1.0 TFG

1.1 resumen

Pipeline de Datos para una Aplicación de datos Agroalimentarios.

Resumen: El trabajo consistirá en la reestructuración de una aplicación de gestión de datos agroalimentarios con el objetivo de soportar la gestión y el mantenimiento de los datos de diferentes clientes. Estos datos, actualmente, se actualizan e incrementan de forma periódica con un añadido de trabajo manual que se puede optimizar mediante técnicas de integración y despliegue continuo.

1.2.0 INTRODUCCIÓN

1.2.1 contexto

El propósito de este proyecto es mejorar el flujo de trabajo del equipo de "SpectralGeo" mediante la automatización del proceso de recogida y procesado de datos agronómicos que se utilizan en la creación de modelos de inteligencia artificial para el cultivo.

AGRAI es una aplicación para la gestión de cultivo que permite al agricultor monitorizar el estado de sus parcelas. El estado actual de la aplicación se centra el despliegue de datos agronómicos y vegetativos georreferenciados a través de en una interfaz web. Los usuarios de la aplicación, normalmente agricultores o cooperativas, pueden consultar el estado de su parcelario junto con algunas predicciones, como puede ser la cantidad de Kg que se van a cosechar en una fecha determinada. El acceso a dicha aplicación puede darse desde equipo de sobremesa o un dispositivo móvil, aunque es este último lo que parece que se utiliza más.

caption

Actualmente, cómo se procesa la información que utiliza nuestra aplicación es un proceso tedioso para el equipo. En este proceso, diferentes miembros trabajan con tecnologías distintas sobre datos duplicados provenientes de fuentes comunes. Aunque se realiza una planificación y coordinación de los proyectos, se pierde bastante tiempo en la transformación de los datos que cada miembro del equipo necesita para llevar a cabo su labor.

El objetivo de este trabajo es mejorar y automatizar el flujo de trabajo del equipo, convirtiendo la aplicación AGRAI en una herramienta robusta que manipule un único repositorio de datos al cual el resto del equipo pueda acceder, utilizándolo de forma segura y -lo más importante,-sin duplicar y romper la integridad de estos datos.

Es un proyecto ambicioso debido a que no solo es importante el conocimiento técnico sino que serán necesarios cambios en la forma de trabajo del equipo, cambios que se basan en la confianza de cada miembro en la nueva forma de trabajo que se desea implementar.

1.2.2 previos

SpectralGeo se ha especializado en el uso de nuevas tecnologías para sectores como el de la agricultura o el reciclaje, centrándose en proyectos con carácter reivindicativo de la sostenibilidad medioambiental. La relación con clientes como Ecoembes o el desarrollo de software para la gestión sostenible de cultivos lo demuestran.

Durante las prácticas realizas en la empresa se identificó que era totalmente necesario dotar de una arquitectura robusta a la aplicación AGRAI. En estas prácticas se comenzó el desarrollo de un modelo de datos sobre el que dicha aplicación pudiese escalarse posteriormente.

Actualmente, AGRAI aún necesita una reestructuración para la gestión de grandes volúmenes de datos agronómicos, poniendo especial énfasis en los índices vegetativos que gestiona, los cuales provienen del procesamiento de imágenes satelitales. El estado actual de dicha aplicación consiste en la monitorización de cultivos a través del análisis de históricos de datos.

La aplicación ya está creada; el cliente obtiene los resultados que espera cuando consulta el estado de su parcelario en la interfaz de la aplicación. Como equipo, nos organizamos para que la información y los modelos predictivos lleguen al cliente a través de esta, pero son varios los puntos en los que trabajamos en exceso para finalmente mostrar una "predicción" al cliente.

El valor más importante que predecimos es el número de kg que se espera cosechar en la fecha estipulada para el agricultor. Esta predicción hace uso de valores de índices vegetativos a partir de imágenes satelitales tomadas en diferentes fechas. El histórico de datos registrado permite estudiar la evolución del cultivo y, finalmente, mediante métodos de Inteligencia Artificial IA, predecir un valor en la fecha en la que se espera cosechar.

Serán las etapas de infraestructura, diseño, implementación y modelado las que buscaremos optimizar en el contexto de todo el equipo, debido a que estas etapas se reparten entre sus miembros.

caption

Justificamos este cambio como parte de una búsqueda de procesos esbeltos (lean), que consisten en la eliminación de los "desperdicios", o fuentes de despilfarro de tiempo y trabajo en la elaboración de productos o servicios. A través de la solución que vamos a implantar buscamos la optimización continua del proceso y la aceptación de esta cultura de optimización por el equipo.

Los puntos más importantes de los procesos "lean" son los siguientes:

- identificar los desperdicios y tratar de eliminarlos
- mejorar la comunicación interna de la organización
- reducir costes y tiempos de entrega y mejorar la calidad

1.3.0 PLANIFICACIÓN

1.3.1 alcance

El proyecto planteado inicialmente por la empresa tiene una planificación de cuatro meses. A continuación se expondrán los objetivos generales del proyecto para aportar una visión global del trabajo.

- Automatizar el despliegue de la aplicación mediante infraestructura como código.
- Preparar diferentes entornos con las tecnologías necesarias para cada parte de la aplicación.
- Identificar los conceptos del dominio agronómico y de índices vegetativos susceptibles de ser implementados.
- Diseñar un modelo de datos que permita un rápido escalado de dicha aplicación.
- Orientar la BD para manejar grandes volúmenes de datos.
- Automatizar el proceso de trasformación de los datos mediante el diseño de un pipeline.
- Diseñar un algoritmo que exporte la vista minable necesaria para predecir la producción de un cultivo (kg).
- Probar técnicas de Inteligencia Artificial para entrenar el mejor modelo de regresión para la producción anterior.
- Utilizar el modelo de producción de cultivo para predecir nuevos valores que no han sido usados en el entrenamiento.
- Incluir los resultados en la presentación de la aplicación.

1.3.2 metodología

En el equipo hacemos uso de SCRUM para la gestión del proyecto. Dicha metodología permite un desarrollo en cascada de las diferentes tareas propuestas para la creación del pipeline.

Se usará un ciclo de vida iterativo e incremental. Dentro de las fases del proyecto (también llamadas iteraciones), se repiten de manera intencionada una o más actividades del
proyecto. Con estas iteraciones el entendimiento del producto por parte del equipo va aumentando. Las fases (iteraciones) desarrollan el producto a través de una serie de ciclos repetidos, mientras que los incrementos van añadiendo sucesivamente funcionalidad al
producto. Esto, en definitiva, consiste en varios ciclos de vida en cascada. Al final de cada iteración se entrega una versión mejorada.

caption

1.3.3 EDT

Mostramos el desglose de las tareas más importantes para la correcta realización del proyecto.

graph LR; TFG-->PLANIFICACIÓN; TFG-->ANÁLISIS; TFG-->INFRAESTRUCTURA; TFG-->DISEÑO; TFG-->PIPELINE; TFG-->MODELO_IA; TFG-->SEGUIMIENTO; TFG-->MEMORIA; TFG-->CONCLUSIONES; ANÁLISIS-->DOMINIO; ANÁLISIS-->FUENTES; ANÁLISIS-->REQUISITOS; INFRAESTRUCTURA-->CONTROL_VERSIONES; INFRAESTRUCTURA-->ENTORNO; DISEÑO-->ANÁLISIS_Y_REDISEÑO; DISEÑO-->COMPONENTES; DISEÑO-->MODELO_DATOS_RELACIONAL; DISEÑO-->SERVICIOS; REQUISITOS-->FUNCIONALES: REQUISITOS-->NO_FUNCIONALES; PIPELINE-->FUENTES_DE_DATOS; PIPELINE-->DEFINICIÓN_ETAPAS; PIPELINE-->VISTA_MINABLE; MODELO_IA-->DATOS; MODELO_IA-->MODELO_PRODUCCIÓN; MODELO_IA-->PREDICCIONES;

Tarea	Descripción
PLANIFICACIÓN	
INTRODUCCIÓN	Explicamos el contexto del problema planteado y cómo vamos a implementar la solución.
ALCANCE	Qué pretendemos conseguir con el proyecto acontando las tareas
RECURSOS HUMANOS	Aquellas personas que intervienen y la explicación de sus funciones
COMUNICACIONES	Elementos de comunicación interna y externa, incluimos interesados y clientes.
METODOLOGIA	Conjunto de procedimientos que usaremos para la planificación y gestión.
EDT	Estructura en árbol con el desglose en tareas del trabajo
ENTREGABLES	Creación de los paquetes de trabajo del proyecto
ESTIMACIÓN DE TIEMPOS	Estimanción en horas para cada paquete de trabajo
GANTT	Tiempo determinado en el calendario para cada uno de los paquetes de trabajo.
PLAN DE RIESGOS	Determinaremos los principales riesgos que pueden suceder durante el proyecto y los categorizaremos por nivel de impacto y de suceso.
ANÁLISIS	
CONCEPTOS PREVIOS	Explicamos el contexto del problema planteado y cómo vamos a implementar la solución.
PROYECTO AGRAI	Presentamos el estado actual de la aplicación AGRAI.
AUTOMATIZACIÓN	Exponemos los distintos pasos a través de los cuales se transforman los datos.
CATÁLOGO DE REQUISITOS	Obtendremos los requisitos funcionales y no funcionales del proyecto.
INFRAESTRUCTURA	
ENTORNO	Decidimos los entornos de desarrollo necesarios así como las tecnologías necesarias en cada punto.
INFRAESTRUCTURA CÓDIGO	Desarrollamos un script para dejar la máquina que contiene la aplicación en un estado estable con todas sus dependencias instaladas.
DISEÑO	

Tarea	Descripción
ANÁLISIS MODELO PREVIO	Explicamos el contexto del modelo de datos previos y sus fallos.
REDISEÑO ENTIDADES	Explicamos qué entidades conformarán el modelo y por qué la base de datos quedará normalizada.
DISEÑO DE SERVICIOS	Encapsulamos las consultas más frecuentes en servicios fáciles de acceder.
CARGA CON DATOS	Cargamos el modelo con datos para verificar su consistencia.
DESPLIEGUE DE APLICACIÓN	Desplegamos la aplicación para comprobar su funcionamiento.
PIPELINE	
FUENTES DE DATOS	Explicamos las fuentes de datos con las que trabajamos.
DEFINICIÓN DE ETAPAS	Acotamos y definimos los pasos en los que se transforman los datos hasta llegar al modelo de forma consistente.
VISTA MINABLE	Seleccionamos del modelo relacional las 'features' para el modelo IA de producción.
MODELO IA	
DATOS	Exponemos los datos que vamos a utilizar, procedentes de la vista anterior.
MODELO DE PRODUCCIÓN	Usamos varias técnicas de Inteligencia Artificial para crear un modelo de producción de Kg para el cultivo.
PREDICCIONES	Utilizamos el modelo anterior dentro del sistema para predecir los kg de cosecha a partir de una fecha dada.
SEGUIMIENTO	
CONTROL	Realizamos pruebas de integración para asegurar la consistencia de los datos.
GESTIÓN DE TIEMPOS	En una tabla registramos el tiempo estimado, el tiempo real invertido y la desviación del tiempo en cada una de las tareas del proyecto.
MEMORIA	
ESTRUCTURA	Diseñamos la estructura basada en múltiples notas con la documentación y explicaciones del proyecto.
REDACCIÓN	Redactamos la memoria a medida que avanza el proyecto.
EXPORTACIÓN	La memoria se escribe en notas de "markdown" debido a la simplicidad con la que se pueden integrar diagramas y código. Al terminarl, exportaremos la memoria como páginal "html".
REVISIÓN	Al finalizar el proyecto, revisamos el escrito para asegurarnos de no cometer errores.

