data/sentinel-2

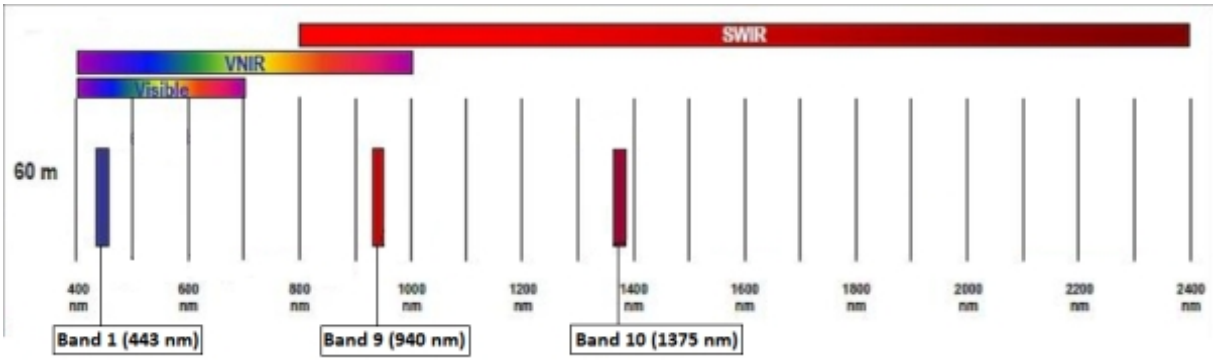
la resolución espacial que estemos usando mediremos unas bandas espectrales u otras.



(a) Teledetección con 10 metros de resolución espacial.



(b) Teledetección con 20 metros de resolución espacial.



(c) Teledetección con 60 metros de resolución espacial.

Figura 2.2: Resolución espectral y resolución espacial de Sentinel-2. Fuente: [8]

Cada píxel será definido con lo que se denomina **firma espectral** que es definido por las trece bandas del satélite. Esta firma espectral refleja las propiedades espectrales de la superficie observada. Por ejemplo, aunque varios píxeles puedan representar el mismo tipo de cultivo, edificación o cuerpo de agua, cada uno tendrá valores espectrales ligeramente diferentes. De

esta forma, al promediar una gran cantidad de datos de una misma zona, podemos identificar los valores espectrales que definen las características de un cultivo o una superficie específica.

Combinando los valores espectrales de diferentes bandas podemos obtener **índices espectrales**, que vienen definidos por fórmulas matemáticas (normalmente lineares). Encontramos multitud de índices a cada cual más especifico, sin embargo, destacamos los índices de vegetación y agua al ser el objetivo de este trabajo la cobertura terrestre. Existen numerosos índices de vegetación entre ellos, el más popular se conoce como NDVI (Normalized difference vegetation index) y se calcula con la siguiente fórmula. Ecuación de índice de vegetación [9]:

$$NDVI = \text{Index}(NIR, RED) = \frac{NIR - RED}{NIR + RED} \tag{2.1}$$

$$NDVI = \text{Index}(B8, B4) = \frac{B8 - B4}{B8 + B4} \tag{2.2}$$

Tal y como vemos en las fórmulas superiores la ecuación 2.1 indica las bandas necesarias para su cálculo que son la roja (B4) y la franja de infrarrojo cercano (B8). La ecuación 2.2 indica los valores del índice para el satélite Sentinel-2 vemos que son necesarias las bandas cuatro y ocho que tal y como aparecen en la figura 2.2a tienen una resolución espacial de 10 m y unos valores de 665 nm y 842 nm de radiación espectral respectivamente.

Gracias a estos valores de las bandas y este índice podemos crear una gráfica del porcentaje de reflectancia respecto al espectro de la banda y obtener el tipo de vegetación en el píxel concreto de estudio tal y como vemos en la Figura 2.3. Ahí vemos como a un mayor porcentaje de reflectancia (vegetación) el cultivo se encontrará en mejor estado, siendo la vegetación sana la que más porcentaje tiene con un 90 %. Vemos como el rango del espectro de estudio es importante para calcular los índices puesto que entre los 500 nm y 600 nm encontramos valores de vegetación seca y sana similares debido a que en ese rango de onda el satélite capta otro tipo de información del suelo y no la vegetación.

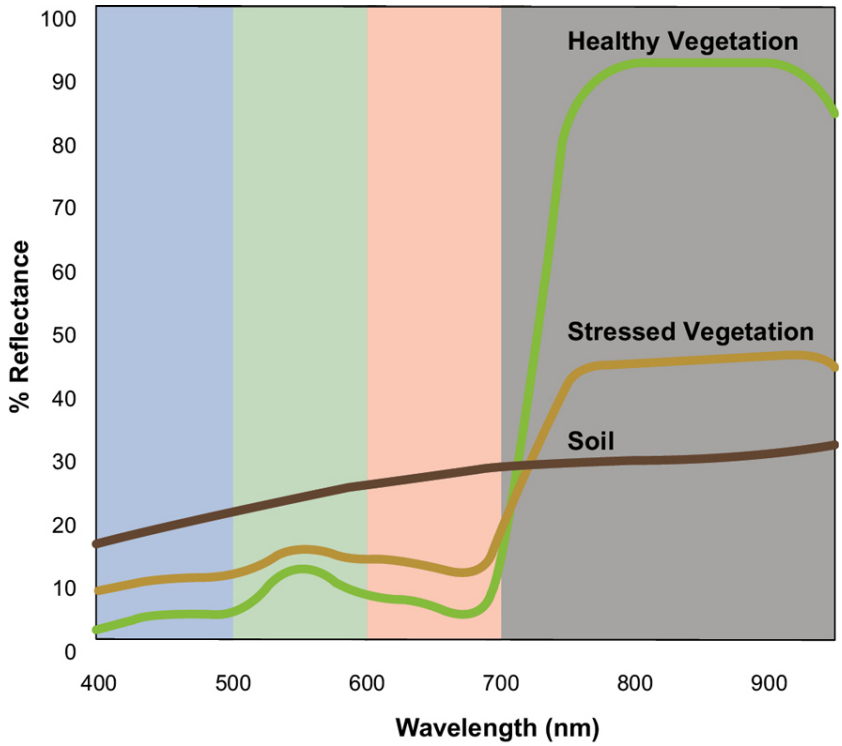


Figura 2.3: Firma espectral de vegetación sana, seca y suelo. Fuente: [10]

Una imagen capturada por los satélites Sentinel se conocen como 'tiles'. Cada 'tile' es una sección rectangular de la superficie terrestre capturada por el satélite. Los distintos 'tiles' suelen estar disponibles en diferentes resoluciones espaciales y temporales, lo que permite una amplia gama de análisis y aplicaciones. En este proyecto se ha trabajado con este concepto.

2.2. Sistemas de información geográficas

Los Sistemas de Información Geográfica (SIG) son herramientas tecnológicas que permiten adquirir, almacenar, analizar y visualizar datos geoespaciales, es decir, información relacionada con la ubicación geográfica de elementos y fenómenos en la Tierra. Estos sistemas integran datos de fuentes que van desde mapas a imágenes satélites. Un sistema SIG consta de:

- **Datos geoespaciales**: Son la base del SIG y pueden provenir de diversas fuentes, como imágenes satélites, fotografías aéreas, sensores terrestres, datos GPS y bases de datos cartográficas. Estos datos contienen información sobre la ubicación, forma, tamaño y atributos de los objetos geográficos.
- **Hardware**: Comprende los equipos informáticos, dispositivos de entrada y salida, y

otros componentes necesarios para la adquisición, procesamiento y visualización de los datos geográficos.

- **Software**: Incluye las aplicaciones informáticas específicas para manejar, analizar y visualizar los datos geoespaciales. Los programas SIG permiten realizar operaciones como la creación de mapas, el análisis de relaciones espaciales, la gestión de bases de datos y la generación de informes.

En nuestro caso nuestros datos geoespaciales constan de coordenadas de latitud y longitud formando polígonos además de una clase que indica que hay en esa zona si bosque, agua o construcción. En la literatura al trabajar con este tipo de datos se usan formatos como geojson, shapefile o kmz. En este proyecto se ha utilizado principalmente el formato shapefile al ser el utilizado por SIGPAC.

Respecto al hardware suelen ser necesarias máquinas con muchos cores para paralelizar los procesos y las gestión de recursos ya que es común trabajar con Tbytes de datos. Finalmente respecto al software vamos a utilizar el software **QGIS** que más adelante en 3.3 comentaré acerca de el.

2.2.1. Datos vectoriales

Los datos vectoriales nos permiten representar las características del mundo real en un sistema de información geográfica (GIS). Cualquier distintivo del paisaje es considerado una clase diferente y cada clase tiene indexada su correspondiente atributo que consiste en una descripción en texto o numérica de lo que representa. Un vector de una clase tiene su forma representada mediante una geometría, las geometrías se crean cuando uno o más vértices de una misma clase están interconectados. Un vértice describe la posición en el espacio usando las coordenadas 'X' e 'Y' . Además existe la coordenada 'Z' en algunas ocasiones utilizada para representar la altura o profundidad como en el plano cartesiano común.

En función del número de vértices la geometría obtenida será diferente: De tener un único vértice nos referimos a un punto, si más de un punto está conectado por una linea recta encontramos una multi-línea o línea de líneas (polyline), finalmente si tres o más vértices están presentes y el último es igual al primero estaremos frente a un polígono. Todo esto se puede ver perfectamente en la figura 2.4, observamos como distintas características del paisaje (clases) estás representadas en la figura. Según la forma de estas utilizamos una forma u otra

de representación en el plano, para los árboles hacemos uso de un punto (rojo), en el caso de las carreteras y los ríos mediante una multi-línea podemos abarcar el terreno que ocupan (verde y azul respectivamente) y finalmente mediante un polígono cerrado marcaremos las construcciones o casas (blanco). [11]

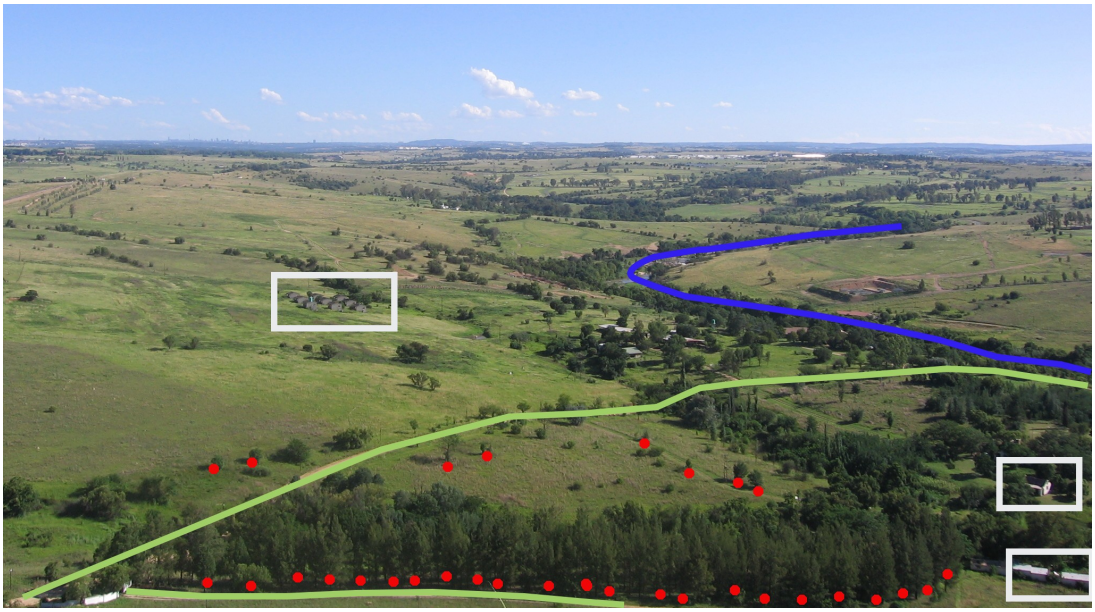


Figura 2.4: Clases representadas en el paisaje. Ríos (azul), carreteras (verde), árboles (rojo), casas (blanco). Fuente: [11]

2.2.2. Mapas de bits

Los mapas de bits son matrices de datos con metadatos geográficos asociados. Este enfoque nos permite abordar los problemas geográficos desde un punto de vista diferente. A diferencia de los datos vectoriales (conjuntos de coordenadas que forman geometrías) en los que guardamos la clase, su atributo y la posición geo-espacial en coordenadas cartesianas, ahora utilizamos una matriz de píxeles o celdas de NxM donde cada posición representa las condiciones del área cubierta por esa celda en cuestión.

Este tipo de mapas es utilizado cuando queremos representar una zona que es continua durante una amplia área y no es fácilmente divisible por datos vectorizados. Observando de nuevo la figura 2.4 vemos como la densidad de hierba no es igual en todo el área de la foto al igual que no todos los árboles poseen las mismas características en cuanto a copa, tronco o altura por lo que al utilizar un polígono para cubrir toda esa zona mucha información se pierde al simplificar y detectaría las distintas densidades de hierba o los distintos tipos de árboles

como una única clase o tipo. Es decir que para zonas no homogéneas los datos vectorizados no son del todo óptimos y pese a que podríamos realizar un polígono por cada variación en el paisaje esto podría llevarnos mucho trabajo.

Mediante los mapas de bits ponemos solución a estos problemas. Es común usar este tipo de matrices para añadir aún más significado a las capas de datos vectorizados obteniendo mapas de información muy completos. Estos mapas no solo nos permiten representar superficie del mundo real como imágenes de satélite también con ellos representamos ideas más abstractas, por ejemplo un uso común de estos mapas es la clasificación de terreno según su riesgo de incendio. Cada celda tendrá un valor del uno al nueve indicando el riesgo de incendio.

Este tipo de datos es obtenido mediante fotos aéreas o imágenes satélites. Respecto a los satélites llamamos teledetección al proceso de toma de la imagen y el envío de la misma a la estación de almacenamiento mediante radio señales. Los satélites al tomar una foto como ya se ha comentado en el apartado 2.3 detectan la reflectancia de la longitud de onda de la superficie de la tierra en registros que superan los visibles por el ojo humano. Este tipo de imágenes 'raster' son llamadas multiespectrales, por lo que una imagen satélite sera representada en datos 'raster' como un mapa de bits por banda espectral que el satélite sea capaz de tomar.

Figura 2.5: Imágenes satélite. De derecha a izquierda mono-banda gris, color verdadero y color falso. Fuente: [12]

Como ya hemos visto en 2.2 el Sentinel-2 obtiene trece bandas espectrales cada una representada por un mapa de bits diferentes. Para entender mejor este apartado he añadido en una aplicación de visualización las bandas RGB (Red, Green, Blue) que son visibles para el ojo humano obteniendo la imagen a color (true color) de una zona de interés, como vemos en la imagen central de la figura 2.5.

Un detalle a tener en cuenta en los datos ráster es la escala con la que se toman los datos. Los datos ráster generalmente son recogidos mediante mapas ya existentes, registros topógrafos o dispositivos de posición global (satélites). Cada mapa puede tener una escala diferente por lo que al incorporar los datos en la una aplicación de visualización de datos podremos tener problemas de escala y que los datos no correspondan con la posición real de los mismos. La solución a este problema se explicará más en detalle en el apartado 2.2.3.

2.2.3. Sistema de referencia de coordenadas

Los sistemas de referencia de coordenadas son cruciales para guardar los datos vectoriales o matriciales digitalmente y conseguir mantener la escala y resolución original. El problema que existe con los datos geográficos viene dado por la forma de la tierra ya que al ser esta un geoide y estar aplastada por los polos siempre que tomemos una medida aérea es necesario calcular el nivel de aplanamiento para obtener la referencia elipsoidal. Ese coeficiente viene dado por la fórmula:

$$f = (a - b)/a \tag{2.3}$$

La fórmula anterior se refiere a la ecuación del aplanamiento elipsoidal donde: 'a' es la longitud del semieje mayor y 'b' la longitud del semieje menor. Entre los numerosos CRS (Coordinate Reference System) que existen a día de hoy reconocemos como estándar aquellos que son geocéntricos, es decir, aquellos cuyo origen coincide con el centro de masas de la Tierra.

Debido a esta curvatura una zona de interés se representa en una elipsoide mediante sus coordenadas cartesianas (X, Y, Z) o sus coordenadas curvilíneas (ϕ, λ, h) tal y como vemos en la figura 2.6. En referente a las coordenadas curvilíneas la latitud se define como el ángulo formado en el plano meridiano entre el plano ecuatorial y el vector elipsoidal normal que cruza el punto 'P', así mismo la longitud se mide en el plano ecuatorial como el ángulo entre el meridiano de Greenwich (eje-X) y el meridiano que atraviesa el punto 'P' , cabe destacar que un mismo punto en la superficie puede tener diferentes coordenadas geodésicas en función del sistema geodésico de referencia que usemos. Entre los sistemas más utilizados encontramos

el GRS80 (Geodetic Reference System 1980), WGS84 (World Geodetic System 1984) o el más actual conocido como GPS (Global Positioning System) [13].

