



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

**Sistema de Recomendación
basado en Aprendizaje
Profundo
Documentación Técnica**



Presentado por Raúl Negro Carpintero
en Universidad de Burgos — 30 de junio
de 2019

Tutor: Bruno Baruque Zanón

Índice general

Índice general	I
Índice de figuras	III
Índice de tablas	IV
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	2
A.3. Estudio de viabilidad	4
Apéndice B Especificación de Requisitos	5
B.1. Introducción	5
B.2. Objetivos generales	5
B.3. Catalogo de requisitos	5
B.4. Especificación de requisitos	7
Apéndice C Especificación de diseño	9
C.1. Introducción	9
C.2. Diseño de datos	9
C.3. Diseño procedimental	10
C.4. Diseño arquitectónico	10
Apéndice D Documentación técnica de programación	19
D.1. Introducción	19
D.2. Estructura de directorios	19
D.3. Manual del programador	19

D.4. Compilación, instalación y ejecución del proyecto	19
D.5. Pruebas del sistema	19
Apéndice E Documentación de usuario	21
E.1. Introducción	21
E.2. Requisitos de usuarios	21
E.3. Instalación	21
E.4. Manual del usuario	21
Bibliografía	23

Índice de figuras

B.1. Diagrama de casos de uso	8
C.1. Esquema del patrón MVC [1]	11
C.2. Diagrama UML del proyecto	12
C.3. Diagrama UML del paquete <i>vista</i>	13
C.4. Diagrama UML del paquete <i>controlador</i>	14
C.5. Diagrama UML del paquete <i>modelo</i>	15

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

La planificación es un punto fundamental en cualquier proyecto software. En ella, se tiene que estimar qué cosas hay que hacer y cuánto tiempo y dinero va a llevar terminirlas. De esta manera, se podrá comprobar si se llega a tiempo a los distintos plazos y así evitar retrasos, lo que supondría pérdidas de dinero y tiempo.

La planificación se puede dividir en dos partes clave:

- **Planificación temporal:** consiste en dividir el proyecto en etapas (*sprints*) con una duración de x días. En estas etapas hay que dejar claro qué cosas se quieren conseguir y cuánto tiempo o esfuerzo se va a dedicar a cada una de ellas. Los *sprints* no tienen por qué durar siempre lo mismo, se pueden ajustar a las necesidades de cada momento.
- **Estudio de viabilidad:** con este estudio se podrá ver si el proyecto puede seguir adelante (es viable) o si no, con lo que habría que tomar medidas para cambiar la situación. Se divide en:
 - **Viabilidad económica:** según la cual se deberá calcular por cuánto tiene que venderse el servicio para compensar el dinero, el tiempo y el esfuerzo puestos en el proyecto. Se incluye salario del personal y la compra de software de terceros necesario para la realización del proyecto.
 - **Viabilidad legal:** se estudia el licenciamiento del software que se va a utilizar para proceder dentro de la legalidad.

A.2. Planificación temporal

Para el desarrollo del proyecto se decidió seguir la metodología ágil *Scrum*. Al ser un proyecto académico con una sola persona trabajando en él, no se siguen a raja tabla todos los rasgos característicos de *Scrum*. Las características que se han seguido son:

- Desarrollo incremental en cada *sprint*.
- Cada *sprint* tiene una duración aproximada de dos semanas.
- Se realizan reuniones al final de cada *sprint* con el objetivo de repasar lo hecho y pensar en los objetivos del siguiente *sprint*.

A continuación se desarrolla el contenido de cada *sprint*.

Sprint 1 (- 13/12/2018)

Este *sprint* se corresponde con el milestone **Búsqueda de información inicial**. En él se estudia el capítulo dedicado a los sistemas de recomendación del libro *Mining of Massive Datasets* [?].

También se realiza el curso de *fast.ai* [?]. Además, se escoge la librería de *LightFM* y se obtiene un modelo inicial.

Sprint 2 (13/12/2018 - 18/01/2019)

Este *sprint* se corresponde con el milestone **Explorar modelo con LightFM**. En él se intenta comprender el funcionamiento de *LightFM* y se utiliza el conjunto de datos de *Movielens* con el modelo.

Sprint 3 (18/01/2019 - 16/02/2019)

Este *sprint* se corresponde con el milestone **Recomendación híbrida y por contenido con LightFM**. En él se obtienen las primeras versiones del modelo híbrido y por contenido de *LightFM*.