1.3.4 entregables

Para acotar el proceso de trabajo identificamos los siguientes entregables que generaremos a lo a medida que avance el proyecto. Estos entregables quedarán en la memoria en sus correspondientes apartados o como anexos si tienen una extensión más larga. La siguiente tabla contiene que lleva a cada uno de estos artefactos.

IDENT	ENTREGABLE	DESCRIPCIÓN
E01	Módulo de Inicio	Análisis de viabilidad que permitirá determinar si es posible desarrollar el proyecto.
[E02]	Planificación del proyecto	Determinaremos los requisitos del proyecto, crearemos un enunciado para el alcance, realizaremos una descripción detallada de tareas junto a una EDT, estimaremos el tiempo y coste de los paquetes de trabajo, desarrollaremos un cronograma, estableceremos los estándares, procesos y métricas de calidad, determinaremos un plan de identificación de riesgos y crearemos el plan de gestión de cambios.
E03	Infraestructura como código	Desarrollo de un 'script' con las instrucciones bash para dejar la máquina que despliega la aplicación en un estado estable.

IDENT	ENTREGABLE	DESCRIPCIÓN
E04	Documento de Diseño	Documento en el que se explican las decisiones tomadas para la creación de las entidades del modelo y sus relaciones.
E05	Esquema modelo de Datos	Esquema UML que representa las entidades del modelo y sus relaciones. Modelo simplificado de datos exraido del ORM Django
E06	Arquitectura Pipeline	Arquitectura y diagramas del proceso de transformación de datos, junto con sus scripts. Memoria + Código
E07	Vista Minable	Método algorítmico que genera la estructura tabular de datos necesaria para el modelo de producción
[E08]	Datos Vista	Estructura tabulada con las 'features' necesarias y los datos requeridos. (Estudio con 25 parcelas)
[E09]	Modelo de producción	Cuaderno jupyter en el que buscamos el mejor modelo posible

1.3.5 tiempos

Para poder realizar un posterior seguimiento y control del proyecto asignamos a cada tarea un tiempo adecuado a la complejidad que estimamos para ésta. En caso de que posteriormente nos alejemos de lo que aquí planificamos, anotaremos dichas desviaciones en la sección de 'seguimiento y control'

V	Tarea	Horas
1.0	DOP	10
1.1	INTRODUCCIÓN	1
1.2	ALCANCE	1
1.3	RECURSOS HUMANOS	1
1.4	COMUNICACIONES	1
1.5	METODOLOGIA	1
1.6	EDT	1
1.7	ENTREGABLES	1
1.8	ESTIMACIÓN DE TIEMPOS	1
1.9	GANTT	1
1.10	PLAN DE RIESGOS	1
2.0	ANÁLISIS	10
2.1	CONCEPTOS PREVIOS	1
2.2	PROYECTO AGRAI	1
2.3	PIPELINE	1
2.4	CATÁLOGO DE REQUISITOS	1
3.0	DISEÑ0	10
3.1	ANÁLISIS MODELO PREVIO	1
3.2	REDISEÑO ENTIDADES	1
3.3	CARGA CON DATOS	1
3.4	DESPLIEGUE DE APLICACIÓN	1
3.4	PIPELINE	1
4.0	IMPLEMENTACIÓN	10
4.1	DATOS DE ENTRADA	1

V	Tarea	Horas
4.2	DEFINICIÓN DE ETAPAS	1
4.3	VISTA MINABLE	1
4.4	MODELO DE PRODUCCIÓN	1
4.4	AUTOMATIZACIÓN COMPLETA	1
5.0	PRUEBAS	10
5.1	PRUEBAS	1
6.0	MEMORIA	10
6.1	ESTUDIO PREVIO	1
6.2	ESTRUCTURA	1
6.3	REDACCIÓN	1
6.4	REVISIÓN	1
7.0	SEGUIMIENTO Y CONTROL	10
7.1	GESTIÓN DE TIEMPOS	1
7.2	REUNIONES	1

1.3.6 gantt

El siguiente diagrama propone un desglose de tareas para la planificación del proyecto. El tiempo estipulado es de dos meses y una semana.

gantt title Planificación dateFormat DD-MM axisFormat %d section ANALISIS ALCANCE: 01-01, 2d METODOLOGIA: 02-01, 3d EDT: 03-01, 5d GANNT: 04-01, 5d RRHH: 05-01, 3d COMUNICACIONES: 06-01, 5d RIESGOS: 07-01, 5d section INFRAESTRUCTURA DJANGO+JUPYTER: 08-01, 3d SCRIPT_.SH: 09-01, 4d section MODELO_DATOS RELACIONES BD: 10-01, 20d REDISEÑO: crit, 11-01, 20d section PIPELINE BLOQUES: 30-01, 10d FEATURES: 05-02, 5d VISTA: 05-02, 5d section MODELO_PRODUCCIÓN LOAD: 10-02, 5d VIEW: 11-02, 5d REGRESSION: 12-02, 10d BEST_MODEL: 15-02, 10d Phase 2 complete: milestone, 45-01, 0d section TEST TEST 1: 20-01, 30d TEST 2: 30-01, 30d Phase 3 complete: milestone, 01-03, 0d section MEMORIA MEM_WRITE: 01-01, 60d SEG_Y_CONT: 01-01, 60d Project complete: milestone, 10-03, 0d

Este diagrama es una estipulación de cómo se desarrolla el proyecto. En la sección final de seguimiento y control analizamos las desviaciones encontradas y cómo han sido solucionados estos contratiempos.

Destacamos que las tareas de escritura de la memoria y pruebas unitarias se realizan durante casi todo el proyecto. Las pruebas nos permiten asegurarnos de que el proyecto funciona adecuadamente: así no pasamos a una nueva tarea si no hemos dejado la aplicación estable al terminar la anterior.

1.3.7 rrhh

El personal implicado en el Trabajo de Fin de Grado será:

INTERESADO	LABOR
Alberto Esteban Larreina	Ejecutor y responsable del proyecto. Desarrollará el proyecto en su totalidad
Antonio Rúbio	Tutor en la empresa. Se dedicará a corregir los aspectos técnicos del proyecto referentes al desarrollo del mismo y su planificación.
Jónathan Heras	Tutor académico. Se dedicará a corregir aspectos referentes a la documentación y la forma en la que se desarrollará el proyecto y me guiará a cerca del desarrollo de este.
Marcos Martinez	Compañero en la empresa. Probará el software desarrollado con un mayor volumen de datos.

1.3.8 comunicaciones

Para mantener informados tanto al tutor académico como a la tutora de la empresa, utilizaremos los siguientes canales:

CANAL	DESCRIPCIÓN

CANAL	DESCRIPCIÓN
Reuniones	presenciales, tanto con el tutor académico como con el tutor de la empresa.
Email	forma de comunicación electrónica para concretar determinadas pautas o establecimiento de citas.
Discord	aplicación de comunicación interna para mensajes directos con los miembros del equipo.

1.3.9 riegos

El objetivo de esta tabla es aumentar la probabilidad de eventos positivos y disminuir la de los negativos.

FUENTE	RIESGO	SI SUCEDE	MINIMIZAR
Ausencia de interesados	Médio	Utilizar menos reuniones y ser independiente	Planificar reuniones con antelación
Tecnología equivocada	Medio	Estudiar nuevas posiblidades	Uso de patrones abstractos elegantes que se puedan implementar en diferentes frameworks
Pérdida código fuente	Alto	Empezar casi de cero	Backups y control de versiones
Pérdida documentación	Alto	Empezar casi de cero	Utilizar herramientas de copias incrementales
Fallo del servidor con la applicación	Alto	Utilizar y configurar uno nuevo	Backups
Falta de horas para terminar el proyecto	Medio	Dedicarse primero a los requisitos más importantes.	Realizar una buena planificación y ajustar el alcance

De la tabla anterior se prestará especial atención a los riesgos valorados como 'Alto', ya que por su incidencia en el proyecto requieren un seguimiento exhaustivo, para los cuales se desarrollarán sus correspondientes planes de contingencia.

Los riesgos de valoración Media y Baja tendrán un control basado en un plan de mitigación simple de seguimientos del cronograma, con intervalos cortos de tiempo.

1.4.0 ANÁLISIS

1.4.1 dominio agronómico

Una vez en contexto, procedemos a justificar las decisiones de diseño que la aplicación AGRAI necesita para llegar al rediseño que hemos identificado como clave.

AGRAI es una aplicación para la gestión de cultivo que permite al agricultor monitorizar el estado de sus parcelas. El estado actual de la aplicación se centra el despliegue de datos georreferenciados por medio de una interfaz web. Los resultados para el usuario de la aplicación son satisfactorios, permitiendo a éste consultar el estado de su parcelario junto con algunas predicciones en cualquier momento.

Tenemos que situarnos en el uso que se le está dando a la aplicación en relación con el diseño de ésta. La monitorización efectuada es válida para cualquier tipo de cultivo susceptible de ser fotografiado desde una imagen satelital y/o de dron. El modelo de datos que utiliza, aunque pobre al comienzo del trabajo, permite registrar tantos cultivos como un potencial cliente desee. Por otro lado, el caso real de uso de la aplicación proviene de bodegas o cooperativas vitivinícolas que quieren predecir la cantidad de kg de cosecha en una campaña determinada (2020-2021).

Como punto de partida buscaremos identificar las entidades del dominio necesarias para que nuestro nuevo modelo de datos represente y relacione toda la información con precisión.

En la interfaz web, el cliente observa un mapa con sus parcelas resaltadas en colores. Este color reflejado en los píxeles hace referencia a los índices vegetativos provenientes de las imágenes satelitales. Los clientes utilizan dicha interfaz para ver el estado de sus cultivos y observar la producción que predicen los modelos de Inteligencia Artificial generados.

caption

A nivel interno, el equipo trabaja con datos provenientes de imágenes descargadas de satélites como SENTINEL, o de grabaciones realizadas por dron cuando se requiere una mayor calidad. Estos datos se mezclan con información proporcionada por estaciones meteorológicas como el SIAR y con la información que los clientes pueden proporcionar sobre campañas anteriores.