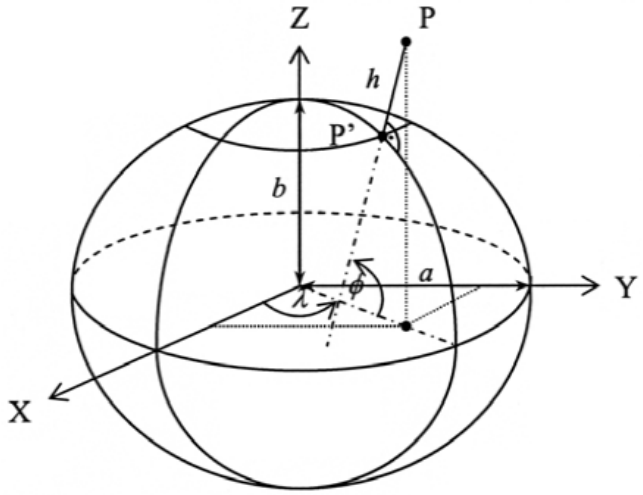


Figura 2.6: Sistema de coordenadas de referencia elipsoidal

Fuente: [13]

Para terminar de entender estos sistemas de referencia a continuación se describen brevemente los bancos de coordenadas más relevantes para este proyecto.

- **WGS84**: El sistema de información geodésica global 1984 es uno de los más utilizados a nivel global. Permite representar puntos con exactitud mediante las coordenadas cartesianas.
- **ED50 and ETRS89**: El Sistema de Referencia Terrestre Europeo 1989, es una versión actualizada del ED50 y bancos anteriores y describe con precisión las posiciones de puntos en la superficie terrestre de Europa y regiones circundantes. [14]

Proyecciones cartográficas

En la práctica en muchas ocasiones es necesario transformar coordenadas geográficas de algún CRS al plano cartesiano de dos dimensiones. Mediante una serie de ecuaciones matemáticas podemos transformar dimensiones en 3D a 2D manteniendo sus respectivas magnitudes y reduciendo los errores al máximo. Para ello a continuación se describe brevemente el sistema de red UTM (Universal Transverse Mercator coordinate system).

El sistema de referencia de coordenadas Universal Transverse Mercator tiene su origen en

el ecuador en una longitud específica. Donde los valores de 'Y' aumentan hacia el sur y los valores de 'X' aumentan hacia el oeste. Para evitar errores y la distorsión creada por la forma de la Tierra, el mundo está dividido en 60 zonas iguales, todas ellas de 6 grados de ancho de este a oeste. Las zonas UTM están numeradas del 1 al 60, comenzando en el antemeridiano (zona 1 a 180 grados de longitud oeste) y avanzando hacia el este hasta llegar al antemeridiano (zona 60 a 180 grados de longitud este) [15]. Conocer como se divide el planeta en los diferentes husos y escoger el cuadrante correcto nos permite corregir la distorsión de la conformidad angular. Por ejemplo si nuestro área de interés fuese Andalucía tendríamos que hacer uso de las zonas UTM29S y UTM30S, tal y como podemos ver en la figura 2.7. En este proyecto se ha utilizado el Sistema de Referencia de Cuadrícula Militar (Military Grid Reference System o MGRS) que se basa en una ampliación del modelo UTM en el cual cada sector se divide zonas de 100.00m² que se nombran de la A a la Z. De esta forma si quisiéramos representar Málaga con esta notación utilizaríamos las zonas UTM30SUF, UTM30SUG, UTM30STF, UTM30SVF. Estas últimas zonas cubren un área mucho menor.

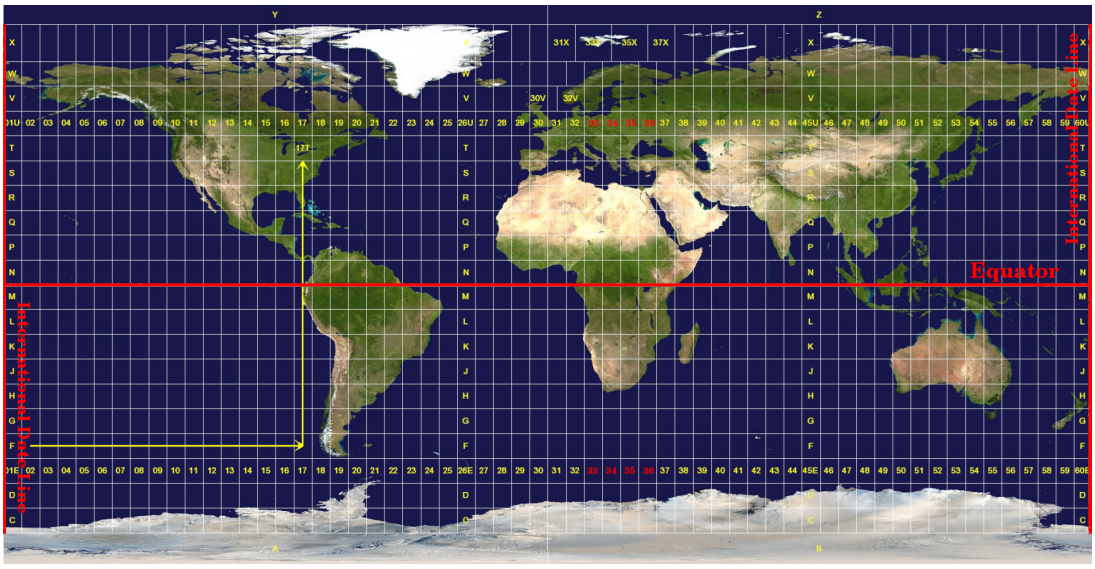


Figura 2.7: Sistema de referencia de coordenadas UTM. Fuente: [11]

2.2.4. Parcelas Agrícolas y Catastro

En esta sección se hace referencia ya al objetivo del proyecto en sí. Tal y como dice el título se ha trabajado con los datos públicos del gobierno de España de parcelas agrícolas y catastro. La iniciativa SIGPAC (Sistema de Información Geográfica de Parcelas Agrícolas y Catastro)

tiene el propósito de facilitar a los agricultores la presentación de solicitudes y documentos para controles administrativos, no obstante a día de hoy esta herramienta es utilizada también en campos como el urbanismo, infraestructuras o la geología. Desde 2005 España ha incorporado una base de datos gráfica de libre acceso de todas las parcelas de cultivo digitalizadas con una precisión de al menos 1:10 metros. Este sistema consta de un mosaico de orto-fotos de todo el territorio nacional con una nomenclatura única para cada parcela. Principalmente las referencias SIGPAC constan de seis elementos distintivos: Primero el año en el que se publicó la información ortográfica, y, a partir de ahí realizamos la búsqueda de nuestra zona de interés de mayor a menor a través de la provincia, municipio, polígono, parcela y recinto hasta llegar al terreno de estudio. Cada zona delimitada por los datos PAC tiene un código de uso que indica que ocupa esa zona desde el tipo de cultivo, si es bosque o si está urbanizado el terreno. Este código de uso que vemos en la tabla 2.1 será fundamental para la validación de los modelos de aprendizaje computacional que se explicará más en detalle en el capítulo 4 [16].

2.3. Modelos de aprendizaje automático en la cobertura terrestre

Como ya se comentó en la introducción la cobertura terrestre se refiere a la distribución y clasificación de la superficie del planeta. El uso de modelos de aprendizaje computacional ha revolucionado el campo de la ciencia geoespacial gracias a la forma en la que se puede procesar y analizar la enorme cantidad de información que es recopilada por satélites diariamente. Este apartado por tanto se centra en explicar el papel de la inteligencia artificial en este contexto.

Entre las muchas funciones que realizan los modelos destacan los de clasificación. Estos modelos aprenden a reconocer y clasificar tipos de terreno mediante datos de teledetección de imágenes multiespectrales, creando mapas detallados y actualizados del planeta. Otra clave es la predicción y detección de cambios o anomalías en la cobertura terrestre histórica para predecir cambios futuros o bien para detectar incendios forestales o la deforestación ilegal. Todos estos ejemplos tienen en común la gran demanda de recursos o la necesidad de datos de alta calidad para no cometer graves errores a la hora de interpretar el resultado final.

A continuación voy a comentar los modelos de inteligencia artificial que más se usan en la literatura analizando sus ventajas y desventajas para la cobertura terrestre.

Uso agrícola	Descripción
AG	Corrientes y superficies de agua
CA	Viales
CF	Cítricos-Frutal
CI	Cítricos
CS	Cítricos-Frutal de cáscara
CV	Cítricos-Viñedo
ED	Edificaciones
EP	Elemento del Paisaje
FF	Frutal de Cáscara-Frutal
FL	Frutal de Cáscara-Olivar
FO	Forestal
FS	Frutal de Cáscara
FV	Frutal de Cáscara-Viñedo
FY	Frutal
IM	Improductivo

Uso agrícola	Descripción
IV	Invernaderos y cultivos bajo plástico
OC	Olivar-Cítricos
OF	Olivar-Frutal
OV	Olivar
PA	Pasto arbolado
PR	Pasto arbustivo
PS	Pastizal
TA	Tierra arable
TH	Huerta
VF	Frutal-Viñedo
VI	Viñedo
VO	Olivar-Viñedo
ZC	Zona concentrada
ZU	Zona urbana
ZV	Zona censurada

Cuadro 2.1: Tabla de Uso Agrícola de los datos SIGPAC. Fuente: [16]

Teorema de Naive Bayes

Este teorema se usa principalmente por su velocidad, eficiencia y capacidad de manejar grandes cantidades de datos, así como su insensibilidad a características irrelevantes considerando cada caso independiente. Esto último es interesante puesto que sería lógico pensar que si un píxel es bosque los contiguos también lo serán aunque realmente la distribución de la cobertura terrestre no tiene porque seguir un patrón. Aún con todo esto este modelo suele ofrecer resultados interesantes. [17]

Vecino más cercano (KNN)

Los modelos K-Nearest Neighbour (KNN) nos ofrecen un enfoque no lineal. En este tipo de análisis nos basamos en las características de similitud de vecinos donde la predicción se realiza teniendo en cuenta el 'K' más cercano. Donde 'K' es un número real que oscila entre

uno y el número de clases espaciales. Este modelo de forma similar al anterior busca reconocer patrones entre las clases, no obstante no es muy escalable a largo plazo por lo que su uso dependerá enormemente del número de muestra que tengamos.

Arboles de decisión

Los clasificadores basados en árboles de decisión suelen ser por los que más se decantan los investigadores a la hora de elegir modelo. Entre ellos quiero destacar los Random Forests (RF) en el cual cada árbol de decisión se entrena en un subconjunto aleatorio de muestras de entrenamiento, la misma muestra puede ser seleccionada para entrenar todos, varios o incluso ninguno de los árboles de decisión que componen el conjunto. Esta división se realiza considerando un número de características (típicamente la raíz cuadrada) que se eligen al azar [18]. Los RF ofrecen una alta precisión de clasificación aunque con el riesgo de producir sobre-ajuste, también son muy robusto frente a valores atípicos.

Durante el proceso de entrenamiento del modelo los datos son separados en entrenamiento, validación y test. Una vez se entrena el modelo, se utiliza el conjunto de validación para ajustar los hiper-parámetros y seleccionar la mejor configuración en nuestro caso los datos de validación son los proporcionados por SIGPAC por lo que la validación se podrá realizar con un conjunto de datos completo acerca de todos los puntos de la península ibérica. Finalmente para evaluar el rendimiento del modelo se se utilizan los datos de test. El conjunto de prueba realiza una una evaluación independiente y no sesgada del rendimiento del modelo en datos no vistos anteriormente.

2.4. Importancia de la cobertura terrestre

La cobertura terrestre desempeña un papel cada vez más crítico en nuestro mundo actual debido a su impacto en áreas clave como la conservación del medio ambiente, la lucha contra el cambio climático, la planificación urbana, la seguridad alimentaria y la investigación científica. Comprender cómo se distribuyen y utilizan las distintas superficies terrestres, desde ecosistemas naturales hasta áreas urbanas, es esencial para abordar problemas como la pérdida de biodiversidad y la preservación de hábitat naturales. Además, la cobertura terrestre contribuye significativamente a la mitigación del cambio climático. También tiene un impacto directo

en la planificación de ciudades sostenibles y en la gestión de recursos, y su estudio impulsa avances tecnológicos en teledetección y sistemas de información geográfica. En conjunto, la cobertura terrestre es un campo interdisciplinario de gran relevancia que desempeña un papel crucial en la toma de decisiones para un futuro más sostenible y equitativo en nuestro planeta.

3 Metodología para el desarrollo software

3.1. Selección de la metodología

Para la elección de la metodología de desarrollo se han planteado distintas opciones para realizar el proyecto. El trabajo se ha realizado en todo momento en individual con el apoyo de los tutores, por lo que métodos de desarrollo ágiles se adecuan a la situación puesto que un desarrollo iterativo e incremental puede permitirnos estudiar la materia al mismo tiempo que la desarrollamos. Al comienzo se planteó usar un modelo en cascada o prototipado, sin embargo, estos modelos no incorporan reuniones con el tutor que sirvan de guía para el proyecto por lo que estas opciones quedan descartadas.

En consecuencia se ha elegido scrum como modelo a seguir. Al aplicar scrum a un equipo de desarrollo de una persona con unos tutores se ha adaptado la metodología para incorporar los requisitos de los que requiere el proyecto. Como es común en todos los trabajos lo primero es realizar una fase de planteamiento inicial en la que se declaran los distintos responsables de cada tarea, jerarquizando a los participantes en líderes de proyecto o no. Como ya se ha comentado se va a adaptar el modelo de manera que los tutores tendrán el control del proyecto e indicarán las medidas necesarias para su correcto desarrollo. El mismo desarrollador se encargará de contabilizar las tareas y de comunicar cualquier incidencia que haya a los tutores. Los sprints serán quincenales y en ellos se debatirá si se han conseguido los objetivos previstos y se comunicará al desarrollador las próximas tareas. En todo momento el desarrollador si así lo ve necesario podrá realizar las tareas con el orden de prioridad que prefiera. En scrum realizamos constantemente estimaciones de tiempo para la realización de las tareas, sin embargo en

esta versión la temporización viene dada por la escuela de informática puesto que en ningún caso el proyecto puede rebasar las 296 horas. Por lo que las tareas para cada sprint tendrán una temporización estimada de entre 5-15 horas, no obstante una tarea se podrá prolongar por varios sprints, de esta manera la carga del trabajo queda bien repartido entre todas las fases del desarrollo. [19]

3.2. Planificación del proyecto

Cada quincena (duración del sprint) se han ido añadiendo tareas en función del estado del proyecto. Al empezar de cero con un tema tan novedoso, se comenzó con un estudio exhaustivo de la materia con tareas de búsqueda de información y documentación. A continuación el desarrollador comenzó a familiarizarse con los datos del proyecto realizando pruebas y desarrollando las bases de lo que sería el código. Más adelante se desarrolló el código al completo para finalizar con la creación de gráficas y métricas estadísticas para la validación de los resultados obtenidos. Por último se han desarrollado pruebas unitarias al código para verificar su correcto funcionamiento.

A la tabla 3.1 falta por sumarle la redacción de esta misma memoria a la que aproximadamente hemos asignado 46 horas llegando por tanto al cupo de horas que la escuela exige.

3.3. Selección de herramientas y tecnologías

La primera cuestión respecto a la selección de tecnologías es el lenguaje de programación. Esta decisión es crucial para el proyecto puesto que de ella dependen los objetivos y requisitos específicos marcados, además del propio conocimiento del lenguaje en cuestión ya que la elección de un lenguaje con el que no se este familiarizado puede ralentizar mucho la cadena de tareas. Para tomar la decisión se tomaron en cuenta los entornos de programación que mejor procesan los datos GIS (Geographic Information System), es por eso que se van a analizar las siguientes opciones: **Matlab, R y Python**.

Todas las opciones son lenguajes de programación con los que ya se ha trabajado previamente durante el grado por lo que ninguno de ellos presentaría ningún problema a la hora de

Fase	Tarea	Tiempo estimado
Investigación	Lectura de artículos	20 h
Investigación	Estudio de la documentación de las aplicaciones	10 h
Investigación	Estudio de los datos y código del grupo	15 h
Diseño	Preparación del entorno de trabajo	10 h
Diseño	Planificación del desarrollo	5 h
Implementación	Descarga de los datos	10 h
Implementación	Procesamiento de los datos	20 h
Implementación	Mapeo de los datos descargados	40 h
Implementación	Obtención de gráficas y métricas estadísticas	30 h
Implementación	Validación del RandomForestrespecto a las métricas	30 h
Refactorización y pruebas	Refactorización del código	20 h
Refactorización y pruebas	Pruebas unitarias	15 h
Despliegue	Script de bash para facilitar la ejecución del código	20 h
Despliegue	Reunión final y entrega del proyecto	5 h

Cuadro 3.1: Estimación de las horas invertidas en el proyecto dividido por fases y tareas. Total 250 horas de trabajo

desarrollar código ergo los tres parten con las mismas ventajas. Matlab es un entorno de programación que destaca por su capacidad de computo, de los tres es probablemente el lenguaje con mayor capacidad de análisis matemático por lo que su uso nos podría facilitar las tareas de proyección de planos y conversión de unidades, además posee una amplia biblioteca de funciones para el desarrollo, sin embargo a la hora de utilizar datos GIS este requiere de extensiones de terceros que pueden a largo plazo traer problemas de dependencias y versiones por lo que queda descartado. Las dos opciones restantes son bastante sólidas ya que ambos poseen una muy buena integración con datos GIS así como amplias librerías de inteligencia artificial de las que Matlab carece. de R destaca la capacidad de cálculo estadístico avanzado y sus gráficos personalizables y de Python destacamos principalmente su versatilidad. Finalmente se ha optado por el lenguaje de programación Python por la gran cantidad de módulos y librerías de calidad de vida que posee agilizando y automatizando procesos que nos permitirá mantener la temporización estimada en 3.1. Otro dato por el que optamos por este lenguaje es que QGIS está realizado en Python.

