ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA
Universidad de Córdoba

EPSC

MANUAL DE CÓDIGO

Grado en Ingeniería Informática

"Asistente para la configuración de Pipelines de Aprendizaje Supervisado mediante LLMs"

"Assistant for configuring Supervised Learning Pipelines using LLMs"

Autor: Isaac Manuel Cejudo Alfaro

Directores: Dr. José Raúl Romero Salguero

Dr. Carlos García Martínez

UNIVERSIDAD D CÓRDOBA

Contenido

1. Introducción

Este manual presenta la estructura y los componentes del código desarrollado en el Trabajo Fin de Grado. Su objetivo es entender la codificación del proyecto.

2. Organización del proyecto

El proyecto se encuentra alojado en GitHub, en el siguiente enlace:

https://github.com/p82ceali/proyecto-tfg

En él, podemos encontrar los siguientes elementos:

- **.env.example:** Este archivo es un ejemplo de como se debe generar el archivo *.env* donde se deberá indicar la *API Key* de *Gemini*.
- **.gitignore**: Archivo donde se indica los elementos que se deben subir al repositorio de *GitHub*.
- **LICENSE**: Es la licencia de este Trabajo Fin de Grado, a nombre de Isaac Manuel Cejudo Alfaro.
- **README.md**: Archivo donde se muestra como instalar y configurar el proyecto para poder utilizarlo.
- **pyproject.toml**: Son las dependencias que se deberán instalar para poder utilizar el proyecto.
- *src/*: Directorio donde se encuentra alojado todo el código del proyecto.

Como se ha comentado anteriormente, el código se encuentra en el directorio src/tfg_ml, estructurado en distintos elementos:

- *agents/:* Todos los distintos agentes que componen el sistema.
- *adapters/:* Son las herramientas o *tools* que usan los distintos agentes para utilizar las librerías externas.

- *pipelines/:* Definición de los pipelines.

- *interface/:* Donde se encuentra la interfaz de usuario.

- *context.py:* Archivo donde se configura la memoria del sistema.

3. Explicación del código

En este punto se abordará la explicación del código de cada archivo que dispone el sistema como se hizó en la Memoria. Para obtener mas información, consultar el repositorio de GitHub.

11.3.1. agents/coordinator_agent.py

Este módulo implementa el agente coordinador, la pieza central que orquesta toda la conversación. Al iniciarse, configura el modelo de lenguaje con un rol y un objetivo muy claros: gestionar las peticiones del usuario y decidir qué hacer con cada una. Para ello, carga un conjunto de herramientas de delegación. Estas no procesan datos por sí mismas, sino que actúan como pasarelas, forzando la invocación del subagente más adecuado en cada caso.

Cuando la interfaz le envía un mensaje, lo primero que hace el coordinador es un filtrado básico. Si detecta preguntas que no tienen nada que ver con *machine learning*, responde el mismo con un mensaje breve, en el idioma del usuario, y no activa más componentes. Pero si la petición sí está relacionada con ML (por ejemplo: "*describe Price column*"), el coordinador interpreta la intención a partir de las palabras clave y del contexto reciente. Entonces dispara una sola herramienta de delegación: análisis, preprocesamiento, selección de características, selección de instancias o entrenamiento/evaluación.

Esa herramienta se encarga de construir la llamada al sub-agente especializado, pasándole el *dataset* activo y los parámetros relevantes (bien porque el usuario los indico en el chat, bien porque el coordinador los dedujo del contexto). Luego espera su respuesta. Cuando el sub-agente devuelve el resultado, el coordinador lo entrega tal cual, al usuario, sin reescribir nada, para preservar la trazabilidad. Además, registra en el contexto compartido que acción se ejecutó, sobre que columnas o variables y con que resumen de resultados.

Si ocurre un error, el coordinador captura la excepción y responde de manera orientativa. De esta forma, el coordinador convierte lo que empieza como una conversación en acciones concretas sobre el pipeline. Todo ello bajo un control de delegación manual, que compensa las limitaciones del modelo jerárquico nativo de CrewAI.

11.3.2. agents/análisis_agent.py

Este agente es el especialista en análisis exploratorio de datos (EDA), una especie de lupa inicial para entender mejor los *datasets*. Este agente interpreta preguntas típicas como "descríbeme la variable A" o "dame estadísticas por categoría" e invoca a las Tools definidas en adapters/analysis_tools.py

Cuando recibe una petición del coordinador, lo primero que hace es comprobar que los datos estén cargados y que la columna solicitada exista. Si detecta que falta algún detalle, pide una aclaración mínima antes de continuar. Con los parámetros ya definidos activa herramientas como *DescribeFeatureTool* o *ComputeStatisticTool* de las que se hablará más adelante.

