SISTEMA DE RECOMENDACIÓN COLABORATIVA SOBRE EL DATASET MOVIELENS

Pablo Hernández Jiménez[1*†], Eric Ríos Hamilton[1*†] y Enmanuel Vegas Acosta[1*† 1]

[1] *Sistemas Inteligentes, Escuela Superior de Ingeniería y Tecnología, Universidad de La Laguna, Canarias, España*

Entregado: November 18, 2025

Última Actualización: November 18, 2025

Resumen Este trabajo presenta el diseño e implementación de un sistema de recomendación colaborativa utilizando el dataset MovieLens, con el objetivo de predecir las preferencias de los usuarios y mejorar la personalización en la selección de películas. Se comparan enfoques de **filtrado colaborativo basado en memoria** (vecindarios usuario-usuario y ítem-ítem con similitudes cosine y Pearson) y modelos basados en factores latentes mediante descomposición matricial (SVD/SVD++). El estudio aborda el **preprocesamiento de datos** (filtrado de usuarios/ítems esparsos, normalización de calificaciones y particionado temporal) y evalúa el desempeño con **métricas de error** y de **ranking**, además de un análisis de cobertura y diversidad.

1. SISTEMAS DE RECOMENDACIÓN

Los sistemas de recomendación son herramientas que ayudan a las personas a encontrar contenido relevante dentro de un conjunto enorme de opciones[4]. En los sistemas actuales, esta tarea se apoya de forma intensiva en métodos de inteligencia artificial[5] y de aprendizaje automático[6], que permiten aprender del comportamiento de los usuarios, detectar patrones en sus interacciones y ofrecer recomendaciones cada vez más precisas.

De forma muy simplificada, pueden distinguirse dos grandes enfoques:

- **Métodos colaborativos**
- **Métodos basados en contenido**

1. 1. MÉTODOS COLABORATIVOS

Los sistemas de filtrado colaborativo se basan en las valoraciones y acciones de las personas usuarias[7]. La idea es sencilla: si dos usuarios han mostrado interés por elementos similares en el pasado, es probable que compartan gustos también en el futuro. El sistema compara las valoraciones de un usuario con las de otros usuarios y, a partir de esas similitudes, recomienda elementos que aún no ha visto, pero que han gustado a personas con preferencias parecidas.

[*] Autor(a/es) de correspondencia: Pablo Hernández Jiménez «alu0101495934@ull.edu.es», Eric Ríos Hamilton «alu0101549835@ull.edu.es» y Enmanuel Vegas Acosta «alu0101281698@ull.edu.es» .

[†] Pablo Hernández Jiménez, Eric Ríos Hamilton y Enmanuel Vegas Acosta contribuyeron de manera equivalente a este trabajo.

[4] Una introducción general puede encontrarse en Recommender system (Wikipedia, en inglés) .

[5] Artificial intelligence (Wikipedia, en inglés) .

[6] Machine learning (Wikipedia, en inglés) .

[7] Véase Collaborative filtering (Wikipedia, en inglés) .

1. 2. MÉTODOS BASADOS EN CONTENIDO

Por su parte, **los sistemas basados en contenido** se fijan en las características de los propios ítems que se quieren recomendar[8]. En el caso de una película, por ejemplo, el sistema puede considerar el género, la duración, el director, el reparto o el país de producción.

Con esta información construye un **perfil de usuario** que resume qué tipos de contenido le gustan. Para ello analiza qué elementos ha valorado positivamente o con cuáles ha interactuado más. Después, recomienda nuevos contenidos que comparten rasgos importantes con aquellos que ya han resultado de su agrado.

1. 3. ENFOQUES HÍBRIDOS

En la práctica, la mayoría de los sistemas de recomendación actuales son **enfoques híbridos** que combinan métodos colaborativos y basados en contenido[9]. Al integrar las fortalezas de ambos enfoques, estos sistemas:

- ofrecen recomendaciones más robustas,
- reducen las limitaciones de cada método por separado y
- proporcionan una experiencia más satisfactoria y personalizada para las personas usuarias.

1. 4. DEEP LEARNING EN SISTEMAS DE RECOMENDACIÓN

En los últimos años, el uso de **deep learning** ha transformado en gran medida el diseño de sistemas de recomendación[10]. Las redes neuronales profundas permiten trabajar con grandes volúmenes de datos y capturar patrones complejos que resultan difíciles de modelar con técnicas más tradicionales.