También se obtiene un modelo inicial con *PyTorch* y se empieza a pensar en las métricas que se usarán para evaluar los modelos.

Sprint 4 (16/02/2019 - 03/03/2019)

Este *sprint* se corresponde con el milestone **Trabajar en la documentación**. En él se empieza a ampliar los apartados de la documentación relativos a los conjuntos de datos utilizados.

Sprint 5 (03/03/2019 - 13/03/2019)

Este *sprint* se corresponde con el milestone **División de los datos en LightFM**. En él se dividen los conjuntos de datos en *train* y *test* y se obtienen de nuevo los modelos.

Sprint 6 (13/03/2019 - 28/03/2019)

Este *sprint* se corresponde con el milestone **Medidas de calidad LightFM**. En él se obtienen las métricas finales para los modelos de *LightFM*.

Además, se guardan las matrices y los modelos en archivos *pickle* como medida de persistencia.

Sprint 7 (28/03/2019 - 05/04/2019)

Este *sprint* se corresponde con el milestone **Últimos pasos con LightFM**. En él se intenta acabar todo lo relacionado con los modelos clásicos (tanto código como documentación).

Sprint 8 (05/04/2019 - 24/04/2019)

Este *sprint* se corresponde con el milestone **Ampliar documentación**. En él se amplía la documentación relativa a *LightFM* y se empiezan a obtener los primeros diagramas UML con la estructura del proyecto hasta el momento.

Sprint 9 (24/04/2019 - 08/05/2019)

Este *sprint* se corresponde con el milestone **Cambios en el diseño, Flask y DL**. En él se empieza a obtener la interfaz web con *Flask* y se escoge *Spotlight* como la librería de aprendizaje profundo.

También se modifica la documentación para ir acorde a los cambios del código.

Sprint 10 (08/05/2019 - 22/05/2019)

Este *sprint* se corresponde con el milestone **Primeros pasos con Spotlight**. En él se obtiene un primer modelo de *Spotlight* y se unifican los archivos de Entrada y Persistencia para que solo haya uno de cada.

Se continúa trabajando en la documentación.

Sprint 11 (22/05/2019 - 05/06/2019)

Este *sprint* se corresponde con el milestone **Completar modelos de Spotlight**. En él se obtienen el resto de modelos de *Spotlight* y se generalizan los métodos de lectura de datos y obtención de matrices.

Se continúa trabajando en la documentación.

Sprint 12 (05/06/2019 - 12/06/2019)

Este *sprint* se corresponde con el milestone **Diseño GUI**. En él se piensa qué forma va a tener la interfaz web obtenida a través de *Flask*.

Sprint 13 (12/06/2019 - 20/06/2019)

Este *sprint* se corresponde con el milestone **Obtención GUI**. En él se termina de crear la interfaz web y se empiezan a emplear los métodos obtenidos anteriormente para que la aplicación sea funcional.

Sprint 14 (20/06/2019 - 03/07/2019)

Este *sprint* se corresponde con el milestone **Último milestone**. En él se termina todo lo que queda por hacer en el proyecto. La interfaz es totalmente funcional y la documentación está terminada.

A.3. Estudio de viabilidad

Viabilidad económica

Viabilidad legal

Apéndice B

Especificación de Requisitos

B.1. Introducción

Este anexo recoge los objetivos generales y la especificación de requisitos del proyecto.

B.2. Objetivos generales

El proyecto persigue los siguientes objetivos generales:

- Comprender los sistemas de recomendación tanto clásicos como basados en aprendizaje profundo.
- Recoger y evaluar los resultados obtenidos por los dos modelos sobre diferentes conjuntos de datos.
- Comparar los resultados.