El agente devuelve el resultado casi tal cual, aunque puede añadir una pequeña frase de contexto si lo cree necesario.

11.3.3. agents/preprocessing_agent.py

Este agente es el encargado del preprocesamiento, la etapa donde los datos se preparan para que los modelos puedan trabajar con ellos sin problemas.

Cuando recibe una orden del coordinador, del estilo "*discretiza la edad en 5 bins*", lo primero que hace es validar la compatibilidad. Por ejemplo, evita aplicar un *one-hot encoding* sobre una columna puramente numérica cuando no tiene sentido. Una vez verificado, invoca la herramienta adecuada.

La respuesta que devuelve incluye información concreta en función de la herramienta invocada y finalmente, se la pasara al coordinador.

11.3.4. agents/feature_selection_agent.py

Este agente actúa como el especialista en selección de características, el encargado de decidir que variables merecen quedarse en el modelo y cuales conviene descartar. Para ello recurre a un conjunto de *Tools* de filtrado y ranking: desde los *k-mejores* por *chi-cuadrado* o *F-test*, hasta criterios como el umbral de varianza, la importancia medida con *Random Forest* o la correlación con la variable objetivo.

El funcionamiento es similar a los anteriores. El agente recibe una instrucción del coordinador, valida los parámetros y ejecuta la *tool* correspondiente.

A nivel práctico, el agente también actualiza el *DataFrame* activo, de manera que los siguientes pasos del pipeline trabajen ya con ese subconjunto optimo de variables.

11.3.5. agents/instance_selection_agent

Este agente es el responsable de la selección y partición de instancias, una pieza clave para garantizar que los experimentos sean consistentes y comparables. Su repertorio abarca desde submuestreos hasta particiones reproducibles en train/val/test, incluyendo variantes estratificadas cuando la distribución de clases lo exige. Todo esto se verá más en detalle más adelante.

11.3.6. agents/model_training_agent.py

Este agente es el encargado del entrenamiento y la evaluación de modelos supervisados, la fase donde el pipeline se convierte en resultados tangibles.

Recibe una petición, selecciona el algoritmo apropiado, ajusta unos hiperparámetros por defecto y entrena el modelo mediante su Tool de entrenamiento integrada con *scikit-learn*. Tras esto, calcula las métricas más adecuadas al caso: *Accuracy* y F1 para clasificación; MAE, RMSE y R2 para regresión. Además, se encarga de persistir el artefacto y guardar un archivo con las métricas obtenidas.

11.3.7. adapters/analysis_tools.py

Este archivo reúne las herramientas de análisis exploratorio de datos (EDA). Como comentamos anteriormente, su propósito es ayudar a entender que hay realmente en los datos antes de dar el siguiente paso.

En primer lugar, cada herramienta tiene clases con validación de entradas mediante *Pydantic*. La herramienta *DescribeFeatureTool*, puede señalar si una variable es numérica o categórica, cuantos valores faltan, cual es el promedio, el valor más frecuente o incluso, en el caso de columnas de texto, listar las categorías que más se repiten.

En cambio, la herramienta *ComputeStatisticTool*, permite calcular estadísticas concretas y, si se pide, hacerlo por grupos.

11.3.8. adapters/preprocesing_tools.py

Este archivo contiene las operaciones de preprocesamiento, esas transformaciones que dejan el *dataset* en un estado mucho más manejable y coherente.
Cada herramienta tiene también clases con validación de entradas mediante *Pydantic*.

La herramienta *DiscretizeFeatureTool* convierte valores continuos, como la edad, en grupos o intervalos fáciles de interpretar. Es como agrupar datos sueltos en cajones ordenados para poder compararlos mejor.

La herramienta *OneHotEncodeTool* transforma una columna de texto con categorías en varias columnas binarias, una por categoría. Así, los algoritmos reciben la información en un formato que realmente entienden, como si tradujésemos palabras en señales de encendido y apagado.

La herramienta *NullCleanerColumnTool* limpia los valores nulos de una columna.

La herramienta *DropColumnTool* es capaz de eliminar una o varias columnas del conjunto de datos.