En este contexto, el deep learning se utiliza de varias formas:

- Para aprender representaciones (o **embeddings**) de usuarios e ítems que resumen sus características en vectores numéricos de baja dimensión.
- Para combinar información muy diversa: valoraciones explícitas, historial de clics, texto (por ejemplo, descripciones o reseñas), imágenes o incluso audio y vídeo.
- Para modelar la evolución temporal de las preferencias de los usuarios y adaptarse mejor a cambios en sus intereses.

Un ejemplo habitual es el uso de redes neuronales para extender el filtrado colaborativo clásico: en lugar de trabajar solo con una matriz de valoraciones, el modelo neuronal aprende representaciones latentes más ricas a partir de múltiples fuentes de información. Esto permite mejorar la calidad de las recomendaciones, especialmente en escenarios con muchos ítems, datos ruidosos o preferencias cambiantes.

Aunque estos modelos suelen ser más costosos de entrenar y de explicar, han demostrado ser especialmente eficaces en entornos de gran escala, como plataformas de vídeo, música o comercio electrónico, donde incluso pequeñas mejoras en la precisión de las recomendaciones tienen un impacto significativo en la experiencia de las personas usuarias.

2. ENTORNO DE DESARROLLO

La parte de codificación de este proyecto se organizará en torno a la herramienta `uv`[11], que se empleará tanto para la gestión de dependencias como para la creación y el aislamiento de los distintos entornos de ejecución. Esto permitirá definir de forma reproducible las librerías necesarias en cada fase del proyecto (prototipado, ETL, modelado y backend) y simplificar la instalación y actualización del entorno en distintas máquinas de trabajo.

El desarrollo se llevará a cabo utilizando Visual Studio Code[12] como entorno principal de edición y depuración. Además, se hará uso de librerías del ecosistema científico de Python para la carga, limpieza, exploración y transformación del conjunto de datos, así como para la construcción y evaluación de los modelos necesarios para alcanzar el objetivo final del trabajo.

En una primera fase, la aplicación se implementará y ejecutará en un entorno de línea de comandos. Las distintas funcionalidades se prototiparán y validarán previamente en notebooks de **marimo**[13], lo que permitirá iterar de manera rápida sobre las ideas, realizar pruebas controladas y documentar de forma más clara el flujo de trabajo y los resultados intermedios.

[8] Una descripción accesible puede encontrarse en Content-based filtering (IBM, en inglés) .
[9] Introducción general a sistemas híbridos en Hybrid recommender systems (Wikipedia, en inglés) .
[10] Una visión general se presenta en Deep learning (Wikipedia, en inglés) .
[11] Información general en uv (GitHub).
[12] Visual Studio Code.
[13] marimo: reactive Python notebooks.

2. 1. ORQUESTACIÓN DE LA ETL Y DEL ENTRENAMIENTO CON DAGSTER

Para gestionar de forma estructurada los procesos de extracción, transformación y carga de datos (ETL), así como las tareas de entrenamiento y reentrenamiento de los modelos, se empleará **Dagster**[14] como herramienta de orquestación. Mediante la definición de **jobs** y **ops** (o **assets**), Dagster permitirá describir de manera declarativa el flujo completo de trabajo: desde la ingesta de los datos brutos hasta la generación de modelos listos para ser consumidos por la aplicación.

El uso de Dagster aporta varias ventajas:

- facilita la separación clara entre las distintas etapas de la ETL y del ciclo de vida del modelo,
- permite monitorizar la ejecución de cada paso (por ejemplo, la calidad de los datos, la duración de las tareas o posibles errores), y
- hace posible reejecutar solo aquellas partes del flujo que lo necesiten cuando cambien los datos o la configuración.

De este modo, la canalización de datos y el proceso de entrenamiento se integran en un sistema reproducible, trazable y más sencillo de mantener a medida que el proyecto crece.

2. 2. DESARROLLO DEL BACKEND

Una vez que la implementación alcance un grado suficiente de estabilidad y madurez, se procederá a desarrollar un backend que exponga la lógica de negocio a través de una API. Para ello se utilizará principalmente FastAPI[15], por su buen rendimiento, su sintaxis declarativa y su buena integración con herramientas de validación de datos y documentación automática. La validación y el tipado de los datos de entrada y salida se gestionarán con Pydantic[16], lo que permitirá definir modelos de datos claros, consistentes y fácilmente reutilizables dentro de la aplicación.