Por otro lado, el proceso actual de esta información es tedioso para el equipo. Los diferentes miembros trabajan con tecnologías distintas sobre datos, muchas veces duplicados, que provienen de fuentes comunes. Los requisitos que identificamos en el siguiente punto consideran la idea de optimización continua necesaria.

caption

1.4.2 fuentes de datos

Mencionábamos la búsqueda de optimización del proceso como una de las principales ideas para mejorar el flujo de datos que utiliza la aplicación. El rediseño que se quiere implantar y la arquitectura resultante tienen que cumplir con los principios de dicha metodología. Para que este trabajo resulte satisfactorio, el software tiene que poder utilizarse por el equipo con agilidad, consiguiendo que quede como una herramienta física que coordine bien todas las tareas de los participantes que la usan.

Para poder realizar este proyecto decidimos trabajar con un número pequeño representativo de datos de la aplicación, actualmente en producción. Un menor volumen de información permite realizar pruebas y da pie a fijarnos en las relaciones y los esquemas estructurales con más precisión.

Como estamos buscando una optimización continua, a medida que voy haciendo pruebas con mi entorno local, un compañero se encarga de utilizar el mismo proyecto con volúmenes mayores de datos, que la aplicación utiliza en producción.
La reestructuración del modelo de BD que realizamos contempla un posterior escalado de la aplicación que permitirá la integración continua con más datos y nuevas tecnologías.

La siguiente tabla muestra las principales fuentes de datos de las que se obtienen y enlaza la información. El volumen de datos que se puede llegar a manejar es grande ya que por cada parcela se almacenan varios índices vegetativos en cada uno de sus píxeles.

FUENTE DATOS	DESCRIPCIÓN
QGIS	información geomométrica de parcelas y sus pixéles
Índices Vegetativos	información provenientes de imágenes satelitales descargadas en diferentes fechas
Cultivos / Variedad	información tabulada en excels sobre tipo de cultivos y sus variedades

Para poder realizar el trabajo utilizaremos, por tanto, una muestra representativa de los datos debido a que el proceso de descarga de índices y persistencia de datos es largo para realizar las pruebas. El sistema está pensado para trabajar con muchas parcelas; en las pruebas que yo voy a realizar escogemos una muestra de 25 parcelas y pensaremos una descarga de índices para no más de 4 fechas diferentes.

Para hacernos una idea, contemplando solo 25 parcelas, podemos almacenar 3000 píxeles. Por cada pixel vamos a registrar varios índices vegetativos (ndvi, ndre) y para generar el histórico de datos esta información se multiplica por el número de fechas contempladas. Es decir, que aunque trabajamos sobre un volumen reducido para probar la automatización, sigue siendo mucha la información que tiene que almacenarse en la base de datos.

Es el histórico de índices en diferentes fechas lo que permite al sistema realizar modelos predictivos. A mayor volumen de datos, más precisión podremos obtener en los modelos posteriormente. El punto importante de este trabajo es la mejora del proceso y la automatización del flujo de datos; ello nos llevará a que finalmente podamos obtener un modelo de regresión para predecir los kg de cultivo, tal y como el agricultor necesita. La muestra que utilizamos, al ser representativa con 25 parcelas, predecirá con un cierto sobreajuste a los datos. Por tanto nuestro esfuerzo se centra en que el proceso que implantamos sea reproducible para cualquier volumen de datos, y en que sin desarrollo adicional podamos utilizar el mismo proceso para predecir en el volumen que queramos.

1.4.3 automatizacion

El proceso que vamos a implantar requiere de un diseño que posibilite estructurar y manejar grandes volúmenes de datos. Actualmente, existen varias tecnologías que abordan el concepto de *Pipeline* desde un punto de vista global. No vamos a utilizar ninguna de estas tecnologías; nuestro proceso se va a desarrollar a medida, modelando un Pipeline como patrón de diseño, encargado de la transformación de los datos en bloques reutilizables para su persistencia en el modelo.

El primer paso para realizar una carga de datos, es el diseño de un modelo relacional que soporte esta información de la forma más estructurada y organizada posible. Hará falta la comprensión de conceptos del dominio agronómico para dar sentido a esta información y obtener así un esquema sólido sobre el que poder preguntar sin limitaciones.

graph LR; infraestructura-->diseño_relacional; diseño_relacional-->pipeline_datos; pipeline_datos-->vista_minable; vista_minable-->modelo_producción;

Con un modelo relacional para los datos, los bloques atómicos de carga se podrán ejecutar periódicamente utilizando las fuentes de datos necesarias. El proceso de automatización termina, una vez que están almacenados los datos, con la extracción de una vista minable que permite obtener el mejor modelo de regresión para los kg de cultivo que se van a cosechar en una fecha determinada.

1.4.4 tecnologías

El código de la aplicación que heredamos está escrito en Python. El lenguaje es una decisión adecuada debido a la necesidad de integrar técnicas de procesamiento de datos e inteligencia artificial. El framework de Django para Python permite construir un

proyecto robusto y desplegarlo en un servidor con una interfaz web.

Django nos ofrece las herramientas necesarias para trabajar desde un alto nivel de abstracción y poder diseñar una aplicación sólida y estable. Algunas de estas herramientas son:

- ORM, Object-Relational-Mapping: permite crear un modelo de datos y gestiona automáticamente la BD (base de datos) subyacente. Abstrae las consultas SQL y evita tener que realizar migraciones manuales de los esquemas.
- Static File Generator: podemos diseñar la interfaz de la aplicación en formato web y desplegar en un servidor.
- Commands-System: gestión de comandos internos mediante los que se puede automatizar tareas; utilizaremos esta arquitectura para diseñar el pipeline de datos y almacenar la información proveniente de diferentes fuentes.

Aunque el framework es muy potente, harán falta otras herramientas y entornos para completar con éxito la automatización que buscamos para así conseguir un proceso de optimización continua. En el siguiente punto hablaremos de la infraestructura que la aplicación requiere y de cómo podemos solventar algunos de los problemas de integración más importantes.

caption

La utilización de *Python*, aparte de incluir este framework, nos da la posibilidad de utilizar las librerías de inteligencia artificial y ciencia de datos que son necesarias para la creación del modelo de producción de cultivo. Algunas de librerías principales que vamos a utilizar son las siguientes:

LIBRERÍA	DESCRIPCIÓN
scikit-learn	creación de modelos de inteligencia artificial
numpy	tratamiento de datos multidimensionales
pandas	uso de datos tabulados con herramientas para su procesamiento
jupyter	integración de cuadernos procesables con el resto del proyecto

1.4.5 requisitos

La siguiente tabla muestra los requisitos funcionales críticos para completar con éxito el proyecto, terminando el trabajo con un producto estable que el equipo pueda utilizar.

REQUISITO	DESCRIPCIÓN
RF1	automatizar el despliegue de la aplicación identificando las librerías y dependencias necesarias
RF2	rediseñar el modelo de datos para poder escalar la aplicación y su gestión de datos.
RF3	documentar el proceso de desarrollo generando los documentos de diseño pertinentes.
RF4	crear un entorno común que permita trabajar al equipo sobre las mismas fuentes de datos.
RF5	obtener una vista minable con la selección de *features* provenientes del modelo.
RF6	creación de un modelo de producción probando varios algoritmos de inteligencia artificial.
RF7	despliegue de los datos del modelo en la interfaz de la aplicación.

Como requisitos no funcionales para la aplicación identificamos los siguientes, entendiendo que se completarán a lo largo de todo el proyecto.

REQUISITO	DESCRIPCIÓN
RNF1	Crear un entorno Linux local con la misma configuración necesaria en producción.
RNF2	Preparar las fuentes de datos con una muestra pequeña representativa de la aplicación en producción.
RNF3	Estudiar las posibles features para la vista minable que utilizará el modelo de producción de cultivo
RNF4	Implantar un flujo de trabajo en el equipo para utilizar las mismas fuentes de datos
RNF5	Utilizar software libre

1.5.0 INFRAESTRUCTURA

1.5.1 arquitectura física

El entorno local, sobre el cual desarrollamos, y el de producción, donde se despliega la aplicación, son actualmente diferentes.

Todos los miembros del equipo utilizamos máquinas Windows, mientras que en el servidor de producción encontramos una máquina Ubuntu. Esto crea un problema importante en el proceso de instalación de las librerías necesarias para trabajar con los datos; se pierde gran cantidad de tiempo en preparar el entorno de cada persona que va a trabajar con el repositorio de la aplicación; además, las librerías necesarias necesitan de una gran cantidad de dependencias que varían en cuanto a las versiones. Como solución propondremos una nueva forma de trabajo que nos asegure un entorno común con las mismas librerías y paquetes.

graph LR; subgraph desarollo WSL--> WSL.1 WSL--> WSL.2 WSL--> WSL.3 end subgraph producción WSL.1--> Ubuntu WSL.2--> Ubuntu WSL.3--> Ubuntu end BD_test-->WSL BD_prod-->Ubuntu subgraph repositorio CODE --> GIT GIT --> WSL GIT --> Ubuntu end

Parte del trabajo consiste en la automatización del despliegue de la aplicación. En este punto vemos coómo la infraestructura como código permite aislar las librerías necesarias creando un script que deja una máquina en estado estable para ejecutar la aplicación.

Para poder utilizar tecnologías que posibiliten la integración continua, transformamos el entorno de desarrollo desplegando la aplicación en UNIX. Identificamos las siguientes dependencias que necesitan ser instaladas:

- postgres 14 + postgis
- python3.10
- GDAL 3.3.2
- Django
- Virtual-Env [librerías ciencia de datos]

Suele ser complicado encontrar todas las dependencias con sus versiones correctas; en este caso el punto más complicado ha sido la instalación de Python con su versión correspondiente de GDAL, librería que permite tratar con los datos geoespaciales. Además, el sistema gestor de BD, 'postgres', necesita una extensión especial, 'postgis', para poder guardar los datos georreferenciados. Como este tipo de trabajo es casi de prueba y error, la utilidad del script que obtenemos es de gran valor.