11.3.9. adapters/feature_selection_tools.py

Este archivo reúne las herramientas de selección de características, cuyo objetivo es quedarse con las columnas del *dataset* que realmente aportan valor para el objetivo final y dejar fuera aquellas que añaden poco o nada.

La validación de entradas se realiza mediante *Pydantic*, como las herramientas anteriores.

La herramienta *SelectKBestTool* selecciona un numero fijo de columnas consideradas las mejores, en función de una puntuación calculada automáticamente.

La herramienta *VarianceThresholdTool* elimina aquellas columnas que apenas cambian y que, por tanto, aportan escasa información.

La herramienta *RFImportanceSelectTool* ordena las columnas según su importancia estimada por un modelo sencillo, lo que sirve como una guía práctica para decidir.

La herramienta *CorrelationFilterTool* retira columnas muy redundantes, porque están excesivamente correlacionadas con otras.

11.3.10. adapters/instance_selection_tools.py

Este archivo se encarga de las operaciones sobre las filas del *dataset*, es decir, sobre instancias.

En primer lugar, la validación de entradas también se realiza mediante *Pydantic*.

La herramienta *RandomSampleTool* selecciona una muestra aleatoria de las filas del *dataset*. Es muy útil para reducir el tamaño de los datos y así trabajar de manera más ágil.

La herramienta *StratifiedSampleTool* es parecida a la anterior con la diferencia que mantiene las proporciones de las clases. Por ejemplo, si el *dataset* tiene un 60% de

hombres y un 40% de mujeres, la muestra respetara esa misma proporción. Esto es clave para evitar sesgos cuando se trabaja con categorías desbalanceadas.

La herramienta *ClassBalancedSampleTool* construye una muestra equilibrada eligiendo el mismo número de filas por clase o calculándolo a partir de un máximo total.

La herramienta *ClusteredSampleTool* selecciona una muestra diversa aplicando agrupación automática sobre las columnas numéricas con *KMeans*. El usuario indica el número de grupos y, de cada uno, toma el ejemplo más representativo. Así se obtiene una muestra pequeña, pero que cubre bien la variedad del *dataset*.

La herramienta *TrainValTestSplitTool* parte el *dataset* de forma reproducible en entrenamiento, validación y prueba.

La herramienta *TimeSeriesSplitTool* realiza la división respetando el orden temporal de una columna de fecha/hora.

11.3.11. adapters/model_training_tools.py

Este archivo implementa el entrenamiento y la evaluación de modelos de aprendizaje supervisado, la etapa final donde los datos se convierten en resultados medibles.
En primer lugar, se realiza la validación de entrada mediante *Pydantic*, recogiendo tanto los datos como la variable objetivo a predecir.

A continuación, separa los datos en dos subconjuntos (entrenamiento y prueba).

Después construye el modelo solicitado dentro de los soportados: *Random Forest* (para clasificación o regresión), *Regresion Logistica* y *SVM* (solo clasificación), *Regresion Lineal* (solo regresión) y *XGBoost*.

Una vez entrenado el modelo, calcula las métricas de calidad sobre el conjunto de prueba. Finalmente, guarda los artefactos, el modelo serializado y un JSON con las métricas.

11.3.12. pipelines/workflows.py

Este archivo encapsula la orquestación *end-to-end* de una interacción. Su función es recibir el mensaje del usuario junto con el *dataset* activo, construir el *CoordinatorAgent*, crear una *Crew* con la *task* que incluye el prompt del usuario y el contexto reciente de la conversación y ejecutar la interacción completa.

De esta forma se mantiene una separación clara entre la lógica de negocio y la presentación.

11.3.13. interface/ui_panel

Este archivo define la capa de presentación de la aplicación, que en este caso es Panel. Al iniciarse, carga extensiones como *Tabulator* para trabajar cómodamente con tablas y construye una vista compuesta principalmente por un elemento para subir el *dataset*, una previsualización del *dataset* y un chat interactivo.

El flujo es sencillo: cuando un usuario sube un *dataset*, la interfaz lee el *CSV* en un *DataFrame* y lo muestra como previsualización para que el usuario pueda conocer los datos.

Cada mensaje del chat activa el pipeline en pipelines/workflows.py y muestra la respuesta final del coordinador.

11.3.14. context.py

Este módulo implementa el contexto compartido, el punto único de memoria en proceso. Su misión es conservar todo lo que va ocurriendo durante la sesión, para que el sistema recuerde lo anterior y pueda tomar decisiones con un contexto ya definido.