B.3. Catalogo de requisitos

Los requisitos derivados de los objetivos del proyecto son los siguientes:

Requisitos funcionales

- **RF-1 Gestión de los datos intermedios:** el programa tiene que ser capaz de gestionar los datos intermedios:

- **RF-1.1 Guardar matrices:** el programa tiene que ser capaz de guardar las matrices generadas por los modelos para ahorrar tiempo en siguientes ejecuciones.
- **RF-1.2 Cargar matrices:** el programa tiene que ser capaz de cargar el conjunto de datos que el usuario quiera.
- **RF-2 Gestión de los modelos:** el programa tiene que ser capaz de gestionar los modelos:
 - **RF-2.1 Mostrar modelos:** el programa tiene que ser capaz de mostrar todos los modelos disponibles.
 - **RF-2.2 Seleccionar modelos:** el programa tiene que ser capaz de dejar al usuario seleccionar el modelo que quiera.
 - **RF-2.3 Guardar modelos:** el programa tiene que ser capaz de guardar los modelos obtenidos para ahorrar tiempo en futuras ejecuciones.
 - **RF-2.4 Cargar modelos:** el programa tiene que ser capaz de cargar los modelos previamente guardados.
- **RF-3 Gestión de los resultados:** el programa tiene que ser capaz de gestionar los resultados:
 - **RF-3.1 Guardar resultados:** el programa tiene que ser capaz de guardar los resultados generados por los modelos.
 - **RF-3.2 Comparar resultados:** el programa tiene que ser capaz de comparar los resultados obtenidos por los distintos modelos.
 - **RF-3.3 Mostrar predicciones:** el programa tiene que ser capaz de mostrar las predicciones obtenidas por los distintos modelos.
- **RF-4 Gestión de los usuarios:** el programa tiene que ser capaz de introducir las valoraciones de nuevos usuarios:
 - **RF-4.1 Añadir valoraciones:** el programa tiene que ser capaz de simular la entrada de nuevos usuarios permitiendo el añadido de nuevas valoraciones.
- **RF-5 Gestión de los conjuntos de datos:** el programa tiene que:
 - **RF-5.1 Mostrar los conjuntos de datos:** el programa tiene que ser capaz de mostrar los distintos conjuntos de datos de prueba.
 - **RF-5.2 Seleccionar el conjunto de datos:** el programa tiene que ser capaz de mostrar dejar que el usuario seleccione uno de los conjuntos de datos de prueba.
 - **RF-5.3 Añadir conjunto de datos:** el programa tiene que ser capaz de dejar que el usuario añada un conjunto de datos propio.

- **RF-6 Ayuda de la aplicación:** el programa tiene que ofrecer información al usuario:
 - **RF-6.1 Mostrar información sobre modelos:** el programa tiene que ser capaz de mostrar información sobre los distintos modelos que se pueden escoger.
 - **RF-6.2 Mostrar información sobre conjuntos de datos:** el programa tiene que ser capaz de mostrar información sobre los distintos conjuntos de datos de prueba.

Requisitos no funcionales

- **RNF-1 Usabilidad:** la interfaz gráfica tiene que ser intuitiva y fácil de usar.
- **RNF-2 Soporte:** el programa tiene que dar soporte a versiones iguales o mayores a Python 3.
- **RNF-3 Localización:** el programa tiene que estar preparado para soportar varios idiomas.

B.4. Especificación de requisitos

En esta sección se mostrará el diagrama de casos de uso y se desarrollará cada uno de ellos.