El primero de los entregables hace referencia a esta parte del proceso. El siguiente script automatiza la creación de una máquina con las librerías necesarias, el cual, una vez ejecutado, deja la aplicación lista para su desligue.

```
#!/bin/bash

curl -fsSL https://www.postgresql.org/media/keys/ACCC4CF8.asc|sudo gpg --dearmor -o /etc/apt/trusted.gpg.d/postgresql.gpg

sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.

sudo apt update
sudo apt upgrade

sudo apt install postgresql-14
sudo apt install postgresql-14-postgis-scripts

sudo service postgresql start

sudo -u postgres createuser -P agrai_user
sudo -u postgres createdb -O agrai_user agrai_db
sudo -u postgres psql -c "CREATE EXTENSION postgis; CREATE EXTENSION postgis_topology;" agrai_db

# gdal native:
sudo add-apt-repository ppa:ubuntugis/ppa && sudo apt-get update
sudo apt-get update

# cargarte la versión de la máquina, dejar solo 1:
sudo apt autoremove python3

# python repositories

sudo apt install software-properties-common -y
sudo add-apt-repository ppa:deadsnakes/ppa

# python concrete installation

sudo apt install python3.10 # version concreta
sudo apt-get install python3.10-dev python3.10-venv
sudo apt install python3.10-dev python3.10-venv
sudo apt install virtualenv

# python env var

export PYTHONPATH="/usr/local/bin/python3.10:/usr/local/lib/python3.10/lib-dynload:/usr/local/lib/python3.10/site-packages'

alias py=python3.10
alias python=python3.10
alias python3=python3.10

# creacion entorno_venv
sudo python3.10 -m venv ../agrai_venv

# GDAL
sudo apt-get install libgdal-dev

# env gdal lib variables

export CPLUS_INCLUDE_PATH=/usr/include/gdal
```

```
export C_INCLUDE_PATH=/usr/include/gdal

# install dep in python agrai_venv

source ../agrai_venv/bin/activate
# pip install --upgrade pip # quitar
sudo python3 -m pip install -r requirements.txt
```

1.5.2 entorno

Es importante preparar un entorno común para el equipo. Django es un framework muy útil por la integración de las herramientas que hemos visto (ORM, web-interfaces, etc.), pero para conseguir un workflow adecuado necesitamos incluir otras herramientas propias de ciencia de datos como pueden ser cuadernos de jupyter y sus entornos, con las librerías necesarias para ejecutar modelos predictivos.

Realizamos una migración de todo el entorno a UNIX para posteriormente utilizar herramientas de integración continua. El trabajo en el equipo local se hace a través de una máquina en WSL (windows subsystem for Linux) para trabajar de forma semejante a un contenedor. Uno de los problemas principales para el equipo en su flujo de trabajo ha sido mantener una integridad entre el desarrollo realizado por los diferentes miembros del equipo y el posterior despliegue de la aplicación en un entorno UNIX diferente al que se había utilizado en local en las versiones de preproducción.

Dedicamos varios días a preparar la integración de Django con cuadernos jupyter y a ejecutar diferentes entornos virtuales con las librerías adecuadas en cada momento. Los siguientes ejemplos muestran cómo se ha conseguido integrar las librerías necesarias para el despliegue de los cuadernos y su integración con los datos y los modelos de Django.

```
# base code for jupyter integration

import sys, os, django

BASE_DIR = os.path.dirname(os.path.abspath('../../agrai/'))

sys.path.insert(0, BASE_DIR)

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "agrai.settings")

os.environ["DJANGO_ALLOW_ASYNC_UNSAFE"] = "true"

django.setup()
```

Los cuadernos jupyter que se encuentran en esta memoria se han exportado desde la propia aplicación. Vemos la facilidad con la que el equipo puede redactar informes para clientes concretos, utilizando los datos de la aplicación directamente. El código anterior es el 'snippet' necesario para poder integrar los cuadernos con las llamadas necesarias a los modelos de Django. Estos cuadernos son un punto tan importante de la arquitectura como puede ser el módulo de servicios.

1.5.3 UNIX en local

Hacemos uso de WSL (Windows Subsystem for Linux) para crear el entorno de desarrollo necesario para el proyecto. De esta forma conseguimos que los entornos de pruebas y producción sean muy similares, permitiendo automatizar el despliegue de la aplicación mediante técnicas de infraestructura como código. Una de las ventajas de tener una máquina Ubuntu como entorno de desarrollo es que mediante un script .sh instalamos todas las dependencias necesarias para dejar dicha máquina lista para el despliegue de la aplicación.

Para el trabajo interno de la aplicación necesitamos entornos con librerías más pesadas como numpy o scikit-learn que no deberíamos usar en la producción de la interfaz web, ya que ésta simplemente muestra los datos del parcelario registrado con su producción estimada. Consideramos la posibilidad de crear varios entornos virtuales para los diferentes flujos de trabajo que contempla la aplicación.

- Despliegue de la información de la aplicación mediante interfaz
- Técnicas de ciencia de datos para la creación de modelos a nivel interno, backend.

Actualmente, solo existe un entono virtual sobre el cual se instalan todas las librerías *Python* necesarias para el desarrollo y despliegue. Dicha refactorización sobre la infraestructura se contempla para próximos hitos en el proyecto. Es importante identificar aquellos puntos que sean propensos a un *refactor* grande, pero que no se lleven a cabo en el momento actual. Una vez identificados, valoramos su importancia y vemos si el diseño actual impediría su mejora. En este caso sería muy sencillo, una vez crezca la aplicación, separar los entornos virtuales para mejorar el tiempo de compilación y despliegue.

1.5.4 control de versiones

Para poder desarrollar sobre software que está en producción estamos utilizando un sistema de control de versiones básico. Damos unas pequeñas pinceladas sobre cómo evoluciona el software y las ramas que utilizamos.

El código de la aplicación se encuentra en un único servidor que gestiona su control de versiones a través de SVN, Subversion. El siguiente diagrama muestra el flujo de desarrollo con las ramas creadas para el trabajo independiente sin romper el producto en producción.

gitGraph commit branch origins commit tag:"v1.0.0" commit checkout main commit type: HIGHLIGHT commit merge origins commit branch featureA commit

Al plantear un nuevo rediseño del producto, el trabajo tiene lugar en una rama diferente a la que se encuentra en producción. Esta nueva rama, Origins, gestionará el rediseño del modelo de datos y la implementación del proceso de automatización de los datos. Si el trabajo evoluciona correctamente, los futuros merges no deberían ser problemáticos.

1.6.0 DISEÑO

1.6.1 componentes

Cuando comencé a trabajar con la aplicación, todo el código se encontraba en un único paquete con muy poca forma. A medida que se he ido refactorizando, se han creado entidades para modelar los datos y éstas se han estructurado en componentes atómicas de la aplicación.

erDiagram CUADERNO ||--|{ CORE : uses ROLES }|..|{ CORE : uses METEO }|..|{ CORE : uses AUTO }|..|{ CORE : uses

En Django podemos crear un proyecto compuesto por varias aplicaciones. En la documentación oficial se explica cómo funciona la arquitectura del framework y se exponen ejemplos para que, una vez creadas, podamos acoplar y desacoplar las aplicaciones al proyecto. Es una forma similar de trabajo al modo en que organizamos una librería en múltiples paquetes, pero un poco más complicada debido a que el ORM subyacente traduce los modelos a tablas en una BD. Veamos cómo con solo añadir o quitar un string en nuestro settings activamos o desactivamos la aplicación en el proyecto.

```
INSTALLED_APPS = [
    'django_apscheduler',
    'django_extensions',

    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.gis',

    'core',
    'meteo',
    'roles',
    'cuaderno'
    'auto'
]
```

Estas aplicaciones pueden comunicarse entre sí a través de sus modelos y servicios. En la siguiente tabla explicamos qué hace cada sub-aplicación.

ENTIDAD	DESCRIPCIÓN
Core	Como toda la información que se va a almacenar es referente a las parcelas y sus datos, toda la BD gira en torno a la tabla “Parcela”.
Roles	Separa los interesados que realizan acciones sobre el parcelario de las entidades principales
Meteo	Gestiona la comunicación y descarga de datos de estaciones meteorológicas
Cuaderno	Extiende el almacenamiento de datos relacionado con incidencias en el parcelario.
Auto	Contiene scripts y dependencias externas para la automatización de procesos, como la descarga de imágenes satelitáles.

Esta división en aplicaciones permite desarrollar y hacer pruebas solo con aquella parte del proyecto que necesitemos en el momento dado. Más importante aún la claridad y sencillez de los modelos de cada componente y su respectiva representación en la BD.

1.6.2 carga de datos

Un modelo claro y robusto va a permitir cargar datos en la aplicación de forma ordenada con posibilidad de ser escalados en el futuro. En la implementación del pipeline que veremos en siguientes apartados haremos hincapié en los estados de carga por los que pasan los datos; de momento solo importa destacar que el modelo actual permite mantener un histórico de datos mucho más sólido que el que estaba implantado cuando comenzamos el desarrollo.

Es importante explicar por qué vamos a poner tanto énfasis en que estos datos se gestionen de manera fluida y eficaz. La interfaz web de la aplicación muestra la punta de un iceberg en la que el usuario observa las parcelas coloreadas basándose en los índices vegetativos, también se muestra la producción estimada.

caption

El proceso de carga de datos hace uso de un módulo de descarga de imágenes satelitales que obtiene el valor de estos índices vegetativos. Este script de descarga ha sido desarrollado por el equipo y ha sido colocado dentro del módulo de automatización junto con otras tareas similares. Utilizaremos el módulo como una caja negra y aseguraremos el correcto uso de este para que

los índices vegetativos terminen en sus tablas correspondientes dentro de la BD.

erDiagram AUTO }|..|{ descarga_indices : contains AUTO }|..|{ CORE : uses

Los índices que renderiza el visor de la aplicación son el resultado de todo el proceso que estamos exponiendo. Es el punto más delicado que requiere del diseño de varias entidades para guardar un histórico de datos. En la visualización del histórico de índices vegetativos es el mayor valor que obtiene el cliente cuando accede a la aplicación.

1.6.3 modelo relacional

Como inicio de la posterior automatización, se reestructura la arquitectura de la aplicación junto con su BD para soportar el almacenamiento de nuevos datos. El siguiente esquema de BD es el resultado de la implementación del análisis descrita en el planteamiento.

caption

Para la creación de este modelo he trabajado con el equipo en la identificación de los conceptos que necesitaban ser representados. El diseño en papel da la posibilidad de pensar abiertamente sobre las relaciones entre entidades, además de permitir la transmisión de ideas de forma sencilla durante las reuniones.

Dentro del entorno de trabajo de *Django,* el ORM proporcionado aísla la base de datos y nos permite diseñar directamente en *Python* dicho modelo. Las entidades se diseñan como clases y las relaciones entre ellas se especifican mediante el lenguaje de mapeo proporcionado. De dicha forma creamos las claves foráneas que físicamente contiene la base de datos.

La utilización de este ORM es una ventaja que nos evita usar SQL directamente y permite poblar la base de datos mediante comandos *Python* desde la terminal. De todas formas, la localización física de la BD necesita ser enlazada correctamente. Este aislamiento posibilita utilizar diferentes bases de datos en los entornos locales y de producción.

Varias etapas han sido necesarias hasta llegar a un punto más o menos estable. La herramienta *Graphviz* ha permitido obtener diagramas UML que visualizan la estructura de clases del modelo a partir del código. Esta representación gráfica ha sido realmente útil para poder pensar sobre el diseño a medida que avanzaba.