Este enfoque incremental —desde prototipos en notebooks y ejecución por línea de comandos, pasando por la orquestación con Dagster, hasta un backend estructurado gestionado con uv— facilita la detección temprana de errores, mejora la trazabilidad de las decisiones técnicas y sienta una base sólida para futuras ampliaciones o integraciones del sistema.

2. 3. USO DE IA

En este proyecto se permitirá la utilización de herramientas de IA generativa como Copilot, que está incluido en el editor de texto escogido. Sin embargo, este uso estará limitado mayoritariamente a la consulta de dudas puntuales y otras funciones de autocompletado, de tal manera que se favorezca el aprendizaje de las técnicas aplicadas en la asignatura.

3. USO DE LA LIBRERÍA SURPRISE

Antes de construir el sistema de recomendación colaborativo con la librería **Surprise**[17], se realizará un preprocesado del conjunto de datos MovieLens[18]. Este preprocesado incluirá, entre otras tareas, la limpieza de registros incompletos, el filtrado de usuarios o ítems con muy pocas interacciones y la transformación de los datos al formato requerido por Surprise para su posterior uso en los algoritmos de predicción.

Surprise ofrece una caja de herramientas focalizada en la construcción y mejora de sistemas recomendadores colaborativos. En concreto, proporciona un control detallado sobre los datos, una gama amplia de algoritmos y métricas para calificar y evaluar los resultados, garantizando así un marco de trabajo óptimo y confiable para este tipo de sistemas.

En este proyecto emplearemos los recursos que ofrece Surprise para realizar el módulo recomendador colaborativo.

3. 1. ALGORITMOS DE SURPRISE

En las fórmulas, r_{ui} denota la valoración real del usuario u sobre el ítem i, y $\hat{r}_{ui}$ la predicción del modelo.

[14] Dagster: data orchestrator.
[15] FastAPI framework.
[16] Pydantic.
[17] Official documentation: scikit-surprise (Surprise) .
[18] Classical recommendation dataset: MovieLens .

3. 1. 1. random_pred.NormalPredictor

NormalPredictor predice una valoración basándose en la distribución del conjunto de entrenamiento, asumiendo que esta es normal. Se estima una media μ y una desviación típica σ y se modela:

$$r_{ui} \sim N(\mu, \sigma^2)$$

Cada valoración se genera muestreando de manera independiente de esta distribución.

3. 1. 2. baseline_only.BaselineOnly

BaselineOnly predice la estimación de referencia (**baseline**) para un usuario y un ítem dados mediante:

$$\hat{r}_{ui} = \mu + b_u + b_i$$

donde:

- μ es la media global de todas las valoraciones,
- b_u es el sesgo del usuario u,
- b_i es el sesgo del ítem i.

Si el usuario u o el ítem i son desconocidos, sus sesgos se toman como cero.

3. 1. 3. knns.KNNBasic

KNNBasic es la versión básica de **k-Nearest Neighbours**. Permite calcular similitud entre usuarios o entre ítems según el parámetro user_based de sim_options. La predicción típica puede escribirse como:

$$\hat{r}_{ui} = \frac{\sum_{v \in N_{k(u,i)}} s(u,v) * r_{vi}}{\sum_{v \in N_{k(u,i)}} |s(u,v)|}$$

donde:

- $N_{k(u,i)}$ es el conjunto de los k vecinos más similares que han valorado el ítem i,
- $s(u,v)$ es la similitud entre usuarios (o ítems, según el modo),
- r_{vi} es la valoración del vecino v sobre el ítem i.

Parámetros relevantes:

- k (int): número máximo de vecinos a tener en cuenta,
- min_k (int): número mínimo de vecinos,
- sim_options (dict): opciones para la medida de similitud,
- verbose (bool): si se debe imprimir una traza.

3. 1. 4. knns.KNNWithMeans

KNNWithMeans es igual que KNN normal, pero teniendo en cuenta las valoraciones medias de cada usuario. La fórmula se puede expresar como:

$$\hat{r}_{ui} = |(r)_u + \frac{\sum_{v \in N_{k(u,i)}} s(u,v) * (r_{vi} - |(r)_v)}{\sum_{v \in N_{k(u,i)}} |s(u,v)|}$$

donde $|(r)_u$ y $|(r)_v$ son las medias de valoración de los usuarios u y v.