Para la visualización de los datos vectoriales e imágenes tiff (Tag Image File Format) se ha seleccionado la aplicación **QGIS** [11] puesto que nos ofrece un entorno gráfico con el que leer los metadatos e ir comprobando que los resultados obtenidos se adecuan a los objetivos previstos. Existen alternativas como ArcGIS pero fue descartada rápidamente al ser de pago y más compleja.

En relación al sistema operativo se ha trabajado con **Linux** por las ventajas que nos presenta la terminal a la hora del manejo de grandes cantidades de archivos y sobre todo para realizar conexiones ssh con máquinas virtuales puesto que el proyecto puede escalar a grandes áreas de estudio y máquinas más potentes serían necesarias.

Finalmente se escogió **Git** para el control de versiones de forma que en todo momento se pueda rastrear y gestionar cada nueva función que se añada al código. En concreto se ha utilizado github como aplicación git y donde se almacena el código fuente del proyecto [20].

Docker

Docker es una plataforma que permite la creación, el despliegue y la ejecución de aplicaciones en entornos aislados llamados contenedores. El *Dockerfile* es un archivo de texto que contiene una serie de instrucciones que especifican los componentes y configuraciones

necesarios para construir un contenedor. En este proyecto, se ha utilizado el *Dockerfile* para definir el entorno de ejecución de la aplicación, lo que garantiza su portabilidad y facilita su implementación en distintos sistemas operativos y entornos sin preocuparse por posibles inconsistencias o dependencias [21].

3.3.1. Librerías utilizadas

A continuación se comentan las librerías más utilizadas en el proyecto y el motivo de su elección, asi como una breve introducción de cada una de ellas.

Rasterio y Fiona

El manejo de estas dos librerías fue clave a la hora del desarrollo del proyecto. Comenzando con **Fiona**, este paquete de python permite trasmitir metadatos de archivos *GeoPackage* o *Shapefile*. Ofrece multitud de módulos para leer y escribir datos de características simples en formatos GIS de múltiples capas [22].

Por otro lado la librería de **RasterIO** utiliza *GeoTIFF* y otros formatos para la lectura y escritura de los conjuntos de datos rasterizados en forma de cuadrícula, como imágenes de satélite y modelos del terreno, nos proporciona una API basada en matrices numpy multidimensionales y *GeoJSON* [23].

Matplotlib

Gracias a la librería **matplotlib** [24] y sus funciones hemos podido generar grafos que faciliten la visualización de las métricas obtenidas. Se ha seleccionado esta herramienta por sus opciones de personalización de grafos y ejes. Además gracias a su integración con Numpy esta librería muestra los datos almacenados en matrices de forma muy cómoda.

Numpy

La librería *NumPy* (Numerical Python) es una de las librerías de *Python* más utilizadas por los desarrolladores para la computación científica y numérica debido a sus estructuras de alto rendimiento [25].

Numpy es tan efectivo por varias razones: El primer motivo es que nos proporciona herra-

mientas que integran código en *C*, *C++* y *Fortran* en *Python*, estos lenguajes ofrecen tiempos de ejecución mucho menores a lo que *Python* es capaz. Además la librería permite el uso de *ndarrays* (N-Dimensional Arrays). Estos arrays especiales se basan en el principio de **proximidad de referencias** y se basa en la tendencia del procesador a acceder al mismo conjunto de memoria durante un breve periodo de tiempo. Existe la localidad espacial y temporal que hacen referencia al uso de datos dentro de ubicaciones de almacenamiento relativamente cercanas y a la reutilización de datos específicos y/o recursos en un breve lapso de tiempo. Estos conceptos hacen que al realizar operaciones matemáticas la proximidad de los datos en memoria permiten que el procesador realice operaciones de manera más rápida, además la localidad facilita la implementación de operaciones vectorizadas en *numpy* lo que permite realizar cálculos en lotes de datos en lugar de iterar elemento por elemento en un bucle.

Numba

Numba es otra librería muy popular de *Python* aunque no tan utilizada como numpy. En pocas palabras se trata de un compilador *jit* (Just-in-time) que nos permite exprimir al máximo la eficiencia de operaciones de matrices, funciones y bucles de *Numpy*. La forma de utilizar Numba es a través de su colección de decoradores que se aplican a las funciones para indicar a Numba que las compile. Es por esto que se recomienda el uso de Numba solo en funciones con mucha carga matemática o con muchos bucles. Su funcionamiento se resume en la lectura de los códigos de bytes de la función decorada y lo combina con la información de los tipos de argumentos de entrada. Tras esto analiza y optimiza el código para finalmente utilizar la biblioteca del compilador **LLVM** para generar una versión en código de máquina de su función, adaptada a las capacidades de la CPU que ejecute el código [26].

4 Implementación de la herramienta

4.1. Diseño

En este apartado vamos a comentar la arquitectura del software, así como la división del código en sus respectivos módulos y *scripts*. Todo el software creado se divide principalmente en cuatro partes, que atienden a las siguientes funciones:

- Descarga y procesamiento de los datos.
- Creación de raster y métricas.
- Validación de los modelos.
- Script de linux automático.

Para comenzar la implementación primero debemos de preparar el entorno de trabajo. Para ello se recomienda utilizar entornos virtuales de python con los que aislar las dependencias y versiones del proyecto para evitar problemas a futuro al actualizar el software. El proyecto se ha realizado con una arquitectura establecida por lo que para el correcto funcionamiento de la aplicación tendremos que tener un entorno de trabajo similar al que se enseña en este capítulo 4. El siguiente diagrama muestra a grandes rasgos el funcionamiento de la herramienta 4.1. Observamos como el primer paso (bloque morado) es la descarga de los datos que serán guardados en las carpetas `/data/shp_data` y `/data/shapefiles` respectivamente se requiere de ambas carpetas para que la herramienta funcione puesto que la aplicación buscará los datos requeridos en ellas. El diagrama continúa con el archivo *launch.sh* el cual me voy a saltar por ahora puesto que este automatiza los procesos del bloque de python (en amarillo). Este

bloque consta principalmente de tres *scripts*. Cada uno tiene una serie de módulos y funciones que manipulan los datos ya descargados y crean nuevas imágenes, por lo que nuevos directorios de trabajo se crearán en su ejecución. Finalmente, para validar los resultados y el modelo ejecutaremos el *script* de validación y el de entrenamiento, este último paso lo repetiremos hasta que los resultados obtenidos por la validación sea acorde a lo esperado una vez así sea guardaremos los metadatos del modelo y habremos finalizado.

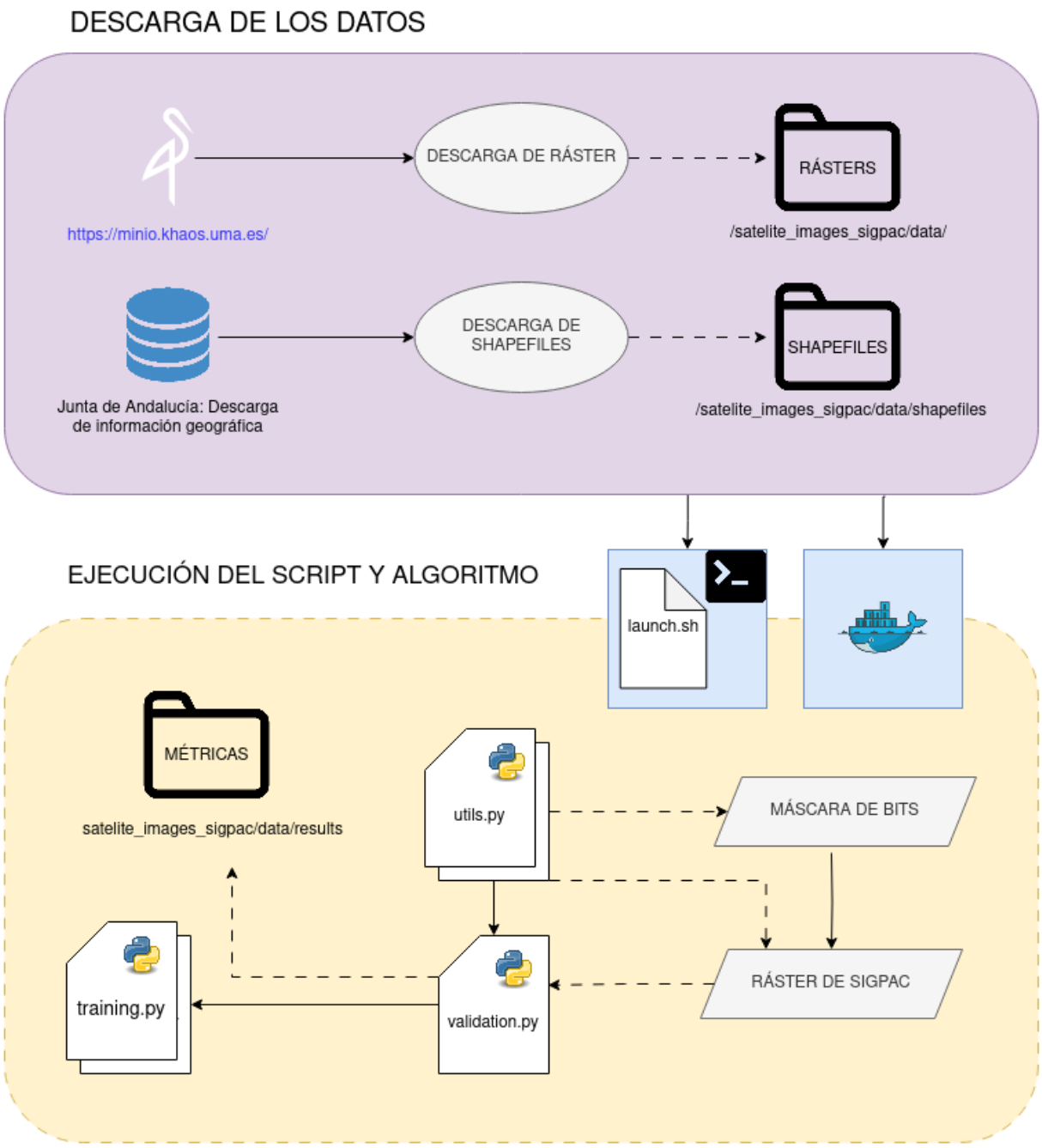


Figura 4.1: Diseño de la arquitectura de la aplicación

Para aclarar el diagrama de 4.1 cabe destacar que los archivos con múltiples fotos como utils.py y training.py se ejecutan de forma paralela con multiproceso. Los rombos de la derecha como máscara de bits y ráster de sigpac son archivos auxiliares que se crean durante la ejecución del algoritmo. Finalmente destacar que el script puede ejecutarse tanto por separado con cada uno de los archivos o bien desde el script de bash launch.sh, de la misma forma se ha dockerizado la aplicación para facilitar su ejecución y una demo.

A continuación, en los siguientes apartados se explicará de manera técnica y más en detalle todos los pasos seguidos para el desarrollo de la herramienta. Desglosando uno a uno los todos los fundamentos previos explicados en el estado del arte 2.

4.2. Descarga y procesamiento de los datos

4.2.1. Datos raster iniciales

Este primer apartado consiste en la descarga y análisis de los datos raster iniciales para comenzar el proyecto. Para ello comenzamos descargando los raster de clasificación que queramos validar, en mi caso, estos datos me los ofrece el grupo de investigación Khaos Research [1]. Ellos utilizan *MinioCloud Storage* como gestor de archivos y datos. Esta herramienta permite almacenar objetos y es tan popular debido a su alta escalabilidad e integración con otras herramientas y servicios. Para sacar partido de esta herramienta crearemos un script que busque entre los cubos de datos de *MiniO* las imágenes con las que queramos comenzar a trabajar. Para facilitar el desarrollo se comenzó trabajando con la provincia de Málaga aunque finalmente se realizó un seguimiento de Andalucía entera. Las *tiles* descargadas en código UTM fueron: 30STF, 30SUF, 30SVF y 30SUG.

La figura 4.2 la visualizo mediante el programa QGIS [11] ya mencionado, y es el resultado final que el modelo de clasificación del grupo de investigación obtiene. Esa imagen solo muestra la zona SUF, por lo que Málaga no esta completamente cubierta, y habría que añadir las otras tres zonas mencionadas. Vamos a comenzar analizando el formato de la imagen. La imagen mostrada se encuentra en sistema de coordenadas de referencia *EPSG:32630 del Registro público de datos geodésicos* - WGS84. Tal y como observamos en la figura la imagen esta en escala de grises cada píxel tiene un color indicando el tipo de terreno que cubre. Esto es

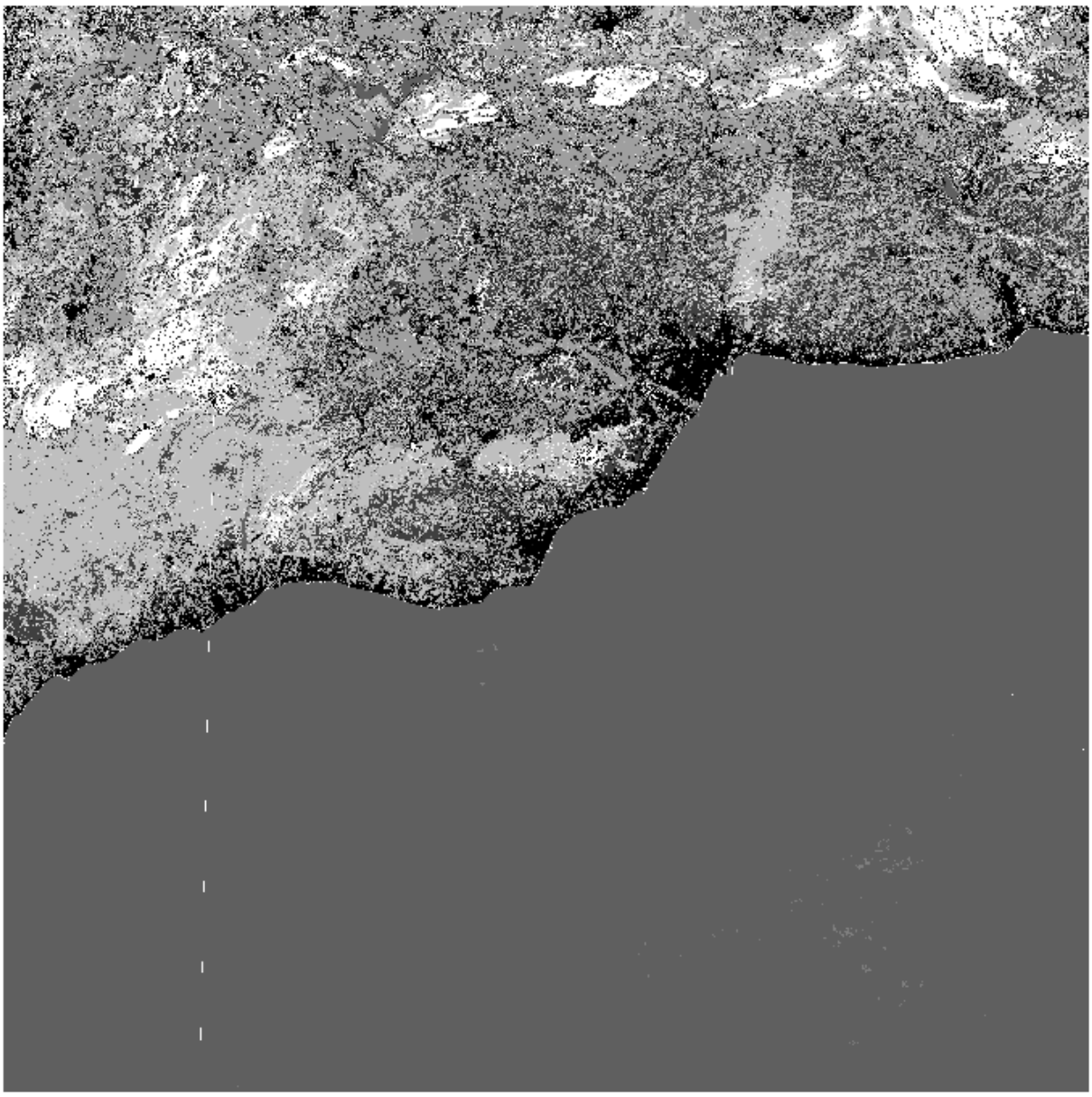


Figura 4.2: Raster clasificado de la zona UTM30SUF. Fuente: [1]

así debido a que se trata de una imagen tiff,(Tag Image File Format) en la que cada píxel de la imagen tiene indexado un valor o descripción, de esta forma, la clasificación está dividida en nueve bandas y cada una representa un tipo de terreno. La clasificación utilizada está basada en el proyecto CORINE de la agencia europea de medio ambiente [3]. Cada valor representa un color con el siguiente tipo de terreno:

0. Dato sin clasificar. (BLANCO)
1. Construcción. (ROJO)
2. Vegetación herbácea. (AMARILLO)
3. Matorral. (NARANJA)
4. Agua. (AZUL OSCURO)
5. Humedal. (AZUL CLARO)
6. Cultivo. (ROSA)
7. Bosque cerrado. (VERDE OSCURO)
8. Bosque abierto. (VERDE CLARO)
9. Roca/Tierra sin vegetación. (GRIS)

Cada píxel de la imagen representa 10 metros (de lado) de superficie real, por lo que cada uno cubre una zona de 100 m^2, en una sola imagen encontramos encontramos 10.980 píxeles de ancho y largo por lo que en total la imagen está compuesta por **120.560.400** píxeles. La zona cubierta por una sola imagen es de aproximadamente 12.056.040.000 m^2 o 12.056,04 km^2. En la imagen vemos como gran parte del terreno es agua por lo que aquellas zonas con costa serán filtradas y reducidas enormemente. Una imagen tiff de este tipo ocupa del orden de 120 mega bytes. Para poner en contexto la península ibérica consta de 504.782 km^2 de terreno por lo que una vez que comencemos a trabajar con provincias enteras o la comunidad autónoma de Andalucía comenzaremos a trabajar con imágenes que pesen giga bytes de datos llegando a los centenares para la totalidad de España. La aplicación *QGIS* nos permite otorgar estilos a las capas para facilitar la visualización la figura 4.3 muestra la imagen anterior con el estilo de capas nuevo. Ahora podemos distinguir algunas zonas como la ciudad de Málaga (ubicada de rojo en la zona centro derecha de la figura) o el parque nacional de las Sierras de las Nieves (ubicado de verde en la zona izquierda). En la enumeración anterior 4.2.1 se indica que color corresponde a cada tipo de terreno.