Entidades Principales

De la situación inicial, nos quedamos solo con las entidades principales. Esta selección de entidades se realiza cuidadosamente para que la aplicación pueda contemplar la mayor información posible haciendo uso de un esquema sencillo pero robusto. Analizamos el diseño detalladamente en los siguientes puntos.

caption

Este rediseño contempla la mayoría de casos posibles para el posterior tratamiento y procesamiento de datos, con posibilidad de crear buenos modelos predictivos. A continuación se describen las entidades principales contempladas en la base de datos.

ENTIDAD	DESCRIPCIÓN
Parcela	Como toda la información que se va a almacenar es referente a las parcelas y sus datos, toda la BD gira en torno a la tabla “Parcela”.
Cultivo	Esta tabla está directamente unida a la anterior, ya que un cultivo puede tener muchas variedades diferentes se modela de forma recursiva con una clave foránea a sí mismo (leer el siguiente punto, Ej: Guisante es un tipo de cultivo, pero tiene varias variedades: tirabeque, snap peas…).
Indice	Esta taba es de vital importancia para el funcionamiento de la aplicación, toda nuestra información lleva a estos datos vegetativos. Serán datos calculados en un dominio, esto se hace así ya que existen cientos de formas de denominar un mismo índice de vegetación y podría ser confuso a la hora de realizar las consultas. Para ello se establecerán algunos campos determinados en el campo “tipo_indice” y se les dará un valor en “valor_indice”.
Mirar_Indice	Contempla un histórico de datos y permite dar valor concreto a un índice vegetativo en una fecha única.
Fenologico	Permite registrar diferentes estados fenológicos por los que puede pasar un cultivo. La relación entre el estado fenológico y el cultivo se realiza a través de la tabla Mirar_Fenologico.
Mirar_Fenologico	Permite dar carácter temporal al estado fenológico de un cultivo. Registramos un estado para un cultivo en una fecha concreta.
Campaña	Es una tabla muy importante, sirve para unir distintos tipos de registros (desde variedades hasta labores de campo pasando por unir los datos de parcelas y subparcelas).
Interesado	Permite realizar el diseño de roles mediante su clave foránea a una parcela. Un intereseado (stakeholder) es una persona que tiene relación con una o varias parcelas (Ej, cultivador, propietario, etc).

caption

Diseño del Cultivo y Fenología

Una decisión complicada sobre el posible histórico de datos es el registro de la evolución de un cultivo en una parcela concreta. Mediante la tabla MIRAR_FENOLÓGICO damos carácter temporal al cultivo concreto de una parcela. De esta forma podemos registrar cuáles son los estados por los que ha pasado un cultivo, desde su "siembra" hasta su "recolección".

El modelado de la entidad FENOLÓGICO ha sido una decisión complicada debido a que no se enlaza directamente con un cultivo. Entendemos que un estado como puede ser el de "siembra" tiene un carácter general y tiene sentido como entidad en sí misma (todos los cultivos pasan por siembra). Es su relación con CULTIVO mediante MIRAR_FENOLÓGICO lo que dice que dicho cultivo está en un estado fenológico concreto en un momento determinado. Por ejemplo, podemos decir que el cultivo "vid" estaba en estado de "siembra" el día "2022-01-23".

También es importante destacar que CULTIVO es simplemente el tipo que se ha registrado, por lo tanto, para que la información sea completa, un avistamiento fenológico sucede sobre un tipo de cultivo físicamente sembrado en una PARCELA. El siguiente esquema muestra cómo estas tres entidades se relacionará para dar carácter temporal y espacial a un tipo de cultivo

erDiagram CULTIVO ||--o{ CULTIVO : contains CULTIVO { string nombre FK string descripcion string es_variedad FK } FENOLOGICO { string nombre FK string descripcion } MIRAR_FENOLOGICO ||--o{ FENOLOGICO : contains MIRAR_FENOLOGICO ||--o{ CULTIVO : contains MIRAR_FENOLOGICO ||--o{ PARCELA : contains MIRAR_FENOLOGICO { string fenologico FK string parcela FK string fecha FK } PARCELA { string idx string estacion float altitud polygon geom }

Otro punto importante de la aplicación es la contemplación de variedades. Para poder mantener una jerarquía con las posibles entidades registradas en el sistema, enlazamos de forma recursiva el cultivo con una clave foránea a su misma tabla. Esta estructura permite el desglose de una jerarquía de cultivos en la que sabemos qué entidad es una subvariedad de un cultivo, dando la posibilidad de almacenar múltiples niveles.

caption

Diseño de Índices Vegetativos

Para nuestro sistema de información una parcela es una agrupación de varios píxeles. Entendemos como PIXEL a la imagen satelital más pequeña que se puede obtener sobre el terreno, a partir de la cual obtendremos los índices vegetativos. *(El valor del índice es una caja negra para este trabajo, proviene de scripts desarrollados por el resto del equipo)*

El trabajo con índices vegetativos por parte del equipo es uno de los puntos más importantes para que el rediseño sea satisfactorio. La entidad MIRAR_ÍNDICE es de vital importancia en el modelo, permite gestionar el histórico de valores de los índices para todos los píxeles. Esta tabla es una de las candidatas para realizar optimización en las consultas, algo que desde el ORM que utilizamos todavía queda un poco lejos.

erDiagram INDICE ||--o{ MIRAR_INDICE : contains INDICE { string nombre FK string descripcion } PIXEL ||--o{ MIRAR_INDICE : contains PIXEL ||--o{ PARCELA : contains PIXEL { string parcela FK polygon geom string idx } PARCELA { string idx string estacion float altitud polygon geom } MIRAR_INDICE { string indice FK string pixel FK date fecha geojson json float valor }

El volumen de datos que la tabla MIRAR_INDICE puede contener es grande. Solo con la muestra representativa de las 25 parcelas con las que estamos trabajando (3000 píxeles), dos índices registrados, y contemplando sus valores en 6 fechas distintas, obtenemos un conjunto de 36.000 valores. Cuando la aplicación escale a un mayor número de parcelas, esta tabla será susceptible de utilizar índices (BD).

Este diseño de índices permite mantener un histórico de datos preciso. Gestiona la evolución de los cultivos a través de las diferentes fechas en las que se persisten los valores de los índices. Destacamos que cada índice se corresponde con una capa de visualización sobre el visor GIS que contiene la presentación de la aplicación, requisito que hace que tenga mucha importancia el poder añadir índices a los datos a media que se vayan contemplando.

Diseño de Roles

Aunque es un punto secundario, la aplicación contempla que diferentes usuarios puedan realizar diferentes acciones. Recordamos que habíamos separado en componentes la aplicación, y los roles se encuentran en un módulo secundario independiente de las entidades anteriores, que representaban los principales conceptos agronómicos y vegetativos. El diseño modular del modelo de datos permite al equipo añadir diferentes roles sin que estos estén *hardcoded* en el código de la aplicación.

erDiagram CULTIVADOR ||--o{ INTERESADO : is CULTIVADOR { string nombre string parent FK } TECNICO ||--o{ INTERESADO : is TECNICO { string nombre string parent FK } PROPIETARIO ||--o{ INTERESADO : is PROPIETARIO { string nombre string parent FK } COOPERATIVA ||--o{ TECNICO : contains COOPERATIVA { string nombre string tecnicos FK } INTERESADO ||--o{ PARCELA : contains INTERESADO { string parcela FK string nombre } PARCELA { string idx string estacion float altitud polygon geom }

Aunque la aplicación hace uso de estas entidades, no serán tan importantes para el desarrollo posterior del trabajo, debido a que la parte que necesita automatizarse es aquella relacionada con los índices vegetativos y la geometría de las parcelas anteriores. Como decíamos, estas entidades quedan separadas en un módulo de la aplicación con la única funcionalidad de controlar el acceso a la aplicación y dar permiso a las funcionalidades que cada cliente puede realizar dependiendo del rol que tenga asignado. La siguiente tabla describe los roles hasta ahora contemplados por el equipo y su relación con el parcelario.

ROL	DESCRIPCIÓN
Cultivador	persona encargada de realizar los mantenimientos en campo
Propietario	propietario catastral de las parcelas
Cooperativa	entidad que agrupa a técnos con diferentes cargos sobre un parcelario
Técnico	personal asignado a un número de parcelas.

Modelo Final

El siguiente diagrama recoge todas las relaciones descritas anteriormente y muestra el esquema completo que usa la aplicación. En los siguientes puntos hablaremos sobre los métodos definidos sobre este esquema de datos, dónde se localizan y cómo han de ser utilizados.

erDiagram CULTIVO ||--o{ CULTIVO : is CULTIVO ||--o{ MIRAR_FENOLOGICO : contains CULTIVO { string nombre FK string es_variedad FK } INDICE ||--o{ MIRAR_INDICE : contains INDICE { string nombre FK string descripcion } PIXEL ||--o{ MIRAR_INDICE : contains PIXEL ||--o{ PARCELA : is PIXEL { string parcela FK polygon geom string idx } PARCELA ||--o{ ESTACION : contains PARCELA ||--o{ MIRAR_FENOLOGICO : contains PARCELA { string idx string estacion FK float altitud polygon geom } MIRAR_INDICE { string indice FK string pixel FK date fecha geojson json float valor } MIRAR_FENOLOGICO { string cultivo FK string parcela FK date fecha string estado } ESTACION ||--o{ ESTACION_HISTORICO : contains ESTACION { string nombre FK } ESTACION_HISTORICO { string estacion FK date fecha charfield file }

1.6.4 servicios

Ahora tenemos un modelo relacional que soporta la gestión de grandes volúmenes de datos. En este punto tenemos que diseñar las diferentes operaciones que necesitamos hacer sobre los datos almacenados. Discutimos cómo tiene que ser la implementación de las operaciones más frecuentes y por qué en un modelo de datos sin estado, con información histórica, es más importante diseñar estas operaciones sobre el modelo relacional como servicios.

El término servicio está sobrecargado y su significado adquiere diferentes matices según el contexto en que estemos. Como resultado, existe una nube de confusión en torno a la noción de servicios cuando se trata de distinguir entre servicios de aplicación, servicios de dominio, servicios de infraestructura, servicios SOA, etc. Las funciones de estos son diferentes y pueden abarcar todas las capas de una aplicación.

De hecho, un servicio es un título un tanto genérico para un bloque de creación de una aplicación porque implica muy poco. En primer lugar, un servicio implica un cliente para cuyas solicitudes está diseñado. Otra característica de una operación de servicio es la de entrada y salida: se proporcionan argumentos como entrada a una operación y se devuelve un resultado. Ms allá de esta implicación suelen estar los supuestos de "statelessness" y la idea de "pure fabrication", según GRASP:

When a significant process or transformation in the domain is not a natural responsibility of an ENTITY or VALUE OBJECT, add an operation to the model as standalone interface declared as a SERVICE. Define the interface in terms of the language of the model and make sure the operation name is part of the UBIQUITOUS LANGUAGE. Make the SERVICE stateless.