Diagrama de casos de uso

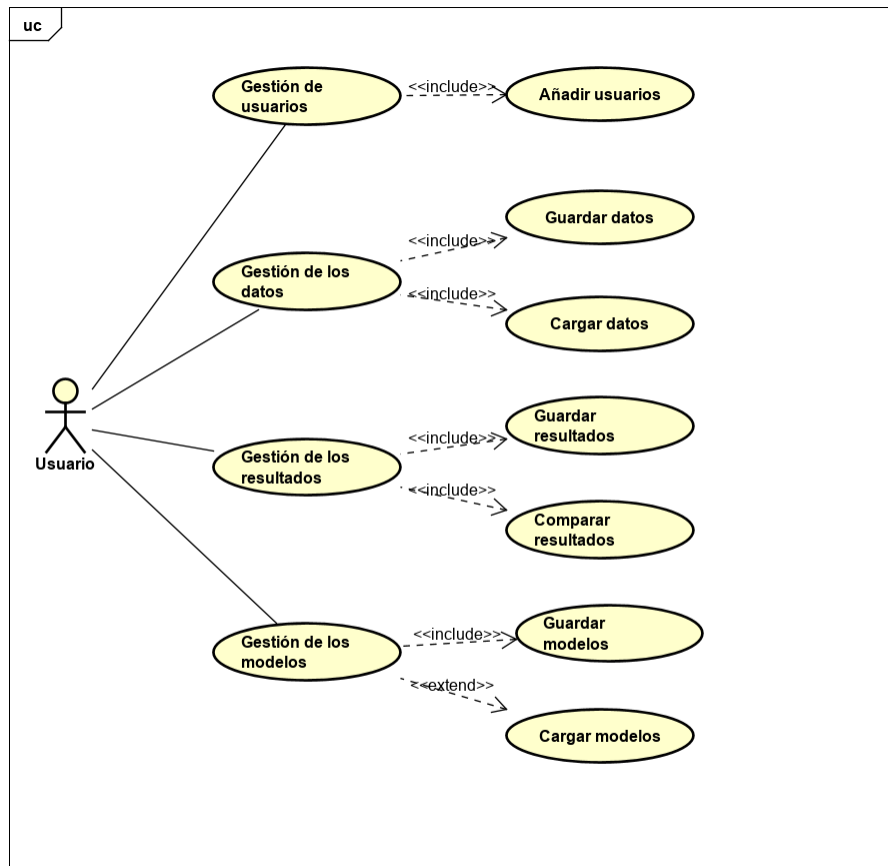


Figura B.1: Diagrama de casos de uso

Actores

Con la aplicación solo interactuará un actor, el usuario que esté probando la aplicación en un momento determinado.

Casos de uso

A continuación, se desarrollará cada caso de uso:

Apéndice C

Especificación de diseño

C.1. Introducción

En este apéndice se explica cómo están conformados los datos que utilizan las librerías usadas en el proyecto, así como la forma en la que está estructurado el mismo.

C.2. Diseño de datos

Todos los datos que he utilizado a lo largo del proyecto están en formato .csv. Lo más normal es que para cada conjunto de datos tenga los siguientes archivos:

- *ratings.csv*
- *users.csv*
- *items.csv*

La estructura de estos archivos suele ser: *idUser*, *idItem*, *rating*, *timestamp* para *ratings.csv*, *idUser*, *name*, *feature1*, ..., *featureN* para *users.csv* y *idItem*, *name*, *feature1*, ..., *featureN* para *items.csv*.

Para poder trabajar con los datos primero los paso a *DataFrames* de *pandas* [3].

Datos con LightFM

Una vez obtenidos los *DataFrames* para cada *.csv*, necesito convertirlos a *Dataset* de *LightFM* [2] para poder trabajar con ellos.

Esta clase se encarga de convertir los datos almacenados en los *DataFrames* en *matrices COO* y *matrices CSR*.

C.3. Diseño procedimental

C.4. Diseño arquitectónico

Para la realización de este proyecto se ha seguido el patrón arquitectónico MVC (*Modelo Vista Controlador*). El objetivo de este patrón es dividir el código en función de su propósito. Sus partes son:

- *Modelo*: el acceso a los datos. Se corresponde con las clases de Entrada y Salida, que leen los datos para dárselo al sistema de recomendación y guardan los resultados.
- *Vista*: la visualización de los datos. Se corresponde con las clases de Interfaz, que muestran la información solicitada.
- *Controlador*: la manipulación de los datos. Se corresponde con las clases de Sistema, que crean los sistemas de recomendación gracias a los datos proporcionados por las clases de Entrada.