3. 1. 5. knns.KNNWithZScore

KNNWithZScore es igual que KNN normal, pero teniendo en cuenta la **z-score** de cada usuario. Sea μ_u la media del usuario u y σ_u su desviación estándar. Entonces:

$$\hat{r}_{ui} = \mu_u + \sigma_u * \frac{\sum_{v \in N_{k(u,i)}} s(u,v) * \left(\frac{r_{vi} - \mu_v}{\sigma_v}\right)}{\sum_{v \in N_{k(u,i)}} |s(u,v)|}$$

Esto normaliza las valoraciones de cada usuario antes de combinarlas.

3. 1. 6. knns.KNNBaseline

KNNBaseline es igual que KNN normal, pero teniendo en cuenta la estimación de referencia (**baseline**) como en BaselineOnly. La predicción puede escribirse como:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{v \in N_{k(u,i)}} s(u,v) * (r_{vi} - b_{vi})}{\sum_{v \in N_{k(u,i)}} |s(u,v)|}$$

donde:

- $b_{ui} = \mu + b_u + b_i$,
- $b_{vi} = \mu + b_v + b_i$.

3. 1. 7. matrix_factorization.SVD

SVD calcula la predicción con:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T * p_u$$

donde:

- μ es la media global,
- b_u y b_i son sesgos de usuario e ítem respectivamente,
- p_u es el vector de factores latentes del usuario u,
- q_i es el vector de factores latentes del ítem i.

Si el usuario o el ítem son nulos, sus sesgos y factores se consideran nulos. Los parámetros se ajustan minimizando un error cuadrático medio regularizado sobre el conjunto de entrenamiento.

3. 1. 8. matrix_factorization.SVDpp

SVDpp es una extensión de SVD que incorpora la información de valoraciones implícitas (por ejemplo, que un usuario haya valorado un ítem, independientemente del valor). La fórmula habitual es:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T * \left(p_u + \frac{1}{\sqrt{|N(u)|}} * \sum_{j \in N(u)} y_j\right)$$

donde:

- $N(u)$ es el conjunto de ítems que el usuario u ha valorado implícitamente,
- y_j son factores implícitos asociados a cada ítem j.

Si el usuario o el ítem es desconocido, sus sesgos y factores implícitos se consideran cero.

3. 1. 9. matrix_factorization.NMF

NMF es similar a SVD, pero basado en **Non-negative Matrix Factorization**. En la versión base, sin sesgos, la fórmula de predicción es:

$$\hat{r}_{ui} = q_i^T * p_u$$

sujeta a la restricción de no negatividad:

$$p_u \geq 0, \quad q_i \geq 0$$

También existe una versión que incorpora sesgos de forma análoga a SVD.

3. 1. 10. slope_one.SlopeOne

SlopeOne es un algoritmo sencillo basado en diferencias promedio entre ítems. Una formulación típica es:

$$\hat{r}_{ui} = \frac{\sum_{j \in R_{i(u)}} (r_{uj} + \text{dev}_{ij})}{|R_{i(u)}|}$$

donde:

- $R_{i(u)}$ es el conjunto de ítems j valorados por el usuario u y que se usan para estimar el ítem i,
- dev_{ij} es la diferencia promedio entre los ítems i y j:

$$\text{dev}_{ij} = \frac{\sum_{u \in U_{ij}} (r_{ui} - r_{uj})}{|U_{ij}|}$$

y U_{ij} es el conjunto de usuarios que han valorado tanto i como j.

3. 1. 11. co_clustering.CoClustering

CoClustering es un algoritmo basado en el **co-clustering** de usuarios e ítems. Los usuarios se agrupan en n_{cltr_u} clústeres y los ítems en n_{cltr_i}, formando co-clusters. Una forma habitual de escribir la predicción es:

$$\hat{r}_{ui} = |(C)_{c(u),c(i)} + \left(\mu_u - |(C)_{c(u),.}\right) + \left(\mu_i - |(C)_{.,c(i)}\right)$$

donde:

- $c(u)$ es el clúster del usuario u,
- $c(i)$ es el clúster del ítem i,
- $|(C)_{c(u),c(i)}$ es la media del co-cluster correspondiente,
- μ_u es la media de las valoraciones del usuario u,
- μ_i es la media de las valoraciones del ítem i.

Si el usuario es desconocido, la predicción se reduce a μ_i; si el ítem es desconocido, a μ_u; y si ambos lo son, a la media global μ.

3. 2. MÉTRICAS DE SURPRISE

La librería Surprise también proporciona herramientas para evaluar la calidad de las predicciones. En el módulo accuracy se implementan métodos para computar métricas de exactitud dado un conjunto de predicciones.