Eric Evans Domain-Driven Design

El tipo de servicios que estamos diseñando e implementando para nuestra aplicación forman parte de la capa de dominio. Estos servicios de dominio a menudo se pasan por alto como bloques de construcción clave, confundidos por el enfoque de las entidades del modelo (o value objects).

Cumpliendo con los principios mencionados, los servicios conforman la siguiente capa de abstracción del modelo de datos implementado. Colocaremos aquellas operaciones que dependan o relacionen más de una entidad en su módulo correspondiente de servicios y no como un método de la clase del modelo. Este tipo de diseño en el que dejamos el modelo casi sin métodos propios puede llegar a entenderse como un anti-patrón, anemic domain model, pero que en nuestro caso, tras buscar la forma más sencilla de acceder a la información, será de gran utilidad.

Justificamos la implementación de gran parte de los métodos sobre la capa de servicios por el tipo de información histórica sobre la que necesitamos hacer las consultas. La mayoría de entidades necesitan de una relación con otra segunda o tercera entidad para devolver la información pertinente. **Como el modelo está orientado a manejar información histórica y sin estado**, optamos por colocar casi todos los métodos de acceso a los datos en un módulo de servicios aparte. Miremos el siguiente ejemplo sobre los históricos fenológicos en el parcelario registrado.

```
class ParcelaCultivos_Service():

    def get_cultivo(parcela_p):

        """ último cultivo que se está cultivando en una parcela """

        mirar_feno = Mirar_Fenologico.objects.filter(parcela = parcela_p.idx)

        if (mirar_feno.count() > 0):

            return mirar_feno.order_by('fecha')[0].cultivo

        else:
```

```
            return None

    def get_historico_fenologicos(parcela_p):

        """ todos los estados fenológicos por los que ha pasado una parcela """

        return Mirar_Fenologico.objects.filter(parcela = parcela_p.referencia)

    def get_historico_range(fecha_inicio, fecha_end):

        """ histórico de todas las parcelas en un rango de fechas """

        return Mirar_Fenologico.objects.filter(fecha__range=[fecha_inicio, fecha_end])

    def get_historico_parcela(parcela1):

        """ todo el histórico de una parcela """

        return Mirar_Fenologico.objects.filter(parcela = parcela1)

    def get_historico_parcela_range(parcela1, fecha_inicio, fecha_end):

        """ hostórico de una parcela en un rango de fechas """

        historico = ParcelaCultivos_Service.get_historico_range(fecha_inicio, fecha_end)

        return historico.filter(parcela = parcela1)

    def get_historico_mismo_fenologico(valor_fenologico):

        """ todo los cultivos que están en un mismo estado fenológico """

        return Mirar_Fenologico.objects.filter(estado = valor_fenologico)
```

Adjuntamos como entregables del proyecto algunos de los servicios diseñados y usados más frecuentemente. El código de estos servicios es utilizado por el resto de miembros del equipo para otras tareas de la aplicación, como puede ser la presentación de los datos en la interfaz web.

El pipeline que vamos a diseñar es posible gracias a la factorización en módulos de la aplicación y a la sencillez y versatilidad del modelo relacional de datos. Los servicios que ahora implementamos abstraen al resto del equipo de la funcionalidad subyacente y me permiten crear diferentes interfaces con propósitos distintos. Son muy importantes, ya que nos acercan al flujo de trabajo de integración continua que estamos buscando.

La siguiente función es una de las más usadas debido a la integración directa con el proceso de descarga de imágenes satelitales del cultivo. Fijémonos en cómo utilizamos las entidades creadas anteriormente para acceder a la información de la BD, el ORM subyacente nos abstrae de otras consultas complejas que podríamos hacer en lenguajes como SQL.

```
def get_indice_func(parcela, indice_p, fecha_p, func, filtro):

    """Método que encapsula el uso de una función agregada junto con un filtro """

    indice_p = Indice.objects.get(nombre=indice_p)

    if ParcelaIndices_Service.is_parcela_filter(parcela, filtro):

    pixels = parcela.get_pixeles()

    indices_pixels = Mirar_Indice.objects.filter(

        pixel__in = pixels,
        indice = indice_p,
        fecha = fecha_p
    )

        if ParcelaIndices_Service.get_indice_filter(indices_pixels, filtro):

            array_valores = list(indices_pixels.values_list('valor', flat=True))

            return func(array_valores)


    return None
```

1.7.0 PIPELINE

1.7.1 transformación datos

Una vez tenemos un modelo relaciona que soporta las principales entidades del dominio agronómico que necesitamos, procedemos a cargar la información proveniente de distintas fuentes en la BD bajo sus correspondientes entidades y con las relaciones necesarias para las consultas. Para esta carga vamos a utilizar un pipeline de datos, veamos a qué nos referimos.

Es importante diferenciar qué tipo de estructura estamos construyendo. Existen actualmente varias arquitecturas para 'pipelines' como pueden ser ETLs u otras. Para la correcta carga y transformación de datos no utilizaremos ninguna de estas arquitecturas, sino que diseñaremos los bloques necesarios en cada paso para poblar nuestra base de datos. Este proceso puede entenderse como una canalización de datos, en la que recibimos información en diferentes fuentes y la dotamos de contexto dentro de la BD.

Canalización de datos

Una canalización de datos hace referencia a los pasos necesarios para mover datos del sistema de origen al sistema de destino. Estos pasos incluyen copiar datos, transferirlos desde una ubicación a otra y combinarlos con otras fuentes de datos. El objetivo principal de una canalización de datos es garantizar que todos estos pasos se produzcan de forma coherente con todos los datos.

En el apartado anterior hemos visto cómo se ha diseñado el modelo de datos. Ahora nos centramos en los pequeños pasos de carga que vamos a dar para que los datos agronómicos de clientes e imágenes satelitales se persistan en dicho modelo. Valoramos la identificación de unidades atómicas de persistencia de información que puedan ejecutarse reiteradamente. Es decir, buscaremos acotar pequeños procesos de carga que puedan ejecutarse en varios puntos dependiendo del volumen de datos de clientes que maneje el equipo en un momento dado.

Procesos de una canalización

Identificamos tres conceptos que definen la canalización de datos que vamos a llevar a cabo y exponemos que hace el 'pipeline' de nuestra aplicación en cada uno de ellos.

- Data Ingestion: diseñamos bloques atómicos de carga que persisten datos de fuentes diferentes en la base de datos de la aplicación.
- Data Transformation: utilizamos los bloques de carga para persistir los datos en nuestro modelo de Django.
- Data Storage: almacenamos los datos en el modelo a través de Django y manteniendo una base de datos relacional en la máquina en la que se encuentra la aplicación.

Como hemos dicho, hay varias formas de realizar este proceso de canalización, en nuestro caso identificaremos los bloques y diseñaremos el proceso casi de forma manual, para que se ajuste con exactitud a lo que queremos hacer. En el siguiente punto exponemos la tecnología subyacente que utilizaremos para ello.

1.7.2 design pattern

En ingeniería de software, un **patrón de diseño** es una solución general y reutilizable para un problema común dentro de un contexto dado. El término pipeline puede hacer referencia a diferentes soluciones y contextos. De momento vamos a centrarnos en su aplicación como patrón de diseño ya que en nuestra aplicación se está utilizando como tal.

**Podemos definir un pipeline como una cadena de elementos de procesamiento (procesos, subprocesos, rutinas, funciones, etc.), dispuestos de modo que la salida de cada elemento sea la entrada del siguiente; el nombre hace referencia al flujo de una o varias tuberías.

En nuestro caso vamos a utilizar la arquitectura de comandos que proporciona Django para diseñar los procesos de transformación de datos que van a poblar la base de datos. La definición y reutilización del proceso de población es muy importante debido a la forma de trabajo que tiene la empresa con diferentes clientes. Dependiendo de la magnitud del cliente se prepara una copia de la aplicación para trabajar con sus datos, por ello una vez definido las unidades atómicas del proceso de carga de datos, se podrán diseñar diferentes pipelines para cada cliente. Para este trabajo nos conformaremos con el diseño y la implementación de un proceso general, pero manteniendo la idea de que pueda ser escalado más adelante.

```
from django.core.management.base import BaseCommand

class Command(BaseCommand):

    def handle(self, **options):

        # now do the things that you want with your models here
```

Las líneas de código anteriores hacen referencia a una implementación concreta del patrón mencionado. Fijándonos en la siguiente figura, la clase **Command** sería el handler y el método **handle** corresponde con handleRequest del esquema. Los próximos scripts que creemos serán los clientes de estas clases. Resaltar que la implementación de Django de este patrón es un poco más compleja porque los comandos que encapsulan la funcionalidad permiten tomar diferentes parámetros.

caption

Con este patrón desarrollamos la infraestructura necesaria para cargar los datos en el modelo. Cada comando representa un proceso de carga de datos, el cual se puede componer posteriormente dentro de un proceso más complejo. Cada unidad de carga se comporta como un filtro que añade de forma ordenada la información al modelo de datos, enlazando las entidades con sus datos correspondientes. Utilizaremos estas unidades para generar diferentes tuberías que permitan dejar varias instancias de la aplicación en estados diferentes.

Para AGRAI, hacen falta transformar los datos de índices vegetativos que maneja el equipo en información ordenada que llegue al modelo de datos desarrollado anteriormente. A continuación explicamos el flujo de procesos de carga más utilizado.

Los datos principalmente provienen de qgis, software para trabajar con información georeferenciada. El equipo pre-procesa la información sobre los índices vegetativos y da valores a los píxeles mediante dicho software. La principal función de este trabajo es dar sentido a esa información, guardando los datos tomados junto con sus relaciones con el parcelario físico.

graph LR; qgis-->parcelas; parcelas-->cultivos; cultivos-->índices; índices-->vista_minable; vista_minable-->modelo;

La información de los píxeles tiene que enlazarse con los datos de parcelas y cultivos. Cuando obtenemos los índices vegetativos solo contemplamos la información sobre la geometría de la parcela, no sabemos nada más. Esta información tiene que almacenarse junto con el cultivo, estado fenológico, datos físicos de la parcela, etc. En la siguiente tabla explicamos cada uno de los scripts que conforman los procesos del pipeline.

SCRIPT	TAREA
load_qgis	Extrae la información geometica de cada pixel proveniente de "qgis" e inserta dicha información en las tablas de "parcela" y "pixel" de la BD.
load_parcela	Lee de un excel información fisca sobre las parcelas e inserta los datos en la BD.
load_cultivos	Lee de un excel información sobre cultivos con sus variedades e inserta los datos en la BD.
download_img	Usa uno de los módulos desarrollado por el equipo para descargar índices vegetativos a partir de imágenes satelitales. No persiste nada directamente, devuelve un dataframe con los datos descargagos.
load_indice	integra la información de índices descargada en el paso anterior en la tabla de "pixel" de la BD.