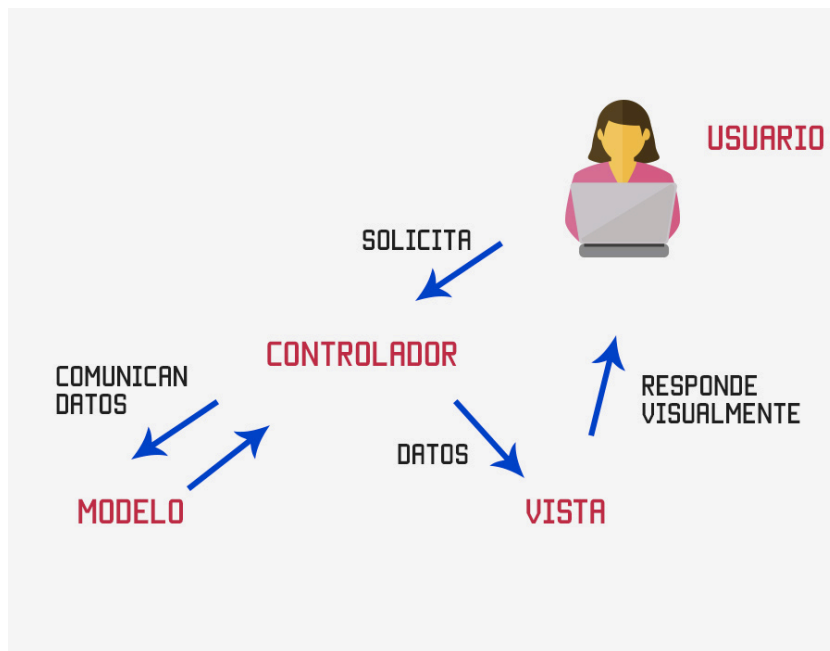


Figura C.1: Esquema del patrón MVC [1]

La estructura del proyecto siguiendo este patrón quedaría de la siguiente forma:

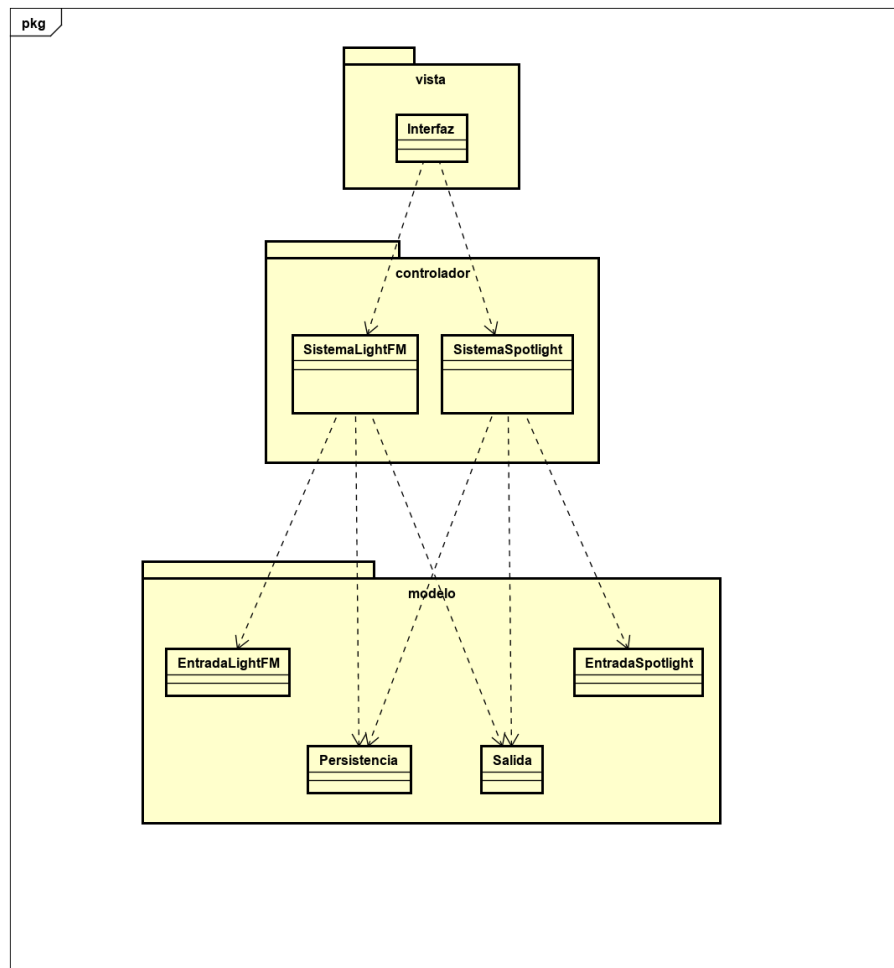
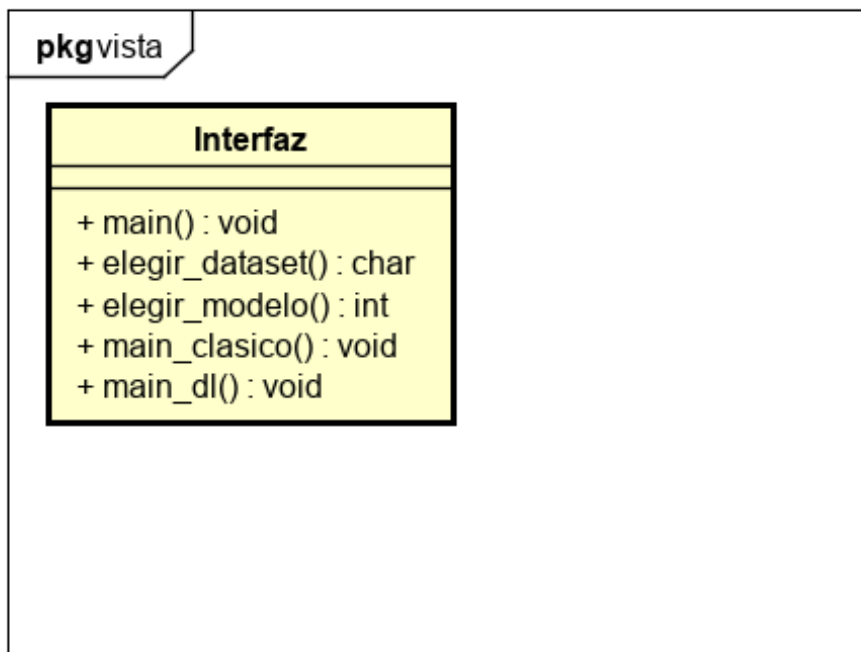
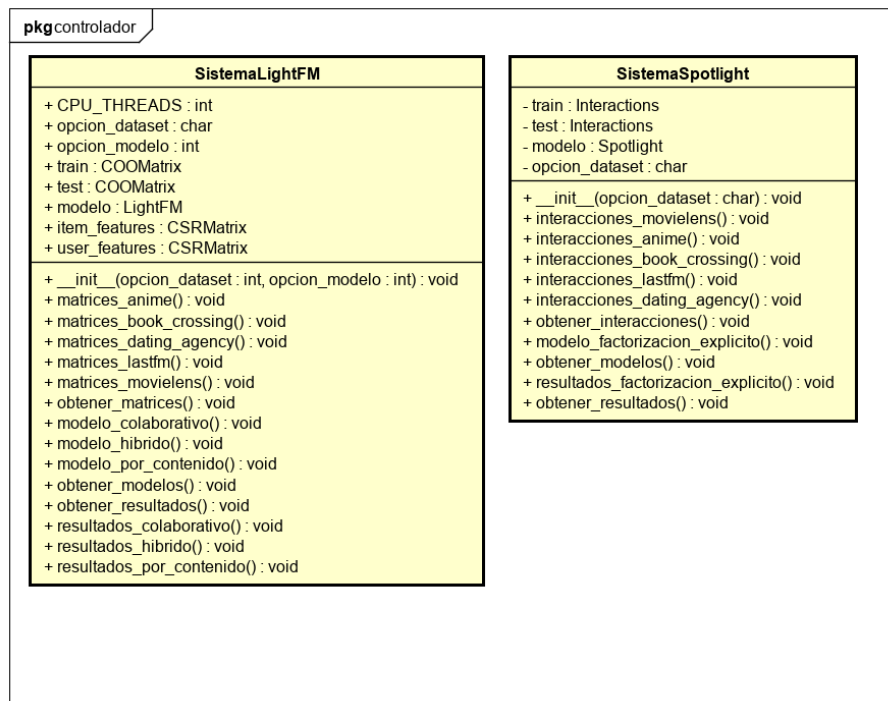
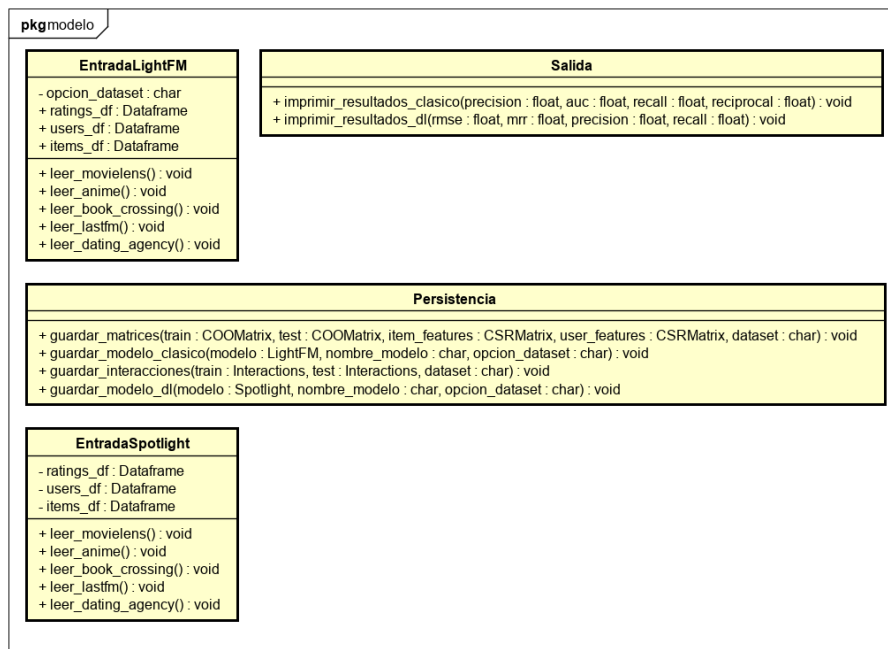