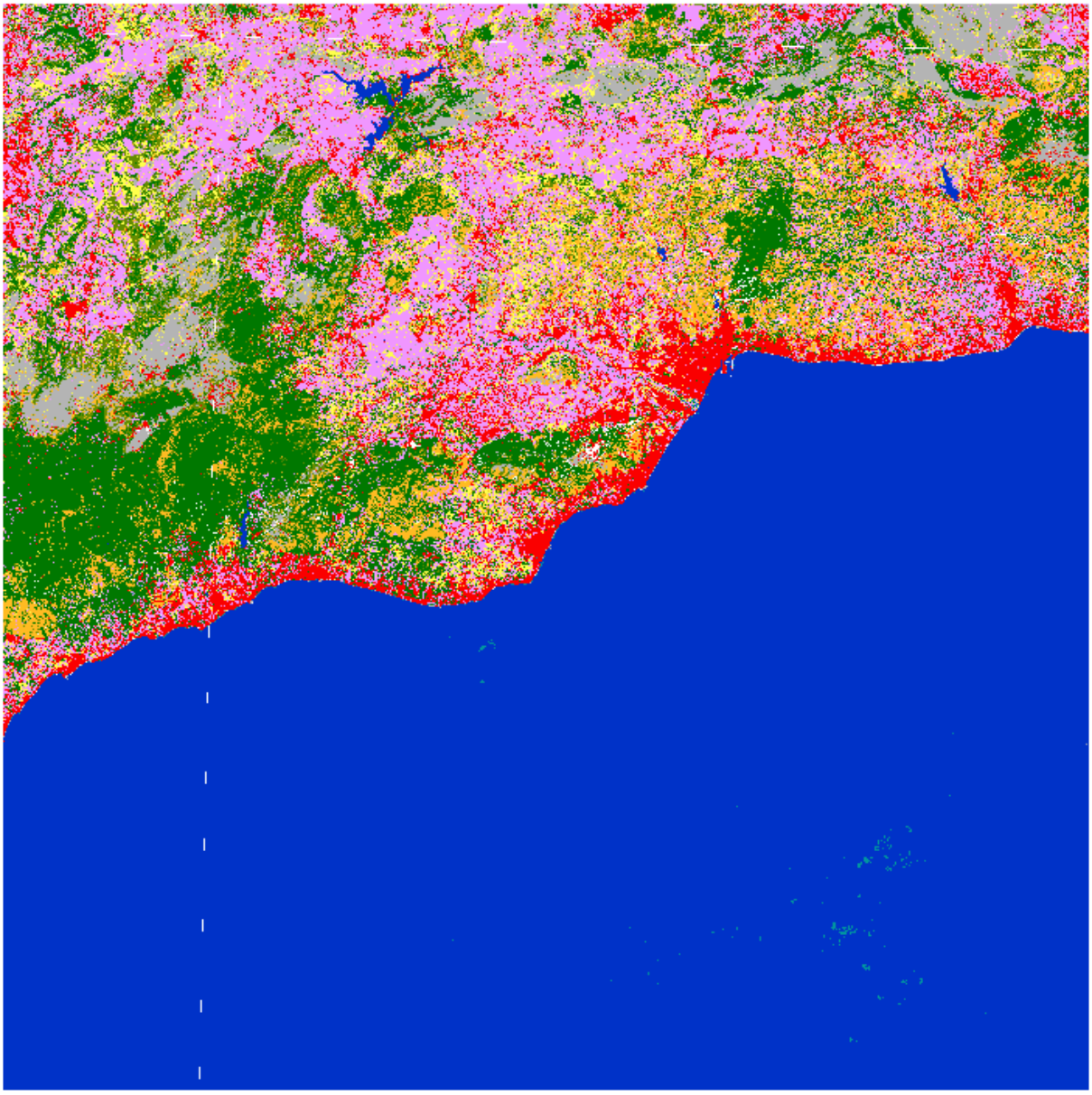


Figura 4.3: Raster clasificado de la zona UTM30SUF con el nuevo estilo. Fuente: [1]

4.2.2. Descarga de Shapefiles

Los datos vectoriales son los datos de descarga que nos ofrece el Ministerio de Agricultura, Pesca y Alimentación. Como vamos a centrarnos en la provincia de Málaga accederemos a los datos SIGPAC proporcionados por la Junta de Andalucía [27]. Dentro de este tipo de datos, la Junta de Andalucía nos los ofrece en formato *shapefile*. Este tipo de archivo es común en aplicaciones GIS y los detectamos por la extensión `.shp`. Estos archivos son sencillos y no topológicos y sirven para almacenar la ubicación geométrica y la información de atributos de

las entidades geográficas. Los *shapefiles* definen estas geometrías en tres o más subarchivos con extensiones concretas que deben de estar almacenados en en el mismo directorio de trabajo para su correcto funcionamiento. A continuación vamos a analizar cada extensión una a una para entender mejor el funcionamiento de este tipo de archivos.

- **.shp**: es el archivo principal y obligatorio que almacena la geometría de la entidad.
- **.shx**: archivo necesario que indexa las geometrías de la entidad.
- **.dbf**: último archivo obligatorio que almacena información de los atributos.
- **.sbn y .sbx**: guardan los índices espaciales.
- **.fbn y .fbx**: guardan los índices espaciales solo para shapefiles de lectura.
- **.ain y .aih**: almacenan los índices de atributos de los campos activos en tablas.
- **.ixs y .mxs**: índice de geocodificación para shapefiles de lectura y escritura.
- **.prj**: almacena la información del sistema de coordenadas.
- **.xml**: metadatos adicionales de información del shapefile.
- **.cpg**: archivo opcional que especifica el conjunto de caracteres utilizado.

Este tipo de archivos son ampliamente utilizados en la industria por su versatilidad y compatibilidad que ofrece. Gracias a que también pueden contener información no espacial en forma de tabla. Estos archivos permiten almacenar información de puntos de interés, líneas o polígonos con información descriptiva de sus atributos indexados siendo la mejor alternativa para los datos geoespaciales. La Junta de Andalucía permite descargar los datos por provincia o municipio. Al subir un archivo *shapefile* en QGIS podremos ver algo similar a la siguiente figura.

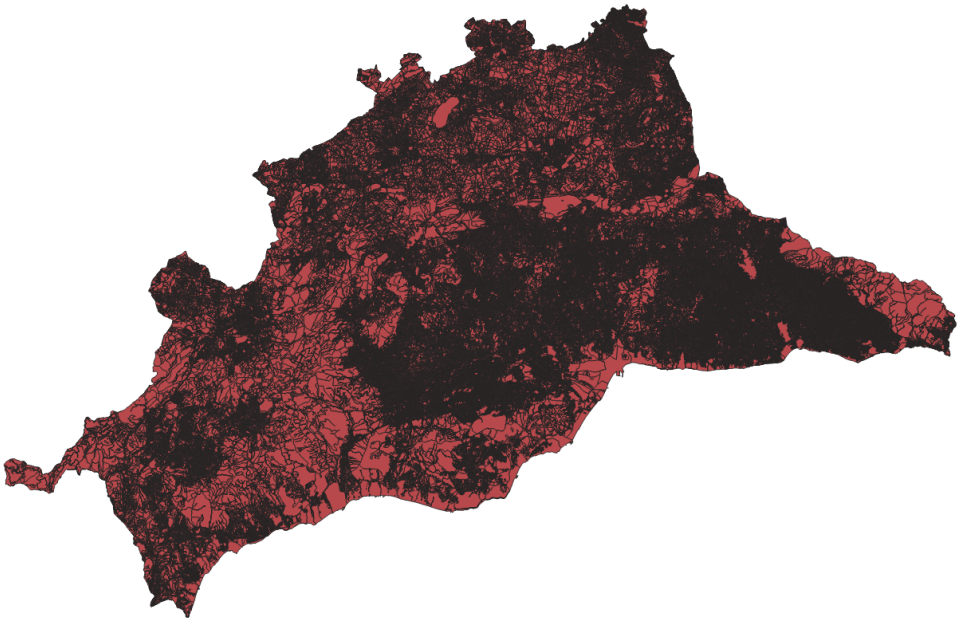


Figura 4.4: Archivo shapefile de la provincia de Málaga.

El *shapefile* mostrado representa la provincia de málaga al completo abarcando muchísima información ocupando 560 mega bytes para entender mejor el formato *shapefile* analizaremos un municipio específico.

(a) Municipio de Alhaurín el Grande. (b) Parcela específica dentro del municipio.

Figura 4.5: Visualización de archivos shapefile.

En la figura 4.5a vemos que ni escogiendo un municipio de los ciento tres que existen en la provincia de Málaga conseguimos aún discernir entre las diferentes parcelas. En la figura 4.5b se ha marcado una concreta, vemos como están trazados los diferentes polígonos que representan las parcelas. A diferencia de los datos raster del anterior apartado estos archivos pertenecen al sistema de referencia de coordenadas *EPSG:25830 - ETRS89* y tienen un límite de precisión de hasta un metro siendo diez veces más precisos que los datos anteriores. Gracias a la precisión de estos datos es que este proyecto se puede llevar a cabo puesto que el clasificador tiene una precisión de 100m^2 por píxel.

Entre los atributos que aparecen en los *shapefiles* encontramos un conjunto de datos geoespaciales, cada parcela se identifica por un ID, un código de provincia y municipio, el área de la parcela, la pendiente media del terreno, un código de uso del terreno y por último la información geométrica del terreno que como se comentó en 2.2.1.

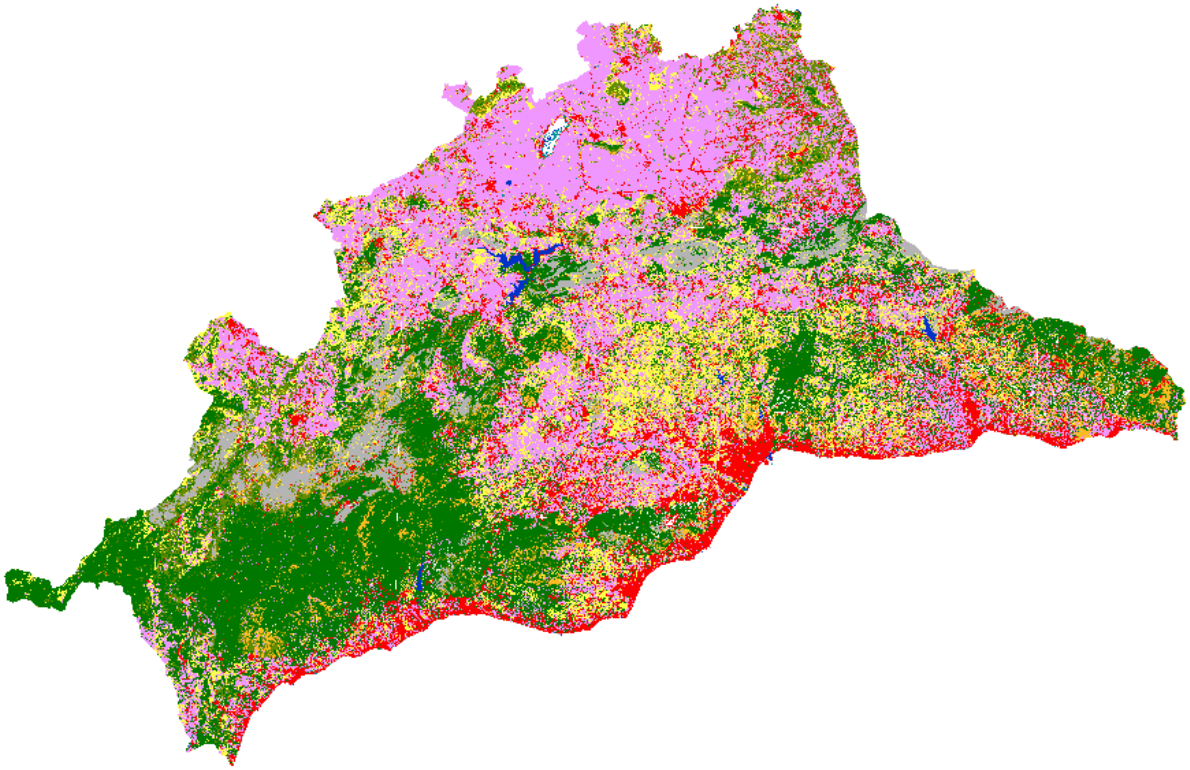
4.3. Creación de raster y métricas

Tras descargar los datos, analizarlos y entenderlos podemos comenzar con la parte más técnica del proyecto que es la proyección de rasters y coordenadas y la creación de nuevos mapas de bits con la información declarada en SIGPAC (Sistema de Información de Parcelas Agrícolas y Catastro).

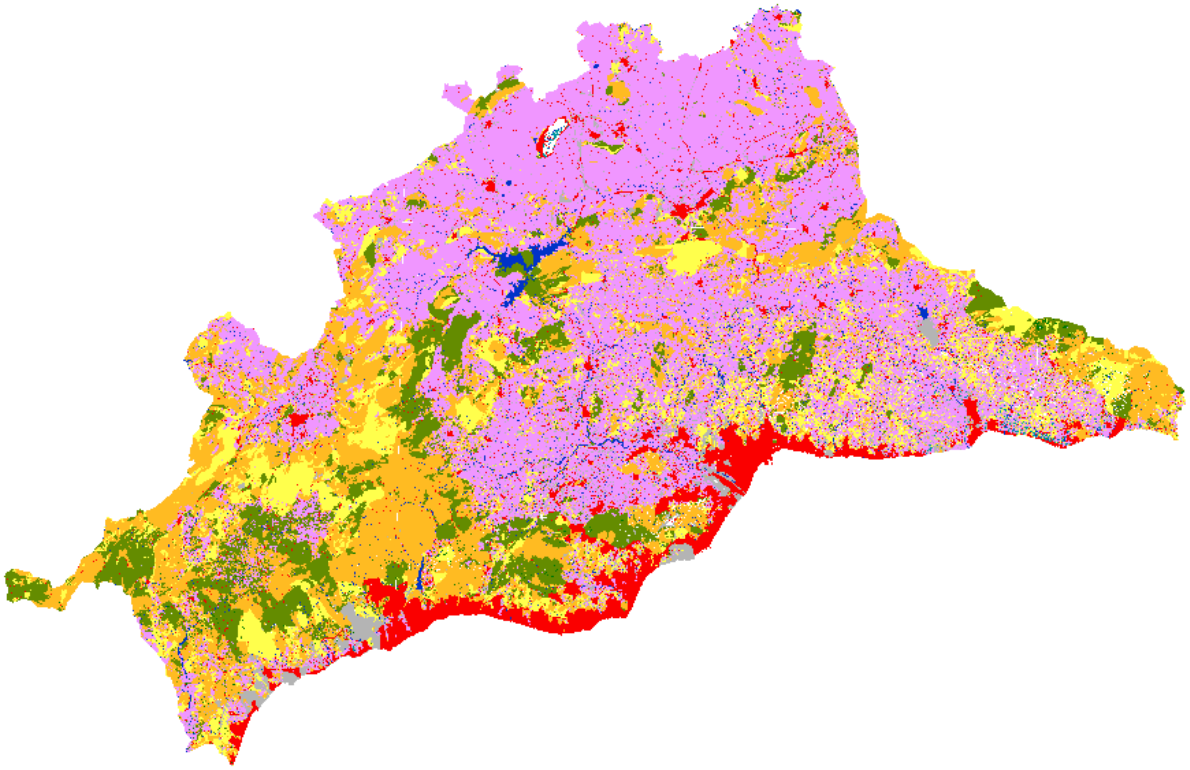
Lo primero de todo es transformar el sistema de coordenadas de *shapefiles* y raster a uno común. El seleccionado es el ETRS89 (Sistema de Referencia Terrestre Europeo) por su uniformidad en toda Europa, precisión y coherencia con los otros sistemas de referencia ya que este se alinea bastante con el sistema geodésico global (WGS (World Geodetic System)). El siguiente paso tras la proyección de los datos a un sistema común sería obtener el área de estudio del raster de la figura 4.2, tal y como vemos en esa figura gran parte del terreno estudiable es agua por lo que un recorte o máscara de la imagen original a una más pequeña nos agilizará todos los procesos, además este paso es fundamental para luego comparar los datos de la banda del raster con las parcelas del *shapefile* puesto que de no solaparse ambos conjunto de datos no podríamos hacer ninguna comparación.