Para entender detalladamente qué hacen los scripts de carga de índices tenemos que explicar algunos de los procesos de automatización de los que hacemos uso.

graph LR; indices --> download_indices; download_indices --> download_today download_indices --> download_range

Estos scripts nos permiten crear el flujo necesario para que la aplicación procese la información y termine en un estado consistente. Parte del proceso necesita ser automatizado y ejecutado reiteradamente, añadiendo los datos a la BD y relacionándolos adecuadamente. En nuestro contexto agronómico, es la descarga de índices la parte del proceso que necesita ser automatizada, mientras que los datos de las parcelas y cultivos y la información geoespacial se pueden cargar directamente en el despliegue de la aplicación.

Con esta estructura de tuberías lista, podemos reutilizar cualquier parte del proceso. Un planificador de tareas nos ayuda a establecer el tiempo que tiene que transcurrir entre las descargas. Sin entrar en más detalle, se ha utilizado el scheduler por defecto del framework de Django que estamos utilizando para esta gestión de tiempos y descarga.

Finalmente, para realizar pruebas, queremos crear un único script que ejecute el proceso de forma ordenada. El resto de tuberías que se adjuntan en el proyecto tienen una forma similar y hacen el mismo uso de los comandos de Django que hemos explicado.

```
class Command(LoggingBaseCommand):

    help = 'Carga la geometría a las parcelas y a los pixeles'

    BASE_DIR = Path(__file__).resolve().parent.parent.parent.parent
    PARCELA_SHAPE = BASE_DIR / 'data' / 'parcelas' / '25' / 'parcelas.shp'
    PIXEL_SHAPE = BASE_DIR / 'data' / 'parcelas' / '25' / 'pixeles.shp'
    FICHERO_PATH Â = BASE_DIR / 'data' / 'excels' / Â 'datos-test-2.xls'
    TRILINEA_PATH = BASE_DIR / 'data' / 'trilineas' / 'PRilineas.json'

    def handle(self, Â *args, **kwargs):

        self.run()

    def run(self):

        print(Path(self.PARCELA_SHAPE))

        call_command('1-load-qgis',parcelas=self.PARCELA_SHAPE, pixels=self.PIXEL_SHAPE)

        call_command('2-load-parcela-data', excel=str(self.FICHERO_PATH))

        call_command('3-load-cultivos', excel=str(self.FICHERO_PATH))

        call_command('4-download-indices',parcelas=self.PARCELA_SHAPE, pixels=self.PIXEL_SHAPE)
```

Además de este último, podemos crear varios entornos con tuberías diferentes que den como resultado bases de datos con estados distintos. Como mencionábamos anteriormente, es muy útil debido al funcionamiento del equipo en relación con el procesamiento de datos con varios clientes. La aplicación puede trabajar con instancias diferentes de la misma base de datos dependiendo del proyecto en el que se encuentre, por ejemplo la misma instancia de la BD sirve tanto para una bodega con variedades de vino que para una cooperativa que contempla varios cultivos como guisantes, olivas, peras, etc.

Por último, este patrón nos permite ejecutar comandos de forma asíncrona y gestionar los procesos con un planificador. Actualmente, aunque el tiempo de carga es alto, podemos asumir una carga lenta. Pero es este mismo patrón de diseño el que nos puede permitir a futuro la carga asíncrona de los datos en cada una de las componentes que lo forman.

Adjuntamos en los entregables los scripts que forman este proceso de persistencia de datos.

1.7.3 vista minable

Hasta este punto habíamos insertado en la base da datos la información que provenía de diferentes fuentes. Hemos dotado de

consistencia y relación a los datos y se ha creado un sistema con capacidad para hacer consultas sobre estos de forma ordenada y sencilla.

Ahora, fuera de la estructura de comandos de *Django* y utilizando el entorno virtual creado y los cuadernos integrados en la aplicación, escribimos el algoritmo para extraer los datos y dar la forma necesaria para los siguientes pasos.

El código que mostramos en este apartado corresponde con la generación de la vista tabulada con la que vamos a crear el modelo de producción de cultivo. Ahora realizamos el proceso inverso a la persistencia de datos que hemos realizado hasta este punto. Utilizamos la información y sus relaciones persistidas en la base de datos de nuestro sistema para generar la vista minable que necesita el modelo predictivo.

```
def statistic_indices(indices = ['ndvi', 'ndre'], func=np.mean):

    df = pd.DataFrame()

    col_ids = []

    for p in Parcela.objects.all():

        col_ids.append(p.idx)

    df['IDX'] = col_ids

    # para cada indice:
    for ind in indices:

        indice_p = Indice.objects.get(nombre=ind) # if type(indice_p) == str else indice_p

        for fecha in fechas:

            col_data = []

            # por cada iteración de esto tienes una fila en el df:
            for p in Parcela.objects.all():

                # la media de todos sus índices
                p_indices = Pixel.objects.filter(parcela=p)

                list_values = []

                for p_itr in p_indices:

                    qs =  Mirar_Indice.objects.get(

                        pixel = p_itr,
                        indice = indice_p,
                        fecha = fecha # fecha para la columna:
                    )

                    import math

                    if ( not math.isnan(qs.valor) ):

                        list_values.append(qs.valor)

                # estadistico:añadir columna

                res = func(list_values)

                col_data.append(res)

            df[func.__name__ + '_' + ind +'_' +str(fecha)] = col_data

            # valor a la tabla

    return df
```

Selección de features

El diseño de la vista minable es complicado, para ello trabajo con el equipo seleccionando los campos de la BD más importantes que formaran la tabla. Nos centramos principalmente en los índices vegetativos registrados en diferentes fechas.

Para poder predecir la producción de cultivo incluimos en la vista varios estadísticos como la media y el sumatorio para cada índice en cuatro fechas representativas de la evolución de este. El diseño de esta estructura proviene de un estudio previo realizado por el equipo, en el que se ha concluido que ciertos estadísticos en unas fechas calibradas funcionan bien para predecir la producción.

Hablábamos anteriormente de entornos virtuales, por el momento la aplicación utiliza un solo entorno virtual Python, pero este punto del trabajo podría hacer uso de un entorno separado con las librerías de ciencia de datos instaladas, de tal forma separaríamos dos procesos importantes. %%Resaltar que en el trabajo me estoy encargando del proceso para hacer pruebas, pero serán diferentes miembros del equipo los que usen estas tecnologías y para que podamos seguir escalando a medida que evolucione la aplicación, estos miembros tienen que poder utilizar los entornos creados.%%

Variable Objetivo

Junto con los datos extraídos añadiremos la variable objetivo de producción. Esta variable, aun sin almacenar en el modelo, hace

referencia a los kilos de producción de cada parcela en su cosecha. A partir de ella, obtenida de históricos de campañas anteriores, junto con las features anteriores, vamos a poder predecir la evolución del cultivo con precisión en estados concretos de su maduración. Se explica con detalle en el siguiente punto.

1.7.4 modelo de regresión

La vista minable que hemos obtenido representa los datos necesarios para crear un modelo de producción para el cultivo. El próximo entregable es el desarrollo de uno o varios cuadernos de jupyter con modelos de inteligencia artificial para la tabla anterior.

caption

En el punto en el que estamos podemos pensar en las múltiples vistas que se pueden generar a partir de los datos persistidos en el BD. Desde aquí desarrollaremos la automatización de la búsqueda del mejor modelo posible para la vista anterior, pero siempre teniendo en cuenta que a partir de los datos persistidos podemos predecir muchos otros valores, no solo la producción de un cultivo.

```
X_train, X_test, y_train, y_test= train_test_split(

    X,y,
    train_size   = 0.8,
    random_state = SEED,
)
```

Resaltamos esta idea debido a que es el punto más fuerte del trabajo realizado. La arquitectura obtenida para la aplicación nos permite generar diferentes vistas con el objetivo de utilizar la información en diversos estudios. Los índices vegetativos y la información persistida acerca de los cultivos se puede presentar en el formato necesario que el estudio requiera.

Por último, deberíamos poder utilizar este modelo obtenido dentro de la arquitectura de la aplicación para predecir casi en tiempo real cómo evolucionan los cultivos que se están monitorizando. El punto en el que se encuentra la arquitectura soporta casi de forma directa la inclusión de las predicciones de la aplicación.

```
clf = RandomForestRegressor(**CV_rfr.best_params_)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

print(y_pred)
```

1.8.0 MODELO IA

-- LINK-NOTE --

[[preprocesamiento]]

[[modelo de regresión]]

[[mejor modelo]]

1.8.1 preprocesamiento

Como los datos con los que trabajamos en los cuadernos provienen de una base de datos estructurada y previamente estudiada, no va a hacer falta un paso de preprocesamiento previo. De todas formas, sí que intentaremos que los datos de producción con los que vamos a predecir estén normalizados y se haya hecho una búsqueda previa de 'outliers'.

Buscamos un modelo de regresión para el número de Kg de cultivo en cada parcela. Aunque los datos son representativos de un parcelario pequeño, diseñaremos los cuadernos para automatizar la búsqueda del mejor modelo con cualquier volumen de datos que podamos necesitar más adelante.

El rango de valores para la mayoría de los índices que obtenemos va de -1 a 1. Los datos de índices negativos que llegan a la vista se pueden considerar como 'outliers'. Esto es debido a que si los índices provienen de una imagen con nubes, no se diferencian los colores del terreno y el índice acaba teniendo un valor muy malo. Para evitar valores corruptos eliminaremos los índices que provengan de la toma de la imagen en una fecha en la que había nubes.

caption

El siguiente repositorio contiene los cuadernos necesarios para la gestión y automatización de los modelos de producción generados a partir los datos de las 25 parcelas estudiadas. Recordamos que estamos pensando en escribir el código necesario para automatizar en la medida de lo posible la búsqueda del mejor modelo de producción. De tal forma que cuando el sistema contemple un mayor número de parcelas, la ejecución de los cuadernos nos siga generando el mejor modelo de producción posible.

1.8.2 modelo de regresión

La vista minable que hemos obtenido representa los datos necesarios para crear un modelo de producción para el cultivo. El próximo entregable es el desarrollo de uno o varios cuadernos de jupyter con modelos de inteligencia artificial para la tabla anterior.

caption

En el punto en el que estamos podemos pensar en las múltiples vistas que se pueden generar a partir de los datos persistidos en el BD. Desde aquí desarrollaremos la automatización de la búsqueda del mejor modelo posible para la vista anterior, pero siempre teniendo en cuenta que a partir de los datos persistidos podemos predecir muchos otros valores, no solo la producción de un cultivo.

```
X_train, X_test, y_train, y_test= train_test_split(

    X,y,
    train_size   = 0.8,
    random_state = SEED,
)
```

Resaltamos esta idea debido a que es el punto más fuerte del trabajo realizado. La arquitectura obtenida para la aplicación nos permite generar diferentes vistas con el objetivo de utilizar la información en diversos estudios. Los índices vegetativos y la información persistida acerca de los cultivos se puede presentar en el formato necesario que el estudio requiera.