Sea r_i la valoración real y $\hat{r}_i$ la predicción para cada observación $i = 1, ..., n$.

3. 2. 1. Mean Squared Error (MSE)

El **Mean Squared Error** (MSE) se define como:

$$\text{MSE} = \frac{1}{n} * \sum_{i=1}^{n} (r_i - \hat{r}_i)^2 \tag{1}$$

Penaliza más los errores grandes que los pequeños debido al cuadrado. Cuanto menor sea el MSE, más precisa es la predicción.

3. 2. 2. Root Mean Squared Error (RMSE)

El **Root Mean Squared Error** (RMSE) es la raíz cuadrada del MSE:

$$\text{RMSE} = \sqrt{\frac{1}{n} * \sum_{i=1}^{n} (r_i - \hat{r}_i)^2} \tag{2}$$

Es la métrica más común en sistemas de recomendación. La raíz permite expresar el error en la misma escala que las valoraciones (por ejemplo, 1-5 estrellas).

3. 2. 3. Mean Absolute Error (MAE)

El **Mean Absolute Error** (MAE) mide el promedio de las diferencias absolutas:

$$\text{MAE} = \frac{1}{n} * \sum_{i=1}^{n} |r_i - \hat{r}_i| \tag{3}$$

Es más robusto frente a **outliers** que el MSE, ya que no penaliza tanto los errores grandes. Cuanto menor sea el MAE, más exacto es el modelo.

3. 2. 4. Fraction of Concordant Pairs (FCP)

La **Fraction of Concordant Pairs** (FCP) mide la capacidad del sistema para ordenar correctamente los ítems según las preferencias del usuario. Se consideran pares (i, j) tales que un usuario ha puntuado $r_i > r_j$ y se verifica si el modelo también predice $\hat{r}_i > \hat{r}_j$.

Sea C el número de pares concordantes y T el número total de pares comparables. Entonces:

$$\text{FCP} = \frac{C}{T} \tag{4}$$

Un valor de FCP cercano a 1 indica que el modelo respeta bien el orden de las preferencias.

3. 3. CRITERIO DE SELECCIÓN

En base a los resultados arrojados por las métricas, se seleccionará el único modelo o combinación de modelos que proporcione el mejor resultado, utilizando distintos criterios y algoritmos para dicha selección.

En primer lugar, para cada algoritmo considerado de Surprise se efectuará una validación cruzada **k-fold** estratificada por usuario, utilizando como métrica principal el RMSE por su interpretabilidad en la escala de las valoraciones y su uso extendido en la literatura.

De forma complementaria, se monitorizarán MAE y MSE para detectar sensibilidad a **outliers** y errores grandes, así como FCP para valorar la capacidad de ordenar correctamente las preferencias.

Además, se evaluarán métricas de **ranking** sobre listas Top-N por usuario, concretamente NDCG@k (por ejemplo, $k \in \{5, 10\}$) para capturar la calidad del orden ponderado por posición, MRR (**Mean Reciprocal Rank**) y RecipRank (**Reciprocal Rank** por usuario), con el fin de medir la posición del primer acierto.

La sintonía de hiperparámetros se llevará a cabo con GridSearchCV sobre el conjunto de entrenamiento, optimizando RMSE y reportando la desviación estándar por pliegue para estimar la estabilidad. Cuando el objetivo sea principalmente de **ranking**, se contrastarán también las configuraciones finalistas en términos de NDCG@k y MRR.

De esta manera, el criterio de selección priorizará el modelo con menor RMSE medio en validación y variabilidad reducida; en caso de resultados próximos, se emplearán criterios de desempate atendiendo a:

- mayor NDCG@k y MRR, junto a FCP elevada y MAE comparable o inferior;
- coste computacional (tiempo de entrenamiento e inferencia) y escalabilidad;
- robustez ante la dispersión de los datos (cobertura de predicciones) y comportamiento en escenarios de **cold start**.

Cuando dos modelos muestren comportamientos complementarios, se considerará un ensamblado sencillo mediante promedio ponderado de puntuaciones normalizadas, adoptándolo únicamente si aporta una mejora

estadísticamente significativa en RMSE y en las métricas de **ranking** (NDCG@k, MRR / RecipRank) sin comprometer la latencia.

La decisión final se validará en un conjunto de pruebas mantenido al margen del ajuste, aplicando el mismo procedimiento Top-N y documentando los hiperparámetros definitivos, la semilla aleatoria y todas las métricas reportadas.