Tras realizar la máscara y comprobar que ambas imágenes se solapan completamente podemos comenzar la creación del nuevo raster con los datos SIGPAC. Este algoritmo se basa

fundamentalmente en sustituir el valor de cada píxel del raster de clasificación con el código de uso de la la respectiva geometría a la que pertenece, es decir, reemplazar el valor antiguo por el código de uso SIGPAC codificado tal y como aparecen en la tabla 2.1. Cabe destacar que el nuevo raster resultante estará formado por una banda a la que se le atribuirán valores entre [0, 30]. En las siguientes figuras vemos el resultado del algoritmo, se han cargado en ambas un código de colores similar para facilitar la comparación, en el caso del raster de valores SIGPAC se han agrupado los atributos de uso similares con un mismo color, por ejemplo todos aquellos atributos que representen cultivos se observarán con el color rosa pese a que representen distintos tipos de cosecha.



(a) El valor de banda del raster es la clasificada por Khaos [1].



(b) El valor de banda de raster con los valores declarados en SIGPAC mapeadas a las clases del Corine.

Figura 4.6: Diferentes clasificaciones de la corteza terrestre de la provincia de málaga.

Los desafíos encontrados se repasarán en el apartado 5.1 y más adelante se analizará en profundidad el algoritmo realizado. No obstante si nos fijamos en ambas figuras vemos como la zona de la Sierra de las Nieves presenta una mayor discordancia de colores. Es importante recordar que la precisión de ambas figuras es distinta y, ese es el motivo por el que la zona urbana (rojo) esta mucho mejor delineada en la figura 4.6b, si la miramos con atención podemos llegar a ver hasta el contorno de los ríos o las carreteras.

4.3.1. Explicación y análisis del algoritmo

Este apartado se centra exclusivamente en el estudio de las funciones que componen el algoritmo principal del proyecto transforma los datos del SIGPAC de formato vectorial a rasterizado, se justificarán algunas de las decisiones tomadas además de un análisis de la complejidad de algunas funciones y bucles con el objetivo de brindar un poco más de contexto a la magnitud del estudio. El algoritmo está compuesto por una serie de funciones que son llamadas entre sí de forma anidada por lo que para facilitar su comprensión vamos a explicar su funcionamiento desde lo más general a lo más específico, en otras palabras el último paso será la función que compruebe el estado del píxel dentro de la geometría para realizar la sustitución. A continuación se expone en pseudocódigo el algoritmo realizado en python, la siguiente Figura 4.7 muestra de manera muy general la naturaleza del algoritmo.

1. Se abre una imagen raster como archivo de entrada y se guardan sus metadatos, se recogen del raster todos los puntos con valores no nulos en la banda y se almacenan en un array de numpy. Todos los valores de píxel no nulos son proyectados mediante una transformada afín a valores (x, y).
2. A continuación se divide la lista de puntos anteriores en t partes iguales para una posterior ejecución de código en paralelo. Este paso será explicado en detalle en 5.1.2.
3. Cada hilo creado en el paso anterior ejecutará por su cuenta la función de sustitución de bandas.
4. Esta función leerá los metadatos del *shapefile* en forma de diccionario y recogerá el código de uso de parcela. Tras recoger el código de uso por cada geometría en el *shapefile* se comprobará que puntos del raster solapan la geometría.
5. La función de comprobación de cada geometría se basa en iterar por cada píxel del raster de entrada comentado en el paso 1 y comprobar uno a uno si estos pertenecen a la

```
Function: Algoritmo transformacion datos vectorial a rasterizado.
    open(classification_raster)
    open(shapefile_data)
    raster_projection(classification_raster)
    mask_shapefiles(shapefile_data)
    start_parallel_execution()
    #Each proccess execute next code
    for geometry in shapefile:
        #For each pixel in raster
        check if pixel in polygon:
            True -> Save pixel in list[pixel[index]]
    #All processes lists are merged together
    process_wait()
    affine_transformation()
    #Next method return the new raster
    band_replacement()
```

Figura 4.7: Algoritmo de transformación de datos vectoriales a rasterizados.

geometría en cuestión. Si este pertenece a la geometría el punto se guardará en una lista junto al resto de puntos pertenecientes, devolviendo una lista de listas con todos las posiciones(índices) de los puntos del raster de entrada.

6. Finalmente con los índices guardados en esa lista de listas solo queda aplicarles de nuevo la transformada afín inversa a la anterior hecha para convertirlos de valores (x, y) a (*row*, *col*) y añadirlos a un diccionario en el que la clave será los índices de los puntos en el raster ya transformados y el valor su código de uso SIGPAC correspondiente.
7. Para concluir se recorre este diccionario y se sustituirán todos los valores de píxel del raster antiguo por los nuevos obteniendo los códigos de uso SIGPAC en una imagen tiff que nos permitirá obtener métricas para la posterior validación del modelo de clasificación.

Tal y como vemos en la explicación este algoritmo pasa por una serie de funciones con aplicaciones diferentes. Para estudiar la magnitud del algoritmo vamos a analizar su complejidad. La complejidad algorítmica es una métrica teórica que describe el comportamiento del algoritmo en función de su tiempo de ejecución y memoria requerida. Para ello vamos a analizar el ritmo de crecimiento, analizando cómo crece el número de instrucciones necesarias para resolver el problema en función del tamaño. El orden de complejidad del algoritmo sería de:

$$O(n^4) = O(s \cdot g \cdot v \cdot p)$$

Donde:

s es el número de archivos shapefile que constituyen el área de estudio.

g es el número de geometrías del archivo shapefile.

v es el número promedio de vértices de las geometrías del shapefile.

p es el número de píxeles del raster de entrada.

El factor más determinante en cuanto al crecimiento del algoritmo es sin duda N o el número de *shapefiles* a analizar puesto que el algoritmo se ejecutará una vez por cada archivo. Cada archivo representa un municipio con tamaños dispares por lo que algoritmo dependerá principalmente del número de geometrías del municipio, el promedio de vértices de las geometrías y el número de píxeles del raster de entrada. Cabe destacar que para simplificar el proceso el raster de entrada ha sido reducido exactamente a la misma zona que el *shapefile* representa por lo que todos los puntos son importantes y deben de ser reemplazados. Si ejemplificáramos el algoritmo con valores numéricos el número de procesos resultante para transformar una comunidad autónoma entera sería de miles de millones de comprobaciones y sustituciones. Es por eso que la optimización del algoritmo fue un desafío a la hora del desarrollo y en el apartado 5.1.2 se comentará sobre ello.

A continuación se comenta brevemente el algoritmo que detecta si el píxel pertenece a la geometría de estudio y las principales librerías de python que nos han ayudado al desarrollo.

Algoritmo de punto en polígono

El algoritmo de punto en polígono (Point-In-Polygon) también conocido como algoritmo del número de cruces o regla de la paridad nos permite detectar aquellos puntos que pertenecen a una geometría. Se ha hecho uso del algoritmo del número de cruces o regla de paridad basado en el teorema de la curva de Jordan [28]. Este teorema explica que: si una curva simple (continua que forma un bucle cerrado) divide el plano en dos regiones una interior y exterior , entonces la curva divide al plano en exactamente dos regiones conexas el interior y el exterior.

De este teorema derivan multitud de aplicaciones en áreas de la topología, teoría de números y geometría computacional. En lo referente a este último encontramos la regla de paridad que nos permite comprobar si un punto pertenece a un polígono, todo se reduce a contar cuantas veces un 'haz de luz' imaginario que parte desde el punto en cualquier dirección fija intersecta los bordes de la geometría este será par o impar en función de la posición del punto. [29]

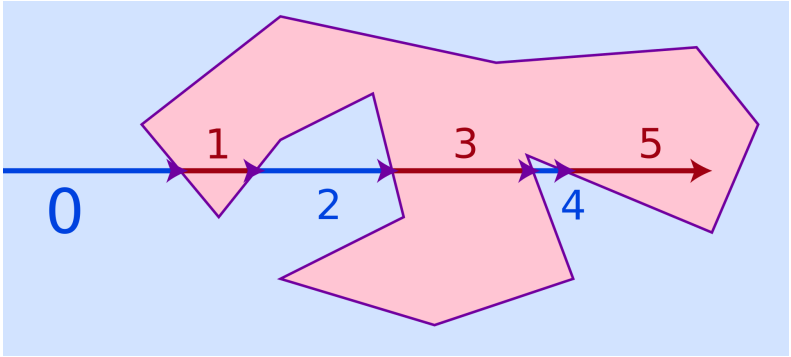


Figura 4.8: Ejemplo del algoritmo de intersección de rayos.

4.4. Proceso de validación de los modelos

Este es el último paso de la implementación y se basa en la comparación de dos raster con diferentes valores de banda. El proceso de comparación de raster crea diferentes imágenes tiff de las que sacaremos datos de porcentaje de acierto de cada clase respecto a los valores de banda. Estos datos serán los que nos permitirán validar el modelo de clasificación. La secuencia seguida para esta nueva creación de rasters se basa en la carga en memoria de ambos archivos para su posterior análisis. El primer raster resultante compara los valores de las bandas y crea un nuevo raster sustituyendo la banda anterior por 0 o 1 en función de si ambos valores coinciden o no. El segundo raster ya es más complejo puesto que se sustituirán los valores de la banda en función de los aciertos y fallos creando una matriz de confusión, para ello se sustituyen los valores de bandas con números entre el intervalo [1, 4] obteniendo una banda con cuatro posibles valores distintos. Las matriz de confusión, por tanto, se ha generado de la siguiente forma:

- Se consideran ***True Positives*** aquellos píxeles cuyo valor de banda sea 1. Esta situación ocurre cuando el valor del píxel en el raster de clasificación y en el raster sigpac coinciden

que el terreno es cultivo.

- Se consideran ***False Positives*** aquellos píxeles cuyo valor de banda sea 2. Esta situación ocurre cuando el valor del píxel en el raster de clasificación indica cultivo pero el raster sigpac no.
- Se consideran ***False Negative*** aquellos píxeles cuyo valor de banda sea 3. Esta situación ocurre cuando el valor del píxel en el raster sigpac indica cultivo y en el raster de clasificación no.
- Se consideran ***True Negatives*** aquellos píxeles cuyo valor de banda sea 4. Esta situación es diferente a las demás puesto que el objetivo principal es la clasificación del cultivo, por tanto, consideraremos como los verdaderos negativos todo los píxeles que no sean cultivo en ninguno de los dos rasters.

En el apartado 5.1.3 se expande lo aquí comentado profundizando en la forma de comparar los rasters de estudio y sus bandas.

4.5. Ejecución del algoritmo

En lo referente a la ejecución del algoritmo durante el desarrollo del código, en las etapas más avanzadas comenzó a hacerse tedioso la llamada a todas las funciones que componen el algoritmo de manera secuencial una a una cambiando los parámetros de entrada de las funciones para cada ejecución. Por ello para simplificar las llamadas a las funciones y evitar errores humanos al indicar los parámetros se tomó la decisión de crear un script que ejecutará el algoritmo tras recibir los parámetros de entrada de forma automática. Por este motivo tras finalizar el algoritmo explicado en los apartados anteriores se implementaron los scripts ***run.py*** y ***launch.sh***.

Finalmente para garantizar que el programa creado puede ejecutarse en cualquier máquina se ha creado un **dockerfile** para ejecutar todo en un contenedor aislado. En el apartado de desafíos 5.1.4 se explicará en detalle y de forma técnica todo lo relacionado con la ejecución del algoritmo y por qué se tomó la decisión de un script de bash y docker.

4.6. Refactorización y pruebas

Con relación a la implementación 4 el último contacto con código en el proyecto fue la refactorización del código y pruebas unitarias con el objetivo de entregar un código limpio, estructurado y bajo los estándares de calidad del software. Durante la refactorización se crearon los *docstrings* de *Python* para documentar el código y facilitar la legibilidad del mismo, también se agruparon algunas funciones y se eliminaron líneas de código redundantes para mejorar la reutilización del código fuente por otras personas.

En cuanto a las pruebas se optó por *tests* unitarios puesto que eran las que mejor se adaptaban a nuestro caso. Las pruebas se desarrollaron utilizando la librería *pytest* y se evaluó el correcto funcionamiento de las funciones principales del algoritmo de manera aislada creando objetos prueba e imágenes manuales cuyos resultados con previamente conocidos.

Compendio del código

A continuación se recopila en una tabla todos los scripts implementados y sus funciones más destacadas:

Script	Nombre	Función
utils.py	reproject_raster	Reproyecta un raster a cualquier sistema de referencia de coordenadas configurado.
utils.py	mask_shp	Recorta una imagen en formato TIFF con las geometrías de un archivo shapefile.
utils.py	merge_tiff_images	Fusiona todas las imágenes TIFF almacenadas en un directorio.
utils.py	is_point_in_polygon	Determina si un píxel dado se encuentra dentro de una geometría/polígono.
utils.py	replace_band_matrix	Reemplaza en la banda del raster el código de uso de la parcela SIGPAC.
utils.py	start_parallel_execution	Ejecuta el proceso con múltiples hilos para mejorar el rendimiento.
validation.py	raster_comparison	Compara los valores de bandas de dos imágenes TIFF en formato *TRUE* y *FALSE*
validation.py	raster_comparison_confmatrix	Compara dos imágenes TIFF y devuelve una matriz de confusión de aciertos y fallos.
validation.py	create_dataframe_and_graphs	Crea y guarda un CSV y gráficas de resultados.
training.py	train_model_land_cover	Entrena un modelo RandomForest.
launch.sh	Script de bash	Script automatizado que recibe los datos de entrada y ejecuta el algoritmo completo.

Cuadro 4.1: Tabla resumen de las principales funciones creadas en el desarrollo del proyecto.

5 Resultados

En este capítulo de resultados nos centramos en el análisis y presentación de todos los casos de uso del trabajo. Por ello se abordan en primera instancia y de manera técnica los desafíos encontrados durante el desarrollo y las soluciones aplicadas para superarlos. Dentro de los casos de uso que se van a explorar entran los mencionados en 1.2 además de algunos extras que, conforme más se indagaba y más conocimiento en la materia obtuve se concretaron durante el desarrollo. En general los resultados han sido muy satisfactorios ya que se ha conseguido implementar todos los objetivos que se planearon. Además se han añadido funciones a los scripts para mejorar la implementación de algunas herramientas, otra característica nueva de la herramienta es la optimización y paralelización de la misma ya que en los estados iniciales del desarrollo no se esperaba que fuese a escalar tanto el proyecto.

En los siguientes puntos se comparan los objetivos esperados en 1.2 con los resultados obtenidos:

- **Punto 1:** Este script mencionado se encuentra en el archivo `utils.py` y en el encontramos todas las funciones indicadas en los sub-apartados. Con este script se han procesado las imágenes proporcionadas por Khaos [1] y se han creado las imágenes .tiff con datos sigpac de todos los municipios por separado de todas las provincias de Andalucía y parte de Castilla y León.
- **Punto 2:** Guardado en el archivo `validation.py` encontramos las funciones de clasificación y comparación de imágenes .tiff mencionadas. Se han generado rasters aciertos/-fallos y matrices de confusión de toda Andalucía y parte de Castilla y León.
- **Punto 3:** El código del proyecto está optimizado y refactorizado para conseguir la mayor legibilidad posible. También se han realizado pruebas unitarias. Todo el código se encuentra guardado en un repositorio de GitHub [20].

- **Punto 4:** Se han realizado los scripts `run.py` y `launch.sh` para la ejecución automática de la aplicación al completo. También se ha desarrollado un Dockerfile para la ejecución de la aplicación en un contenedor aislado.

5.1. Desafíos y soluciones

5.1.1. Escalabilidad y máquinas virtuales

El primer problema que surgió fue comprender la magnitud real del problema y la forma de abordarlo. Puesto que se comenzó trabajando en local en un portátil con 8GB de RAM (Random Access Memory) y 8 núcleos que en principio debería de ser más que suficiente para el desarrollo, no obstante en las primeras etapas del proyecto cuando aún no estaba optimizado el código y se comenzaron a analizar los primeros *shapefiles* de municipios en lugar de *shapefiles* de prueba el tiempo de ejecución para un único municipio se disparó. El tiempo llego a tardar horas para los municipios más grandes. Desde el grupo de investigación *Khaos* [1], con quienes he desarrollado el proyecto, me ofrecieron una máquina virtual de desarrollo que mejoraba las características de mi portátil y aumentaba el rendimiento del algoritmo. Está fue una solución parcial al problema puesto que el masivo tiempo de ejecución se originaba no solo por la gran cantidad de píxeles de análisis sino también por la falta de optimización del código, este último motivo se explica en detalle en el apartado 5.1.2.