Por último, deberíamos poder utilizar este modelo obtenido dentro de la arquitectura de la aplicación para predecir casi en tiempo real cómo evolucionan los cultivos que se están monitorizando. El punto en el que se encuentra la arquitectura soporta casi de forma directa la inclusión de las predicciones de la aplicación.

```
clf = RandomForestRegressor(**CV_rfr.best_params_)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

print(y_pred)
```

1.8.3 mejor modelo

Tras conseguir los datos con la forma correspondiente, exponemos los modelos de inteligencia artificial que vamos a utilizar para buscar el mejor modelo de producción de cultivo (kg) posible.

El problema que estamos buscando resolver es un problema de regresión, en el que conocemos la variable objetivo por el estudio de campañas anteriores. Los clientes, en este caso agricultores, quieren poder predecir cuántos kg de cultivo van a poder cosechar al final de la temporada (septiembre), en estados previos de maduración (enero).

Los cuadernos que conforman el entregable, contienen código que busca el mejor modelo posible para los datos importados con los índices y sus estadísticos. Los índices vegetativos registrados en diferentes fechas junto con la variable de producción de campañas anteriores tienen que poder predecir con precisión la campaña actual.

```
# drop columns with negative mean.

for col in df.columns:

    df2 = df[col].mean()
    if (df2 < 0):
        df = df.drop(columns=[col], axis =1)
```

Antes de ejecutar código para los modelos, analizamos las relaciones entre las columnas y hacemos estudios de correlación y análisis de componentes principales, para ver si algunas de las fechas que han llegado a este punto no añaden valor a la información que queremos predecir. Recordemos que puede que lleguen índices provenientes de nubes y entonces la información empeora el modelo.

Un análisis de componentes principales y la observación de las mejores columnas son el siguiente paso a la limpieza de columnas que se ha realizado en el preprocesamiento.

```
rfe = RFE(estimator=DecisionTreeRegressor(), n_features_to_select=5) Â
model = DecisionTreeRegressor()
pipeline = Pipeline(steps=[('s',rfe),('m',model)])

cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(pipeline, X, y, scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)

print('MAE: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

Para la variable objetivo de producción también hacemos un pequeño análisis. Queremos ver si esta contiene outilers, para ello analizamos la distribución y seleccionamos aquellos valores que se alejan mucho de la media y desviación típica de la muestra. Es importante que los datos con los que vamos a predecir estén limpios para que el modelo obtenga una buena métrica, como puede ser el "mean absolute error", MAE, puntuación para problemas de regresión en el que el resultado está contemplado en las unidades de la variable analizada.

Una vez hemos analizado los datos, hemos limpiado los valores erróneos de los índices, procedemos a probar aquellos modelos

de inteligencia artificial que mejor puedan predecir este valore de kg de producción. Los modelos que probamos son los siguientes.

- SVR
- MLP
- Lineales {Lasso, Ridge, Elastic}

El modelo que mejor MAE obtienen es la red neuronal, esperamos que no sobre-ajuste a los datos, aunque cabe la posibilidad.

```
# MLP
from sklearn.neural_network import MLPRegressor

# utilizamos los mejores parámetros observados en pts anteriores:
mlp = MLPRegressor(

Â  Â  **{
        'activation': 'tanh',
        'alpha': 0.05,
        'hidden_layer_sizes': (120, 80, 40),
        'learning_rate': 'adaptive',
        'max_iter': 50,
        'solver': 'sgd'
Â  Â  }
)
```

Este pequeño estudio nos permite orientarnos para ver por dónde van los datos, pero no es nuestro objetivo que el modelo prediga con exactitud, solo queremos dejar automatizada una pequeña visualización de las relaciones entre los índices y así poder intuir que modelo nos puede funcionar mejor. De hecho, los datos representativos con los que trabajamos nos impiden buscar un modelo real preciso.

Como hemos dicho, un último punto para completar el desarrollo de la arquitectura sería hacer uso de este modelo para mostrar directamente en la presentación de la aplicación valores predichos con datos persistidos en la aplicación, algo que aún no está desarrollado, pero que queda trivial con la arquitectura y modelo de datos obtenidos hasta este punto.

1.9.0 SEGUIMIENTO

-- LINK-NOTE --

[[desviaciones]]

[[memoria]]

1.9.1 desviaciones

El proyecto se ha desarrollado sin grandes cambios relativos a la planificación estipulada, pero a pesar de realizar la planificación del proyecto con su desglose en entregables y horas para los paquetes de trabajo, el tiempo de desarrollo ha diferido de la misma en algunas ocasiones, especialmente en la memoria.

El número de horas planificadas era de 300, las cuales se repartían en varias etapas marcadas
por los diversos puntos de control: planificación, análisis, infraestructura, diseño,
implementación, modelos y seguimiento y control.

No ha habido grandes desviaciones en cuanto a la planificación estipulada en el desglose de tareas. Sí que podemos remarcar que se han dedicado más horas a la redacción de la memoria y a algunas tareas secundarias relacionadas con esta, como pueden ser la integración de los diversos diagramas que aparecen. Podemos estimar entre 15 o 20 horas más para esta tarea sobre las 20 estimadas al principio.

1.9.2 memoria

Esta memoria se incluye como un entregable del proyecto debido a las explicaciones y esquemas que contiene (principalmente para el modelo relacional). Para poder mejorar el flujo de trabajo, ciertos conceptos necesitan ser entendidos por los desarrolladores. La documentación aquí expuesta tiene un gran valor por el lenguaje ubicuo que implanta en el proyecto.

El documento se ha escrito como notas en 'markdown' por la facilidad que da para incluir trocitos de código relativos al proyecto, y así poder explicar decisiones y conceptos. Para integrar los diagramas expuestos utilizamos tecnologías como 'mermaid' que posibilita editar los diagramas directamente en el documento, algo totalmente necesario cuando estos modelos se modifican a medida que avanzamos.

Finalmente, exportamos las notas en el orden que necesario para la memoria y renderizamos el documento en html. El fácil acceso al documento tiene bastante relevancia por el uso que el equipo puede darle al consultar la documentación, especialmente la del modelo relacional y los servicios.

1.10.0 0-conclusiones

-- LINK-NOTE --

[[conclusiones]]

[[bibliografía]]

1.10.1 conclusiones

Este trabajo se ha centrado en el desarrollo de un proceso de automatización que permite la integración continua del flujo de trabajo del equipo para una aplicación de gestión agronómica. Hemos pasado por casi todas las etapas de diseño e implementación, además de la final inclusión de los modelos predictivos para la producción de cultivo.

Durante el proceso hemos influido en los principales bloques de diseño e implementación relativos a la arquitectura de la aplicación, pasando por la infraestructura, modelo de datos, carga de estos datos, y la búsqueda de modelos predictivos a partir de ellos. Acorde al alcance planteado hemos llegado satisfactoriamente a todos los puntos estipulados, recodamos estos puntos:

- Obtenemos un script de creación para el entorno necesario
- Diseñamos un modelo relacional de datos que permita un rápido escalado de la aplicación.
- Automatizamos el proceso de trasformación de los datos mediante el diseño de un pipeline.
- Diseñamos un algoritmo que exporte la vista minable necesaria para predecir la producción.
- Utilizamos técnicas de lA para entrenar el mejor modelo de regresión para la producción anterior.
- Implementamos el modelo para predecir nuevos valores que no han sido usados en el entrenamiento.

Para finalizar, volvemos a los conceptos de los procesos 'lean' que mencionábamos al principio. La fácil comprensión que permiten los esquemas de datos aquí planteados dotan al equipo de trabajo el lenguaje ubicuo necesario para que el software pueda crecer y ser escalado evitando bloqueos o por falta de comprensión. Los artefactos obtenidos y la explicación de los modelos quedan como documentación interna que debe ser mantenida al mismo nivel que el software que la implementa. El lenguaje que maneja el equipo está directamente relacionado con ellos y el buen funcionamiento de la optimización continua dependerá de ello.

graph LR; infraestructura-->diseño_relacional; diseño_relacional-->pipeline_datos; pipeline_datos-->vista_minable; vista_minable-->modelo_producción;

En cuanto a la aplicación, se ha conseguido terminar con un producto funcional que cumple con los requisitos planteados. El proceso de arquitectura conseguido facilita el uso de múltiples datos provenientes de las fuentes estudiadas. El esquema de datos relacional implementado soporta la carga de los índices vegetativos, así como el resto de información relativa al parcelario. Los bloques atómicos de carga diseñados, permiten planificar y automatizar la ingestión de los datos provenientes de las fuentes descritas. Por último, la capacidad para utilizar modelos de regresión sobre los datos hace dota a la arquitectura de la capa necesaria para realizar predicciones y configurar los estudios necesarios. Hemos conseguido crear las bases de un proceso que tiene un gran potencial para monitorizar y proporcionar adecuada toma de decisiones sobre conceptos agronómicos.

1.10.2 bibliografía

- plan

citar PMBOOK

mencionar el libro, no el enlace.

https://www.pmi.org/pmbok-guide-standards/foundational/pmbok#

- Infraestructura

https://en.wikipedia.org/wiki/Infrastructure_as_code

- diseño modelo relacional

Diagamas UML https://mermaid.js.org/syntax/entityRelationshipDiagram.html

Projecto Django https://docs.djangoproject.com/en/4.1/intro/tutorial01/

Django Orm: https://docs.djangoproject.com/en/4.1/topics/db/queries/

Arquitectura de Servicios http://gorodinski.com/blog/2012/04/14/services-in-domain-driven-design-ddd/

Domain Driven Design - Eric Evans

- Automatización Pipeline

Django commands para https://simpleisbetterthancomplex.com/tutorial/2018/08/27/how-to-create-custom-django-management-commands.html

https://medium.com/@bonnotguillaume/software-architecture-the-pipeline-design-pattern-from-zero-to-hero-b5c43d8a4e60

https://www.astera.com/es/type/blog/etl-pipeline-vs-data-pipeline/

- modelo-producción

https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/

https://towardsdatascience.com/random-forest-regression-5f605132d19d

https://www.kaggle.com/code/sociopath00/random-forest-using-gridsearchcv

https://scikit-learn.org/stable/auto_examples/miscellaneous/plot_pipeline_display.html#sphx-glr-auto-examples-miscellaneous-plot-pipeline-display-py

https://scikit-learn.org/stable/modules/cross_validation.html

https://www.cienciadedatos.net/documentos/py19-pca-python.html