Figura C.2: Diagrama UML del proyecto

Por separado, los paquetes contienen:

Figura C.3: Diagrama UML del paquete *vista*

Figura C.4: Diagrama UML del paquete *controlador*

Figura C.5: Diagrama UML del paquete *modelo*

Diseño con LightFM

Para la parte del modelo clásico, se tienen los siguientes archivos: En la clase `EntradaLightFM` se tiene:

- `leer_x`: estos métodos recogen los datos de los *.csv* para cada conjunto de datos. Estos métodos son utilizados por los métodos de obtención de matrices de la clase `SistemaLightFM`.

En el archivo `Salida` se tiene:

- `imprimir_resultados_clasico`: este método imprime las métricas del modelo clásico escogido.

En el archivo `Persistencia` se tiene:

- `guardar_matrices`: este método guarda en un archivo *pickle* las matrices que *LightFM* necesita para obtener los modelos.

- `guardar_modelo`: este método guarda el modelo obtenido por *LightFM* en un archivo *pickle*.

En el archivo `Interfaz` se tiene:

- `elegir_dataset`: este método muestra un menú mediante el cual elegimos un conjunto de datos que utilizar.
- `elegir_modelo`: este método muestra un menú mediante el cual elegimos un modelo concreto a crear.
- `main_clasico`: programa principal si el modelo escogido es *LightFM*.

En la clase `SistemaLightFM` tenemos:

- `matrices_x`: estos métodos crean las matrices necesarias para cada conjunto de datos.
- `modelo_x`: estos métodos crean los distintos modelos de recomendación.
- `resultados_x`: estos métodos obtienen los resultados para cada modelo de recomendación.

Diseño con Spotlight

Para la parte del modelo basado en aprendizaje profundo, se tienen los siguientes archivos: En la clase `EntradaSpotlight` se tiene:

- `leer_x`: estos métodos recogen los datos de los *.csv* para cada conjunto de datos. Estos métodos son utilizados por los métodos de obtención de interacciones de la clase `SistemaSpotlight`.

En el archivo `Salida` se tiene:

- `imprimir_resultados_dl`: este método imprime las métricas del modelo basado en aprendizaje profundo escogido.

En el archivo `Persistencia` se tiene:

- `guardar_interacciones`: este método guarda en un archivo *pickle* las interacciones que *Spotlight* necesita para obtener los modelos.
- `guardar_modelo_dl`: este método guarda el modelo obtenido por *Spotlight*.

En el archivo `Interfaz` se tiene:

- `main_dl`: programa principal si el modelo escogido es *Spotlight*.

En la clase `SistemaSpotlight` tenemos:

- `interacciones_x`: estos métodos crean las interacciones necesarias para cada conjunto de datos.
- `modelo_x`: estos métodos crean los distintos modelos de recomendación.
- `resultados_x`: estos métodos obtienen los resultados para cada modelo de recomendación.

Apéndice D

Documentación técnica de programación

- D.1. Introducción
- D.2. Estructura de directorios
- D.3. Manual del programador
- D.4. Compilación, instalación y ejecución del proyecto
- D.5. Pruebas del sistema

Apéndice E

Documentación de usuario

- E.1. Introducción
- E.2. Requisitos de usuarios
- E.3. Instalación
- E.4. Manual del usuario

Bibliografía

- [1] Uriel Hernandez. Mvc (model, view, controller) explicado.
- [2] Maciej Kula. *Dataset construction*.
- [3] Pandas. *DataFrame*.