Tras mejorar y optimizar el código al máximo posible, se paralelizó el código ejecutándose en múltiples procesos por lo que una máquina con más núcleos reduciría el tiempo de ejecución muchísimo. Tras implementar estas mejoras obtuve acceso a una máquina virtual orientada a la ejecución de código con 128GB de RAM y 64 núcleos con la cual termine todas las ejecuciones necesarias en minutos.

El acceso a estas máquinas ya comentadas fue mediante el protocolo de red **ssh** (Secure Shell) que permite mediante terminal acceder, gestionar y transferir datos de forma segura a través de internet.

5.1.2. Optimización con Numpy, Numba y Paralelismo

Esta sección está muy relacionada con la anterior y aquí se van a comentar todas las estrategias tomadas para la optimización del código y la mejora del rendimiento. Al final del apartado se mostrará gráficamente los distintos tiempos de ejecución de más a menos así como las distintas estrategias para visualizar mejor los resultados.

Numpy y Numba

El uso de los arrays de numpy mejoró el rendimiento del código en varios milisegundos por operación lo que a la larga redujo considerablemente el tiempo de ejecución. Además hemos sacado partido a la librería a la hora de trabajar con las imágenes .tiff y los mapas de bits. Convirtiendo este tipo de datos en ndarrays podemos acceder a valores concretos de forma eficiente. Por tanto para comparar las imágenes se han usado arrays de numpy de 3 dimensiones para las cuales las dos primeras hacen referencia a las posiciones en x e y, y la tercera para el valor de la banda en ese píxel concreto. La siguiente figura 5.1 muestra un ejemplo de como sería un array de 3 dimensiones indexado.

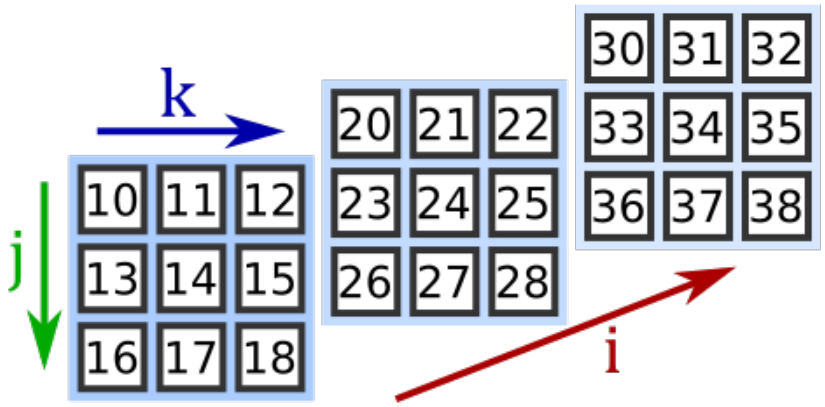


Figura 5.1: Diagrama de un array de 3 dimensiones. Fuente: [30]

Respecto a la librería de numba se ha indicado en aquellas funciones con mayor carga de análisis numérico su respectivo decorador jit para optimizar su ejecución.

Paralelismo

Para comentar el paralelismo efectuado primero hay que comprender como funciona *Python* y el *GIL* (Global Interpreter Lock). Este çandado"permite que solo un hilo pueda contener el control del interprete de *Python*. La alternativa para tratar con este bloqueo es el uso de multiprocesamiento en lugar de *multithreading*, en este caso se generan subprocesos con su respectivo interprete de *Python* y espacio en memoria. Este es el motivo que se haya usado para la optimización del código el módulo de **multiprocesamiento**.

En lo referente al proyecto se plantearon diversas formas de paralelizar el código en subprocesos y mejorar el rendimiento. La opción implementada fue seleccionada puesto que el problema de nuestro algoritmo es la gran cantidad de comprobaciones por píxel que hay que realizar y no por su complejidad en las operaciones. De esta forma para solucionar este problema se ha aplicado la siguiente estrategia: Obtenemos el número de cores de nuestra máquina y dividimos la lista preprocesada de las coordenadas de los píxeles en tantas partes como cores encontremos en la máquina. Aquí ya vemos como un mayor número de cores reduce la longitud de la lista en menos píxeles y por tanto en menos operaciones por núcleo. Tras esta división generamos un diccionario en el que guardaremos los resultados de la función que llamemos. Esta función será invocada cada vez que queramos paralelizar, es decir, una vez por cada core, se inician todos los procesos en simultáneo y se esperarán a que todos terminen para continuar. Finalmente reemplazamos los valores guardados en el diccionario común previamente comentado por los nuevos valores de bandas.

Regresando a lo comentado en el apartado 5.1.1, gracias a todos estos cambios en el código y la máquina virtual de desarrollo de 128 cores se consiguió reducir el tiempo de ejecución de una provincia entera a unos 20 minutos, a diferencia de las horas que se tardaba al comienzo para un único municipio.

5.1.3. Creación de JSONs para vincular las clases

Uno de los grandes problemas que surgieron fue la conexión entre las clases de las imágenes sigpac y las de clasificación. Recordando lo ya mencionado sigpac clasifica las parcelas en códigos de uso que representan el tipo de terreno que es ocupado. En total sigpac utiliza 30 códigos diferentes para esta tarea frente a los 9 utilizados normalmente en la clasificación

de la cobertura terrestre. De esta forma tras la creación de las imágenes *.tiff* con el valor de banda de sigpac se encontraron problemas a la hora de comparar ambas imágenes.

Como solución a este problema se ha optado por la creación de archivos **json** Javas Script Object Notation). Un archivo json es un formato de texto sencillo para el intercambio de datos que se utiliza como alternativa a XML (Extensible Markup Language). Haciendo uso de esta herramienta se vincularon los valores de ambas imágenes según el tipo de terreno, es decir, las características del código de uso de la banda sigpac se han agrupado por los valores de bandas del clasificador. Tal y como vemos en la figura 5.2 cada clave del sigpac hace referencia una lista de valores del clasificador donde podemos encontrar uno o más valores de bandas vinculados. Si verbalizamos lo que está escrito en el json la clave `"1"` representa tal y como vemos en 2.1 a las corrientes y superficies de agua por lo que los valores del clasificador que vinculamos son el `4` y `5` que respectivamente tal y como vemos en 4.2.1 significan agua y zonas húmedas. De forma análoga se pueden leer el resto de valores que van desde la clase `1` hasta la `30` pese a que este cortado en la figura.

```
{
    "style_sheet": {
        "SIGPAC_code": {
            "1": [
                4,
                5
            ],
            "2": [
                1,
                9
            ],
            "3": [
                6,
                7,
                8
            ],

                ...
        }
    }
}
```

Figura 5.2: Ejemplo de como se vería uno de los JSON creados.

Durante el proceso de validación se fueron cambiando algunos valores del json, puesto que nuestro objetivo en todo momento es la correcta validación de los cultivos y no tanto las zonas de agua, urbanas o terrizos. No obstante los valores de píxeles de bosque cerrado y abierto del clasificador fueron los más difíciles de vincular puesto que los datos sigpac diferencian un olivo de un árbol frutal o viñedos mientras mientras que el clasificador directamente clasifica estas zonas según la densidad de la copa de los árboles. Finalmente para estos casos se vincularon todos los valores sigpac cuya descripción fuese arbórea con cultivo, bosque abierto y bosque cerrado (6, 7 y 8). Estos datos ya fueron útiles para el grupo de investigación puesto que se pudo inferir que el clasificador no es todo lo preciso que podría ser a la hora de distinguir árboles frutales respecto a árboles comunes. Por lo que ajustando el modelo en sus hiper-parámetros y comparando resultados podríamos mejorar la precisión en este caso. Esto último se comentará en detalle en los casos de uso.

Lo anterior comentado es lo que hemos usado para realizar los rasters de comparación. Estos rasters se han hecho exclusivamente comparando los cultivos puesto que esta es la clase de estudio, sin embargo podríamos cambiarlo en cualquier momento por otro valor de banda en caso de cambiar la línea de investigación al urbanismo por ejemplo. En lo referente al raster de validación con los valores de banda de una matriz de confusión se consideraron todos los valores de bandas que sean 0 como dato perdido del clasificador y no se tendrán en cuenta a la hora de la validación explicada. El motivo de esto es que el modelo de clasificación elimina todos los píxeles que no rebasen un porcentaje de calidad estimado, es decir, aquellos píxeles o zonas que no se han podido medir bien sus índices de reflectancias por culpa de motivos meteorológicos (nubes principalmente) o por zonas restringidas de uso militar.

5.1.4. Docker

Finalmente este apartado fue el último en realizarse y se creó con el objetivo de evitar cualquier problema a la hora de ejecutar la aplicación. Para ello se ha diseñado un ***Dockerfile*** mediante *Docker*. La utilización de Docker ha simplificado el proceso de despliegue de la aplicación al encapsular todas las dependencias y configuraciones en un contenedor, lo que asegura una mayor consistencia en su ejecución, independientemente del entorno en el que se despliegue.

5.2. Casos de uso

5.2.1. Métricas de validación

A continuación se va a presentar como se ha validado el modelo de clasificación mediante los datos sigpac e imágenes .tiff, para ello continuaremos por donde lo hemos dejado en el apartado 4.4.

Imágenes Aciertos/Fallos

Comenzamos generando las imágenes de aciertos/fallos cuyos valores de píxel, como ya se ha explicado oscilan entre 0 y 2 e indican datos perdidos tal y como aparece en la Figura 5.4 existen muchos píxeles que no coinciden, es decir que tienen un valor diferente en ambos rasters. Observando esta figura comparativa y la Figura 5.3 podemos de un vistazo detectar aproximadamente como de bueno fue el resultado.

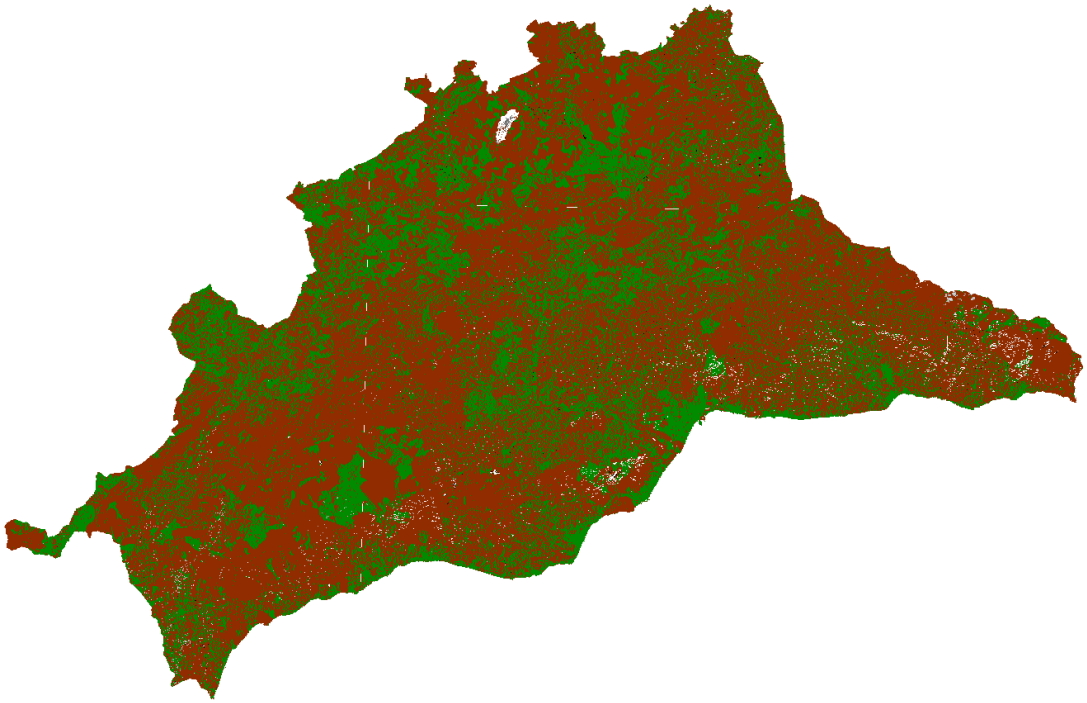
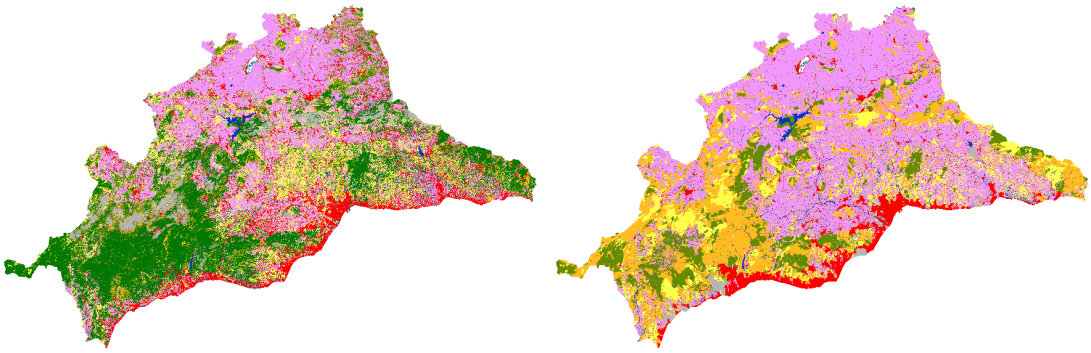


Figura 5.3: Raster resultante True/False.

Para este raster comparamos los valores de banda estrictamente píxel a píxel. Otorgando el color verde solo a aquellos píxeles cuyo valor sea idéntico en sigpac y el clasificador. Por ese

problema encontramos tantos píxeles mal clasificados. Este es el motivo por el que se acabo realizando la matriz de confusión para sacar información más detallada.



(a) Clasificador. (b) SIGPAC.

Figura 5.4: Rasters comparados.

El raster de 5.3 es el resultado de la comparación de los rasters de **??**. Observamos como la zona oeste de Málaga hay muchos fallos en el Parque Nacional de la Sierra de las Nieves ya que el clasificador detecta que esa zona es exclusivamente de bosque y SIGPAC lo considera entre otros; bosque, pastizales y herbáceos. Sin embargo realmente esos píxeles al no ser cultivo en ninguno de los dos rasters son prescindibles y podemos descartarlos.

Imágenes con Matriz de Confusión

De forma análoga al apartado anterior se han generado las imágenes cuyos valores de bandas representan una matriz de confusión, tal y como aparece en la figura 5.5:

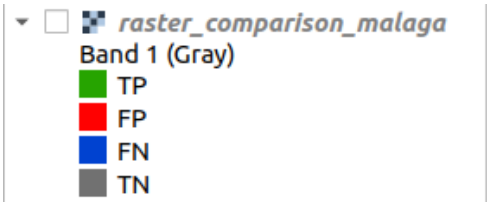


Figura 5.5: Leyenda de los colores de las imágenes.

Contando el número de píxeles de cada valor de banda obtendríamos los valores de la matriz de confusión y, con ella podríamos calcular las métricas estadísticas más utilizadas en la literatura como el *accuracy* o la *precission*. No obstante muchas de estas métricas no nos proporcionarían información útil ya que en nuestro caso, por ejemplo con el *accuracy* no buscamos medir el porcentaje de píxeles correctos respecto al total clasificado ya que muchos

de los píxeles representan terrenos urbanísticos o pastizales que no nos interesan. Ya que el clasificador está entrenado con píxeles de cultivos esta es la clase en la que queremos enforcar las métricas de estudio y analizar exclusivamente la superficie de cultivos.

Por este motivo la métrica a estudiar es el *recall* o *sensitivity*. La sensibilidad mide la capacidad de un modelo para identificar correctamente todos los ejemplos de la clase positiva en un conjunto de datos. Este se calcula como la proporción de valores reales que el modelo predice correctamente entre el total de ejemplos positivos reales tal y como vemos en la siguiente fórmula:

$$\text{Sensibilidad} = \frac{\text{Verdaderos Positivos (TP)}}{\text{Verdaderos Positivos (TP)} + \text{Falsos Negativos (FN)}} \tag{5.1}$$

Aplicando esta ecuación a cada tipo de cultivo de la clase sigpac y comparando las imágenes con valores de banda sigpac y del clasificador podemos obtener el porcentaje de aciertos del clasificador respecto a cada clase. Este cálculo se ha aplicado para obtener varias tablas con las que validar el modelo que se muestran más adelante como en las Figuras 5.2 y 5.1.

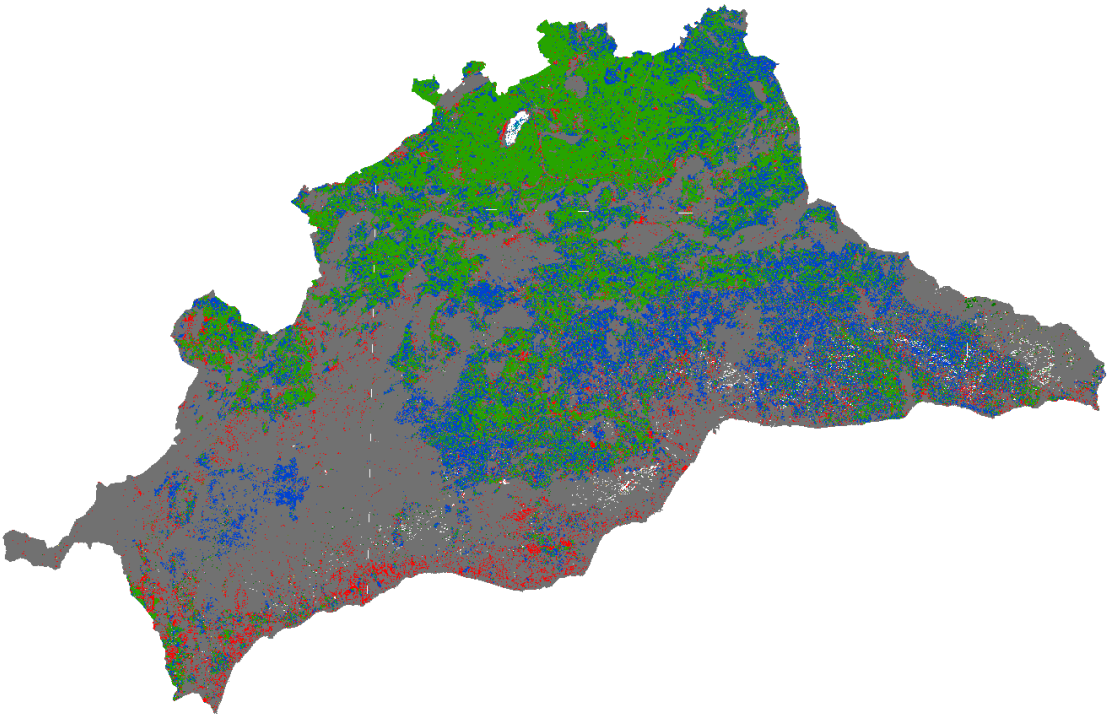
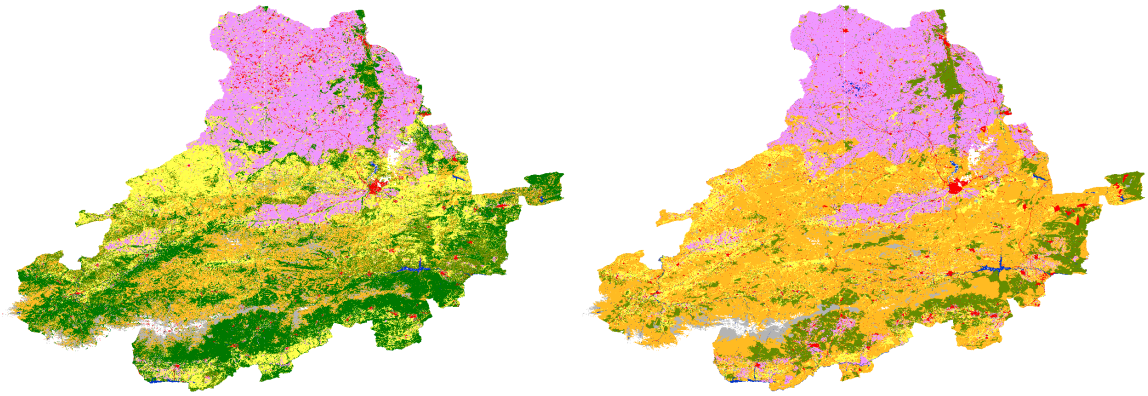


Figura 5.6: Málaga con los valores de píxel de la matriz de confusión.

A continuación se muestra clasificada la provincia de Ávila de la comunidad autónoma de Castilla y León. En la Figura 5.7 observamos rásters tanto del clasificador como de SIGPAC.

De un primer vistazo podemos observar como la zona Norte es la de cultivo y bastante común en ambos rásters. El resto de píxeles son bastante dispares y aquí de nuevo, identificamos como las zonas de bosques y herbáceas son las que peor clasifica el modelo por la dificultad de diferenciar desde una foto multi-espectral aérea la copa de un árbol con un arbusto. Tras aplicar nuestro algoritmo obtenemos la Figura 5.8. Tal y como esperábamos los true positives se encuentran en casi todas las zonas de cultivo por lo que en ese aspecto el resultado puede ser bueno. Todos los píxeles grises que indican que no hay cultivo en ningún caso son prescindibles por lo que el resultado es positivo.



(a) Clasificador. (b) SIGPAC.

Figura 5.7: Rasters de la provincia de Ávila comparados.

Observamos el siguiente porcentaje de acierto en la provincia de Ávila en las diferentes clases. En la Tabla 5.1 vemos como los resultados son muy buenos en general ya que tanto tierra arable como Cítricos-Viñedo poseen un porcentaje de acierto de 86.2 % y 100 % respectivamente. Estas dos clases constituyen más del 80 % del total de píxeles de cultivo. En Castilla y León existen muy pocas plantaciones de olivares cuya clase es la que el modelo peor clasifica por lo que en Ávila soslayamos esa penalización del modelo y obtenemos en general mejores resultados que en otras provincias.

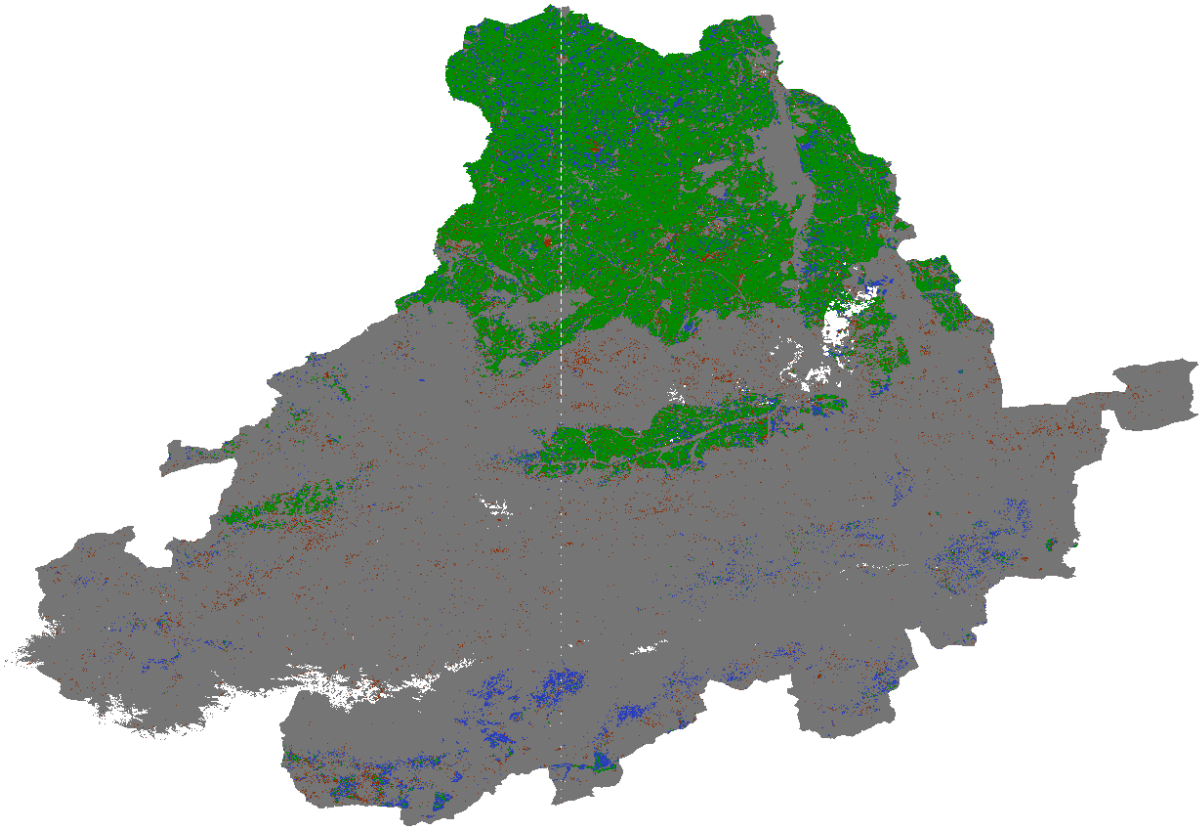


Figura 5.8: Ávila con los valores de píxel de la matriz de confusión.

Clase	Num Pixels	Aciertos	Fallos	**Porcentaje de Acierto**
Cítricos Frutal	1609	0	1609	0.0
Cítricos	48	0	48	0.0
Cítricos-Frutal de cascara	255	0	255	0.0
Cítricos-Viñedo	238909	0	238909	100.0
Frutal de Cascara-Frutal	14063	0	14063	0.0
Frutal de Cascara-Olivar	33	0	33	0.0
Frutal de Cascara	17954	12796	5158	71.271
Frutal de Cascara-Viñedo	290	65	225	22.414
Frutal	215501	28868	186633	13.396
Invernadero y cultivos bajo plástico	2332	576	1756	24.7
Olivar-Cítricos	0	0	0	0.0
Olivar-Frutal	302	161	141	53.311
Olivar	309170	23296	285874	7.535
Tierra Arable	15477547	13350683	2126864	86.258
Huerta	6374	1395	4979	21.886
Frutal-Viñedo	7528	838	6690	11.132
Viñedo	296516	73306	223210	24.722
Olivar-Viñedo	22871	2262	20609	9.89

Cuadro 5.1: Porcentaje de Acierto de Ávila respecto a cada tipo de cultivo.

La siguiente tabla 5.2 recopila el número de píxeles, los aciertos, los fallos y el porcentaje de aciertos de cada una de las ocho provincias de la comunidad autónoma de Andalucía.

Observando la tabla de resultados podemos ver algunas clases con un muy buen porcentaje de aciertos y otras con uno no tan bueno. Los mejores resultados los encontramos en los Cítricos, los Cítricos-Viñedo y la Tierra arable marcados en verde en 5.2. Aquellos no tan buenos como los Cítricos-Frutal, Cítricos-Frutal de cáscara, el Frutal de Cáscara-Frutal y los Invernaderos están marcados en rojo. Gracias al contenido de esta tabla podemos obtener información muy valiosa con la que validar el clasificador y optimizar sus hiper-parámetros.

Clase	Num Pixeles	Aciertos	Fallos	**Porcentaje de Acierto**
Citricos Frutal	1069300	89403	979897	8.36
Citricos	8041756	5856071	2185685	72.82
Citricos-Frutal de cascara	8699	1545	7154	17.76
Citricos-Viñedo	2952034	2950975	1059	99.96
Frutal de Cascara-Frutal	838815	7312	831503	0.87
Frutal de Cascara-Olivar	279257	82857	196400	29.67
Frutal de Cascara	20177469	9080721	11096748	45.0
Frutal de Cascara-Viñedo	12436	3806	8630	30.6
Frutal	14711965	4774161	9937804	32.45
Imvernadero y cultivos bajo plastico	5015050	771053	4243997	15.37
Olivar-Citricos	110703	36820	73883	33.26
Olivar-Frutal	93675	30066	63609	32.1
Olivar	162620685	72449454	90171231	44.55
Tierra Arable	161368050	105376719	55991331	65.3
Huerta	1238266	570850	667416	46.1
Frutal-Viñedo	98892	35013	63879	35.41
Viñedo	2373903	1122568	1251335	47.29
Olivar-Viñedo	147654	55533	92121	37.61

Cuadro 5.2: Porcentaje de Acierto de Andalucía respecto a cada tipo de cultivo.

5.2.2. Validación del modelo

La siguiente Figura 5.9 muestra la matriz de confusión del modelo desarrollado en Khaos para la clasificación de las nueve clases de terreno de la comunidad autónoma de Andalucía. Para validar el modelo vamos a hacer uso de la tabla de métricas 5.2 de Andalucía que se ha comentado en la sección anterior.

La validación del modelo de clasificación de Khaos y el desarrollado en este proyecto presentan similitudes pero con enfoques diferentes. La validación de Khaos se ha realizado con puntos de Andalucía escogidos por humanos y colocados en sitios sencillos de predecir como en el centro de un bosque o en zonas completamente urbanas. Por otro lado gracias a los datos del SIGPAC conseguimos validación perfecta de toda la cobertura del suelo en ella encontramos píxeles perfectos como los usados en el modelo pero también píxeles que contienen más de una clase como por ejemplos los parques en las zonas urbanas. De forma que la validación hecha en este proyecto con los datos SIGPAC es más fiable.

El grupo Khaos como vemos en la Figura 5.9 (nos fijamos solo en la fila y columna de cultivo) obtiene un 76 % de acierto en la clasificación de los píxeles de cultivo, sin embargo nuestro algoritmo obtiene porcentajes muy dispares en cuanto al acierto y fallos de las clases tal y como vemos en 5.2. Esto se debe a que las clases de SIGPAC para cultivo se dividen en otras 18 clases entre las que encontramos cultivos muy distintos a nivel físico y, por tanto a nivel de reflectancia. De forma que aquellas clases como los olivos o árboles frutales sean difícilmente diferenciables con un píxel de un bosque. No obstante comparando ambas validaciones observamos como el modelo de clasificación realiza un muy buen trabajo en la comunidad autónoma de Andalucía.

Finalmente por dar valor al trabajo realizado en este proyecto comentar lo que supone este estudio como ejemplo de validación de cobertura terrestre. Al proporcionar una validación con datos tan fiables y rigurosos como los del SIGPAC mostramos un enfoque automático, diferente a lo existente que permite validar este tipo de modelos de clasificación evitando el trabajo más 'difícil' de la elección de los puntos de entrenamiento y test uno a uno.

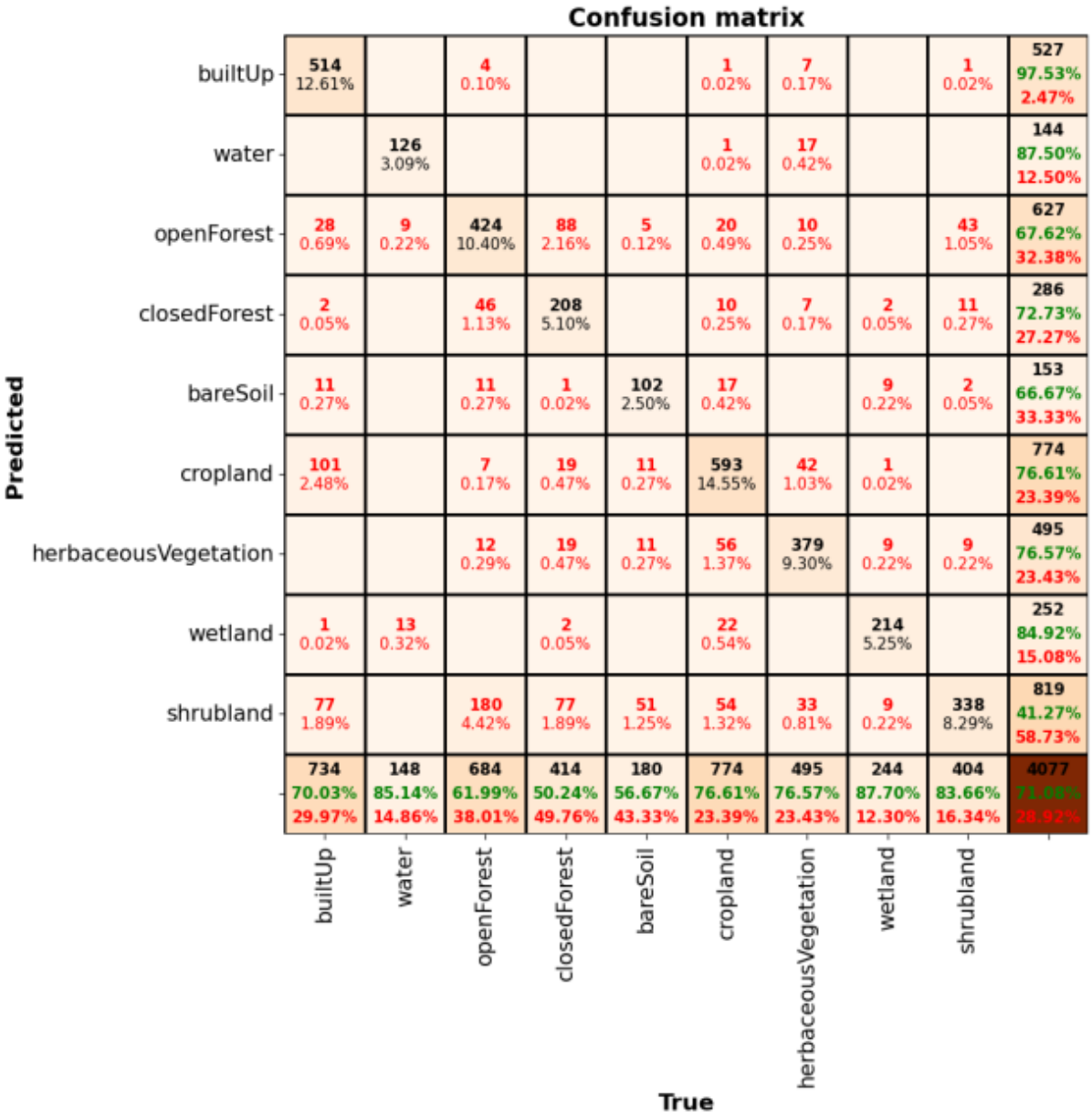


Figura 5.9: Resultado del clasificador de Khaos para la comunidad autónoma de Andalucía.

5.2.3. Ajuste de los hiper-parámetros

Gracias a estas métricas podemos ahora ajustar los hiper-parámetros del RandomForest e intentar obtener mejores resultados. Que más tarde se analizarán de nuevo de la forma ya explicada. Para este apartado se ha hecho uso del flujo de trabajo de cobertura terrestre del laboratorio de investigación [1]. El funcionamiento en detalle de este *workflow* se se puede revisar en el siguiente artículo [2].

En nuestro caso queremos estudiar una provincia concreta por lo que llamaremos al flujo de trabajo indicando solo las zonas de interés a las que le pasaremos el clasificador con los datos entrenados. El resultado obtenido pasará por el proceso de validación de este proyecto e indicará la calidad de predicción de cada clase respecto a los datos sigpac. De esta forma se ha aplicado una validación cruzada y una búsqueda en cuadrícula para encontrar la combinación de parámetros que mejores resultados nos proporcione. Cabe destacar que todos los raster anteriores se han obtenido con los parámetros por defecto del RandomForest de la librería de *scikit-learn*. Entre los parámetros más importantes dada la semántica del proyecto destacan los siguientes atributos con sus respectivos valores por defecto:

- n-estimatores : 100 (Número de árboles en el bosque)
- criterion : gini (Parámetro que mide la calidad de la división)
- max-depth : None (Profundidad máxima del árbol)
- min_samples_split : 2 (Número mínimo de atributos para dividir un nodo interno)
- min_samples_leaf : 1 (Número mínimo de atributos para poder ser un nodo hoja)
- max_features : sqrt (Número de funciones que comprobar para realizar una división)

Leyendo la documentación oficial [31] y los parámetros que ofrece el modelo se ha optado por realizar una búsqueda en cuadrícula de los parámetros del modelo, en otras palabras se analizarán todas las combinaciones posibles de parámetros de la siguiente cuadrícula para obtener y analizar que parámetros mejoran los resultados obtenidos anteriormente. De esta forma podemos ver si existe alguna correlación entre los parámetros del modelo y alguna clase concreta de nuestros datos.

Búsqueda en cuadrícula utilizada:

```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2'],
    'bootstrap': [True, False],
    'criterion': ['gini', 'entropy'],
}
```

Tras recorrer la malla anterior en un bucle y aplicar las distintas combinaciones de parámetros al modelo de clasificación se han validado los resultados del entrenamiento con el algoritmo creado en este proyecto. Por cada iteración se obtienen los rasters de métricas correspondientes y se analizan los porcentajes de acierto. En el que caso que el porcentaje de acierto promedio sea muy bajo o peor que el modelo analizado anteriormente se descartará esa selección de parámetros. Obteniendo finalmente la combinación de parámetros que mejor se ajuste al proyecto.

6 Conclusión

6.1. Conclusiones y resumen de los resultados

Como conclusión podemos afirmar que el proyecto ha sido un éxito. Se han conseguido desarrollar todos los objetivos que se plantearon en el anteproyecto así como nuevos objetivos que han surgido durante el desarrollo. Todo el código queda alojado en GitHub en un repositorio abierto para su uso [20]. El resultado final ofrece tal y como se buscaba un algoritmo de validación de rasters así como múltiples funciones para el manejo de datos geoespaciales. El proyecto se encuentra automatizado de forma que cualquiera siguiendo los pasos indicados en el repositorio pueda ejecutar el algoritmo con sus propios datos. Respecto a los resultados se han obtenido unas tablas que resumen lo bien que se ha clasificado una clase además de nuevas imágenes de validación con su respectiva leyenda de colores, que proporciona una visión clara del rendimiento del clasificador. Esta información no solo valida la efectividad del algoritmo, sino que también facilita la interpretación de los resultados, lo que es esencial en aplicaciones prácticas. Adicionalmente, la implementación de técnicas como el cross-validation ha contribuido a garantizar la fiabilidad de los resultados, ofreciendo una evaluación más robusta del rendimiento del algoritmo y el ajuste de sus hiper-parámetros.

Durante el desarrollo del proyecto se han superado multitud de desafíos respecto al manejo de datos GIS, el funcionamiento del algoritmo y sus funcionalidades. Gracias a este proyecto se ha conseguido aprender buenas prácticas de programación y de integración de datos y código.

Cabe destacar que pese a haber invertido mucho tiempo en la optimización del código el algoritmo sigue siendo un poco lento por la naturaleza de los datos. Por lo que como trabajo futuro se podría tratar de implementar una versión del algoritmo implementando CUDA o GPUs.

En última instancia, en este Trabajo de Fin de Grado se ha buscado exponer y demostrar todo lo aprendido durante el trascurso de mis estudios universitarios en la Universidad de Málaga.

6.2. Futuras líneas de investigación

En cuanto a la investigación este campo ofrece multitud de ramas de estudio muy interesantes. Respecto al *land cover* aún en la literatura no hay ningún método eficaz que permita validar modelos de clasificación. Este proyecto ofrece una forma alternativa para la validación de modelos pero está muy acotado a la región de España y depende de la correcta declaración anual de los datos sigpac. Otra línea sería el desarrollo de métodos robustos de aprendizaje computacional para la cobertura terrestre aplicando algoritmos más complejos que el *random forest* como las redes neuronales o el aprendizaje profundo de forma que se consiga encontrar patrones complejos y representaciones más profundas de los datos y así mejorar la clasificación del modelo entre clases similares.

6.2.1. Fraudes declaración parcelas SIGPAC

Como trabajo futuro de la aplicación, está podría usarse por parte de las autoridades competentes como una herramienta para la detección de fraudes en la declaración de las parcelas del SIGPAC. En Europa existen la Política Agrícola Común (PAC) que se dirige al sector agrario y al medio rural y ofrece apoyo económico a agricultores gestionando el sistema agroalimentario al completo. En otras palabras los agricultores reciben ayudas económicas o bien una reducción de impuestos en función del tipo de cultivo que tengan y la zona donde se cultive, de esta forma se pueden dar casos de fraudes declarando al gobierno y, por tanto a sigpac un tipo de cultivo diferente.

Para solucionar este problema podemos analizar los píxeles clasificados como ***False Positives*** (color rojo en los rasters). Estos píxeles son aquellos que el clasificador ha detectado como cultivo y sigpac no. Para explicar mejor este caso volvemos a la figura 5.6 de Málaga. Observamos algunos puntos rojos sobretodo por la zona Sur-Oeste. Esa zona representa principalmente la zona urbana de Málaga y, muchos de esos píxeles son zonas verdes o parques, que el clasificador al analizar la reflectancia los detecta como cultivo y sigpac los define como zona urbana. No obstante si analizamos las zonas más aisladas y sus proximidades podríamos

llegar a encontrar casos de fraude o plantaciones ilegales.

Bibliografía

[1] University of Málaga. Khaos research group. https://khaos.uma.es/.

[2] Antonio Manuel Burgueño, José F. Aldana-Martín, María Vázquez-Pendón, Cristóbal Barba-González, Yaiza Jiménez Gómez, Virginia García Millán, and Ismael Navas-Delgado. Scalable approach for high-resolution land cover: a case study in the mediterranean basin. *Journal of Big Data*, 10(1):91, June 2 2023.

[3] Proyecto corine de la cobertura terrestre. https://land.copernicus.eu/en/products/corine-land-cover.

[4] Elzbieta Bielecka and Agnieszka Jenerowicz. Intellectual structure of corine land cover research applications in web of science: A europe-wide review. *Remote Sensing*, 11(17), 2019.

[5] R. Brennan and T. L. Webster. Object-oriented land cover classification of lidar-derived surfaces. *Canadian Journal of Remote Sensing*, 32:162–172, 2006.

[6] Copernicus - european space agency (esa). https://www.esa.int/Applications/Observing_the_Earth/Copernicus.

[7] ASTER (advanced spaceborne thermal emission and reflection radiometer). https://www.earthdata.nasa.gov/sensors/aster. Accessed: February 19, 2024.

[8] Sentinel-2 spatial resolution. https://sentinel.esa.int/web/sentinel/user-guides/sentinel-2-msi/resolutions/spatial.

[9] Sentinel-2 vegetation index. https://custom-scripts.sentinel-hub.com/sentinel-2/ndvi/.

[10] Curve comparison reflectancy. https://midopt.com/tools/curve-compare/.

[11] QGIS Association. Qgis 3.28. geographic information system developers manual. https://docs.qgis.org/3.28/en/docs/developers_guide/index.html, n.d.

[12] Rgb satelite image bands. https://pages.cms.hu-berlin.de/EOL/geo_rs/.

[13] V Janssen. Understanding coordinate reference systems, datums and transformations. 1 2009.

[14] J. González-Matesanz, A. Dalda, and J.A. Malpica. A range of ed50-etrs89 datum transformation models tested on the spanish geodetic network. *Survey Review*, 38(302):654–667, 2006.

[15] Richard B Langley. The utm grid system. *GPS world*, 9(2):46–50, 1998.

[16] Usos sigpac. https://sigpacweb.es/usos-sigpac/. Accedido el 2 de octubre de 2023.

[17] Asamaporn Sitthi, Masahiko Nagai, Matthew Dailey, and Sarawut Ninsawat. Exploring land use and land cover of geotagged social-sensing images using naive bayes classifier. *Sustainability*, 8(9):921, 2016.

[18] Manuel Campos-Taberner, Francisco Javier García-Haro, Beatriz Martínez, Sergio Sánchez-Ruíz, and María Amparo Gilabert. A copernicus sentinel-1 and sentinel-2 classification framework for the 2020+ european common agricultural policy: A case study in valència (spain). *Agronomy*, 9(9):556, 2019.

[19] Linda Rising and N.S. Janoff. The scrum software development process for small teams. *Software, IEEE*, 17:26 – 32, 08 2000.

[20] Repositorio personal con el código fuente. https://github.com/jesusaldanamartin/satelite_images_sigpac.

[21] Documentación de docker. https://docs.docker.com/. Accedido el 30 de octubre de 2023.

[22] Documentación fiona. https://fiona.readthedocs.io/en/stable/. Accedido el 2 de octubre de 2023.

[23] Documentación rasterio. https://rasterio.readthedocs.io/en/latest/index.html. Accedido el 2 de octubre de 2023.

[24] Documentación de matplotlib. https://matplotlib.org/3.8.0/index.html.

[25] Documentación de numpy. https://numpy.org/devdocs/reference/index.html#reference.

[26] Documentación de numba. https://numba.readthedocs.io/en/stable/user/5minguide.html.

[27] Junta de Andalucía. Sigpac - visor de información geográfica. https://www.juntadeandalucia.es/organismos/

agriculturapescaaguaydesarrollorural/servicios/sigpac/visor/paginas/sigpac-descarga-informacion-geografica-shapes-provincias.html. Accessed on 2023-10-03.

[28] Thomas C. Hales. The jordan curve theorem, formally and informally. *The American Mathematical Monthly*, 114(10):882–894, 2007.

[29] Chong-Wei Huang and Tian-Yuan Shih. On the complexity of point-in-polygon algorithms. *Computers & Geosciences*, 23(1):109–118, 1997.

[30] Diagrama de ínidices de 3 dimensiones. https://www.pythoninformer.com/python-libraries/numpy/index-and-slice/.

[31] Documentacion scikit-learn python. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

Apéndice A
Manual de uso

Este apéndice que viene a continuación, tiene como objetivo proporcionar a los usuarios una guía detallada de instalación y uso de la herramienta que se ha desarrollado en el marco de este Trabajo de Fin de Grado. En la sección A.1 se comentan los requisitos mínimos para el uso del software y en la sección A.2 se comentan uno a uno los pasos a seguir para probar la herramienta al completo.

A.1. Prerrequisitos

- Versión 3.11 de Python o superior.
- Versión 24.0.6 de Docker o superior (DEMO).
- Versión 2.42 de Git o superior (Clonar repositorio).

De no superar la versión de *Python* indicada es posible que algunas librería utilizadas no funcionen y provoquen problemas de dependencias. Se aconseja el uso de una máquina multicore para la ejecución del algoritmo en paralelo para evitar largos tiempos de ejecución.

A.2. Manual de uso

Para entender como ejecutar la aplicación siga los pasos comentados a continuación. En caso de querer ejecutar la demo y ver los resultados que ofrece el proyecto continúe en A.3.

A.2.1. Código fuente

Todo el código fuente utilizado en este proyecto se encuentra alojado en un repositorio de acceso público de GitHub. Al que podemos acceder desde el siguiente enlace o a través de la referencia [20].

```
    git clone https://github.com/jesusaldanamartin/
satelite_images_sigpac
```

Una vez clonado el repositorio ya puede usar la aplicación. Para ejecutar la aplicación y sacar el máximo partido a sus funciones, por favor, siga los pasos indicados y ejecute las lineas de código por comandos en una terminal.

A.2.2. Instalación de dependencias

Para la instalación de dependencias y paquetes se recomienda utilizar un entorno virtual de *Python* para evitar conflictos en caso de tener ya python instalado. En cualquier caso mediante el gestor de paquetes PyPI (Python Package Index) instalaremos todos los paquetes que aparecen en el archivo `requirements.txt` .

```
python3 -m venv <nombre_venv>

source <nombre_venv>/bin/activate

python3 -m pip install -r requirements.txt
```

A.2.3. Ejecución de aplicación

Para ejecutar la aplicación desarrollada tenemos dos opciones llamando a los archivos `run.py` o `launch.sh` en función del sistema operativo de nuestra máquina. En cualquier caso será necesario sustituir los parámetros que se encuentran entre <> por las rutas de los archivos

de nuestra máquina local.

- -r : Ruta hasta el raster de clasificación.
- -s : Ruta hasta el shapefile.
- -o : Nombre del archivo resultante.
- -t : Yes en caso de que queramos guardar los archivos intermedios temporales, No si queremos descartarlos.

```
bash launch.sh -r <raster path> -s <shp path> -o <output path>
    -t <delete tmp>
```

Test unitarios

Para comprobar que los tests se superan correctamente en caso de que hagamos algún cambio en alguna de las funciones principales, tendremos que ejecutar el siguiente comando indicando la carpeta donde se encuentren los *tests*.

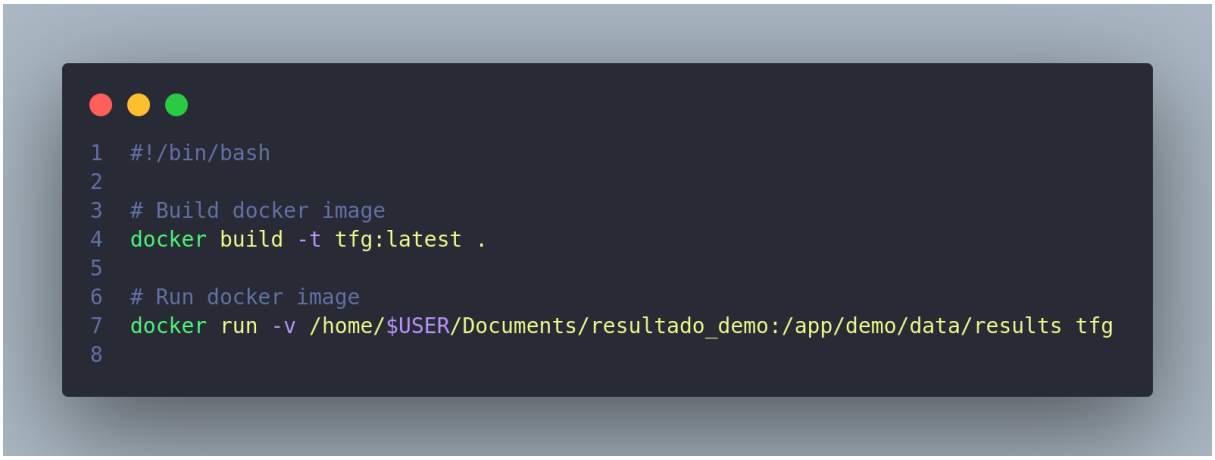
```
pytest tests/
```

A.3. Demo

En caso de no tener algunos de los datos de entrada de la aplicación. Podemos ejecutar esta *demo* y comprobar el funcionamiento del algoritmo y sus resultados. La demo consta de algunos archivos modificados para reducir el tiempo de ejecución al mínimo.

Para ejecutar el *Dockerfile* y, por tanto, la demo solo es necesario llamar al archivo `deploy.sh` de la siguiente manera:

```
bash deploy.sh
```



```
#!/bin/bash

# Build docker image
docker build -t tfg:latest .

# Run docker image
docker run -v /home/$USER/Documents/resultado_demo:/app/demo/data/results tfg

```

Figura A.1: Contenido del Script de lanzamiento de la demo.

En caso de ejecutar el script desde una máquina Unix o MacOs es posible que debamos de ejecutarlo con permisos de super-usuario (sudo). En caso de usar una máquina Windows deberíamos de ejecutar la terminal *WSL* con permisos de administrador.

La siguiente figura A.2 muestra el contenido del Dockerfile que se ejecuta cuando llamamos al script de lanzamiento:

```
FROM python:3.11

WORKDIR /app

#COPY FOLDERS NEEDED
COPY /src /app/src/
COPY /json /app/json/
COPY /demo /app/demo/
COPY requirements.txt /app

#COPY FILES NEEDED
RUN pip install -r requirements.txt

#RUN THE SCRIPT
CMD ["python3", "/app/src/demo.py", \
        "/app/demo/tif/mask_demo.tif", \
         "/app/demo/shp/sp22_REC_41086.shp", \
         "demo", "no"]
```

Figura A.2: Contenido del Dockerfile.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga