ESCUELA POLITÉCNICA
SUPERIOR DE CÓRDOBA
Universidad de Córdoba

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Informática

Aplicación de sistemas de recomendación en entornos educativos

Autor: Francisco de Paula Algar Muñoz

Directores: Dña. Amelia Zafra Gómez
D. Cristóbal Romero Morales

junio, 2025

UNIVERSIDAD
D
CÓRDOBA

Agradecimientos

Quiero expresar mi más profundo agradecimiento a todas las personas que han hecho posible la culminación de este Trabajo de Fin de Grado, pues sin ellos, este trabajo no habría podido terminarse.

En primer lugar, quiero agradecer a mi familia. Su apoyo incondicional, ánimo y cariño han sido fundamentales a lo largo de este camino. Su confianza en mí ha sido una fuente constante de motivación. Gracias por estar siempre ahí, por comprender mis ausencias y por celebrar conmigo cada pequeño logro.

A mis tutores, Amelia Zafra Gómez y Cristóbal Romero Morales, por su inestimable guía, paciencia y dedicación. Su experiencia, compromiso y disposición para orientarme en cada etapa del proyecto han sido clave para el desarrollo y la culminación de este trabajo. Sus aportaciones y comentarios han enriquecido enormemente el resultado final.

Finalmente, quiero dar las gracias a mis amigos, con quienes he compartido innumerables horas de estudio, dudas, ideas y muchas risas. Su apoyo constante, tanto dentro como fuera del aula, ha sido fundamental para superar los momentos de dificultad y disfrutar de esta etapa universitaria. De manera especial, agradezco profundamente a mi pareja por su apoyo incondicional, por creer siempre en mí y por su constante aliento durante todo este proceso. Gracias a todos por su amistad, por motivarme a seguir adelante y por hacer este camino académico mucho más llevadero.

A todos vosotros, gracias de corazón.

Aplicación de sistemas de recomendación en entornos educativos

Resumen

Los sistemas de recomendación han demostrado ser herramientas clave para mejorar la personalización del aprendizaje en entornos educativos digitales. Este trabajo de fin de grado tiene como objetivo desarrollar y evaluar un sistema de recomendación adaptado a plataformas de *e-learning*, utilizando un conjunto de datos representativo del ámbito educativo. A través del análisis de variables como intereses, historial de interacciones y metadatos de cursos, se busca ofrecer recomendaciones personalizadas que mejoren la motivación y el rendimiento del alumnado. El sistema propuesto será comparado experimentalmente con modelos existentes para validar su eficacia y aplicabilidad en contextos reales de enseñanza virtual.

Palabras clave: Sistemas de recomendación, E-learning, Aprendizaje personalizado, Aprendizaje automático, Aprendizaje profundo, Análisis de datos.

Application of recommendation systems in educational settings

Abstract

Recommender systems have proven to be key tools for enhancing personalized learning in digital educational environments. This thesis aims to develop and evaluate a recommendation system tailored to *e-learning* platforms, using a representative dataset from the educational domain. By analyzing variables such as student interests, interaction history, and course metadata, the goal is to provide personalized recommendations that improve student motivation and performance. The proposed system will be experimentally compared with existing models to validate its effectiveness and applicability in real virtual learning contexts.

Keywords: Recommender systems, E-learning, Personalized learning, Machine learning, Deep learning, Data analysis.

Índice

Índice de figuras

Índice de tablas

Acrónimos

ADAM *Adaptive Moment Estimation*. 61

AE *Auto-Encoders*. 10

ALS Mínimos Cuadrados Alternativos (*Alternating Least Squares*). 60

API *Applicatin Programming Interface*. 46

AP *Average Precision*. 18

ASARM Asociación con Soporte Adaptativo. 11

CBF Basados en Contenido. 5, 50, IV

CDD Diagrama de Diferencias Críticas. 38, 72–74, VII

CF Filtrado Colaborativo. 6, 50, IV

CNN Redes Neuronales Convolucionales (*Convolutional Neural Networks*). 10

CPU Unidades Central de Procesamiento. 64

CRISP-DM *Cross-Industry Standard Process for Data Mining*. 23, VII

CSV Valores Separados por Comas. 25, 28, 68, 95, 96, 99, 100

CTR Tasa de Clics (*Click-Through Rate*). 33

CV *Cross Validation*. 35

DA Análisis de datos (*Data Bases*). 27, 29

DB Bases de Datos (*Data Bases*). 27, 29

DL Aprendizaje Profundo. 9, 37, 50, 53

DS Ciencia de datos (*Data Science*). 27, 29

GPU Unidades de Procesamiento Gráfico. 49, 64

HR *Hit Rate*. 18, 66, 67, 69–71, 90, VIII

KFCV *k-fold Cross Validation*. 35, 36, 63, VII

KNN *K*-Vecinos más Cercanos (*K-Nearest Neighbors*). 8, 40–42, 48, 60

LLM Largos Modelos del Lenguaje (*Large Language Models*). 10

LOOCV *Leave One Out Cross Validation*. 36, VII

MAP *Mean Average Precision*. 17, 66–69, 71, 88, VIII

MARS *Mandarine Academy Recommender System*. 25, 65–70, 72, 73, 85–93, 97, VII, VIII

MF *Matrix Factorization*. 42, 61

MLP Perceptrón Multicapa (*Multi-Layer Perceptron*). 10, 51, 55, 56, 58, 59

ML Aprendizaje Automático. 9, 37

MRR *Mean Reciprocal Rank*. 17, 66, 67, 89, VIII

MSD Diferencia Cuadrática Media (*Mean Squared Difference*). 60

MSE *Mean Squared Error*. 14, 60, 66, 67, 85, VIII

NDCG *Normalized Discounted Cumulative Gain*. 19, 66–69, 71, 87, VIII

NMF *Non-negative Matrix Factorization*. 43, 64

RAM Memoria de acceso aleatorio. 64

REST *Representational State Transfer*. 46

RMSE *Root Mean Squared Error*. 15, 66–69, 71, 86, VIII

RNN Redes Neuronales Recurrente (*Recurrent Neural Networks*). 10

ReLU *Rectified Linear Unit*. 55

SGD Descenso del Gradiente Estocástico *Stochastic Gradient Descent*. 43, 44, 68

SVD *Singular Value Decomposition*. 42–44, 48, 64

TFG Trabajo de Fin de Grado. 2, 20, 37, 46, 76, 77, 94

VRAM Memoria gráfica de acceso aleatorio. 64

1. Introducción

Los sistemas de recomendación constituyen herramientas tecnológicas avanzadas que permiten ofrecer contenido personalizado a los usuarios, y su aplicación se extiende a sectores tan variados como el entretenimiento [11], el comercio [16] y la salud [3]. En el ámbito educativo, estos sistemas han emergido como un recurso fundamental para adaptar la experiencia de aprendizaje a las necesidades individuales de cada estudiante. En el contexto del *e-learning*, dichos sistemas permiten ajustar los contenidos y recursos en función del perfil del alumno, considerando variables como intereses, nivel de conocimiento y objetivos de aprendizaje, lo que incide positivamente en la motivación, reduce las tasas de abandono y favorece un proceso de enseñanza-aprendizaje más efectivo y satisfactorio [31].

Además, los sistemas de recomendación ofrecen beneficios significativos para los docentes, al proporcionar una visión detallada del progreso y de las áreas de mejora de cada estudiante, lo que facilita la implementación de estrategias de enseñanza más flexibles y adaptadas. Sin embargo, la integración de soluciones de recomendación en entornos *e-learning* implica enfrentar desafíos técnicos y pedagógicos considerables. Es esencial que las recomendaciones se ajusten a los estilos y necesidades individuales, se gestionen de manera eficiente grandes volúmenes de datos y se asegure la precisión y relevancia de los contenidos sugeridos. A pesar de estas complejidades, existen estudios que demuestran que estos sistemas están transformando la educación en línea, al posibilitar experiencias de aprendizaje más accesibles, inclusivas y personalizadas [4].

El presente Trabajo de Fin de Grado (*TFG*) tiene como objetivo desarrollar un sistema de recomendación para plataformas de *e-learning* utilizando conjuntos de datos públicos representativos del entorno educativo, lo que permitirá evaluar su rendimiento en comparación con modelos previos. Dichos datos deberán abarcar información diversa, desde características de los estudiantes y su historial de interacciones hasta metadatos de los cursos, garantizando una evaluación integral

del sistema. Posteriormente, se realizará un estudio experimental comparativo con sistemas existentes, a fin de analizar y validar el desempeño del sistema desarrollado y obtener una línea base de rendimiento que pueda servir como referencia para evaluar nuevas propuestas adaptadas al entorno del *e-learning*, contribuyendo así a la mejora de la personalización y la efectividad en plataformas educativas digitales.

2. Estado del arte

Los sistemas de recomendación son herramientas informáticas esenciales en la era digital actual, desarrolladas para filtrar, organizar y presentar información relevante de manera personalizada a los usuarios. Utilizan algoritmos avanzados capaces de analizar tanto el comportamiento explícito como implícito de los usuarios, teniendo en cuenta sus preferencias, historial de interacciones y los atributos específicos de los ítems recomendados, como pueden ser cursos, recursos educativos, artículos científicos o contenidos multimedia. El objetivo principal es generar recomendaciones personalizadas que se alineen con los intereses, necesidades y objetivos particulares de cada individuo, facilitando la toma de decisiones y optimizando el proceso de descubrimiento de información.

En las últimas décadas, estos sistemas han evolucionado significativamente, pasando de simples mecanismos de filtrado estático a plataformas inteligentes que incorporan técnicas de aprendizaje automático y modelado del usuario [46]. No obstante, a pesar de los numerosos avances teóricos y prácticos, todavía se enfrenta a una importante limitación: la falta de estudios comparativos rigurosos que permitan evaluar con precisión las fortalezas y debilidades de los diferentes enfoques propuestos. Esta situación se ve agravada por la carencia de conjuntos de datos estandarizados y públicamente disponibles en el ámbito de la recomendación de cursos y recursos educativos, lo que dificulta la comparabilidad y reproducibilidad de los resultados obtenidos por distintos autores. Muchos trabajos recurren a datos propietarios o construidos ad hoc, lo que limita la validez externa de sus conclusiones y complica la replicación de los experimentos.

En esta sección se describen los principales enfoques utilizados en sistemas de recomendación, junto con las métricas más empleadas para su evaluación, para ello, seguiremos la taxonomía propuesta por Guruge et al. [13], que distingue cuatro grandes enfoques: filtrado basado en contenido, filtrado colaborativo, basados en conocimiento y enfoques híbridos. Cada uno de estos métodos presenta ventajas

específicas según el dominio de aplicación, la naturaleza de los datos disponibles y el grado de personalización deseado, lo que hace fundamental su análisis y comparación sistemática para el desarrollo de soluciones eficaces en contextos educativos y otros entornos.

2.1. Sistemas Basados en Contenido (*CBF*)

Estos sistemas recomiendan ítems al usuario en función de la descripción de dichos ítems y de aquellos que han sido de su interés previamente o que son similares a los que le han gustado anteriormente. La Figura 1 ilustra visualmente este enfoque. Por lo general, las preferencias del usuario se comparan con el contenido de los ítems para determinar su relevancia. Las características de interés del usuario pueden obtenerse de forma explícita, a través de cuestionarios, o de manera implícita, mediante el análisis de su comportamiento a lo largo del tiempo.

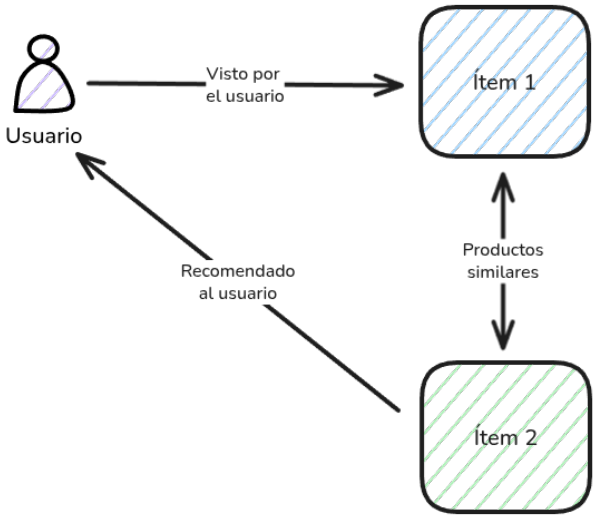


Figura 1: Sistemas basados en contenido

Sin embargo, al depender de las valoraciones propias del usuario para influir en futuras recomendaciones, un enfoque puramente basado en contenido presenta limitaciones significativas. Entre los inconvenientes se encuentran el análisis limitado del contenido, la dificultad para incorporar datos no textuales, la tendencia a

recomendar únicamente ítems muy similares a los ya conocidos (lo que lleva a la sobrespecialización) y la escasez de datos.

2.2. Sistemas Basados en Filtrado Colaborativo (*CF*)

La técnica de filtrado colaborativo es uno de los métodos más consolidados y utilizados en sistemas de recomendación. Se basa en la idea fundamental de que los usuarios que han mostrado comportamientos e intereses similares en el pasado tienden a compartir preferencias en el futuro. Para lograrlo, como se puede observar en la Figura 2, se recopilan y analizan las interacciones o valoraciones de los usuarios, organizándolas en una matriz usuario-ítem. Esta matriz contiene información detallada sobre los gustos y preferencias de los usuarios en una amplia variedad de ítems, como cursos, productos o servicios. A partir de este análisis, el sistema identifica patrones comunes entre los usuarios y genera recomendaciones personalizadas, anticipando las necesidades y preferencias individuales.

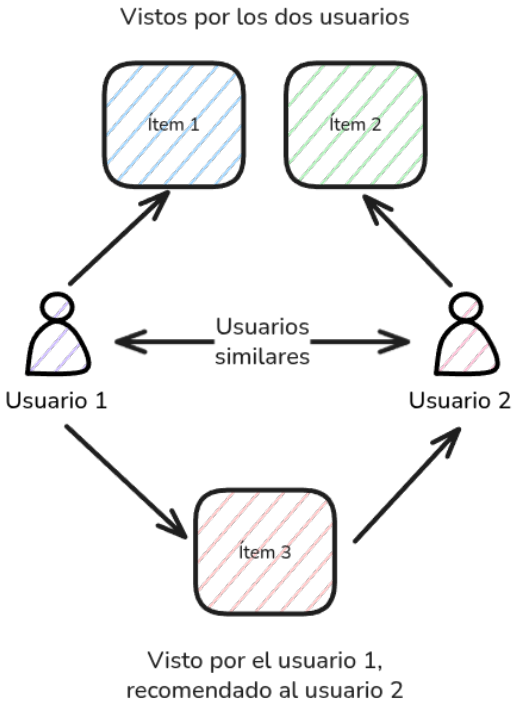


Figura 2: Sistemas basados en filtrado colaborativo

La matriz de interacción, representada en la Tabla 1 suele contener puntuaciones o valoraciones que permiten cuantificar la similitud entre usuarios o incluso entre los propios ítems. De este modo, se pueden aplicar diversas métricas para identificar patrones y determinar la cercanía entre los perfiles. Con base en esta información, el sistema es capaz de generar recomendaciones personalizadas, sugiriendo ítems que han sido valorados positivamente por usuarios con intereses semejantes.

	i_1	i_2	i_3	i_4
u_1	1	4	8	9
u_2	7	3	8	10
u_3	5	6	4	8
u_4	4	7	3	9

Tabla 1: Matriz usuario-ítem

A pesar de su efectividad, esta metodología presenta algunos inconvenientes como la escalabilidad, lo que puede hacer que el procesamiento de grandes volúmenes de datos resulte computacionalmente costoso, el denominado problema del arranque frío (*cold-start*), que dificulta la generación de recomendaciones para nuevos usuarios o ítems, ya que al inicio se dispone de poca información, o el problema de las ovejas grises (*gray sheeps*), que surge cuando ciertos usuarios tienen preferencias que no coinciden claramente con ningún grupo definido, lo que hace que las recomendaciones para ellos sean menos precisas.

Este tipo de sistemas de recomendación pueden agruparse en dos categorías generales: basados en memoria o basados en modelos.

2.2.1. Basados en memoria

Existen dos tipos de algoritmos dentro de esta categoría dependiendo de como se construya la matriz usuario-ítem: basados en usuario y basados en ítems.

Los algoritmos basados en usuarios recomiendan ítems al usuario activo aprovechando su historial de preferencias y las valoraciones proporcionadas por otros

usuarios con gustos similares. El proceso general se estructura en los siguientes pasos:

1. **Construcción de la matriz usuario-ítem**: como se muestra en la Tabla 1
2. **Cálculo de similitudes**: se utilizan diversas métricas estadísticas para comparar el comportamiento de los usuarios. Entre las más comunes se encuentran la Correlación de Pearson (*Pearson Correlation*), la similitud del vector coseno (*cosine similarity*) o la Correlación de Spearman *Spearman Correlation*.
3. **Selección de vecinos**: con las similitudes calculadas, se identifican los usuarios más afines al usuario activo. Para ello, se puede utilizar el algoritmo de K-vecinos más cercanos o *K-Vecinos más Cercanos (K-Nearest Neighbors) (KNN)* [20], que selecciona a los K usuarios con mayor similitud. Posteriormente, se combinan las puntuaciones de estos vecinos para generar recomendaciones personalizadas.

Por otro lado, en los sistemas basados en ítems se asume que dos ítems son similares si han recibido valoraciones parecidas por parte de distintos usuarios. El proceso de recomendación se divide en dos etapas:

1. **Identificación de ítems similares**: se comparan los ítems en función de las valoraciones de los usuarios que han interactuado con ambos. Dos ítems se consideran similares si un número considerable de usuarios los calificó de manera similar. Se emplean métricas como la similitud basada en correlación y la similitud ajustada-coseno para cuantificar esta similitud.
2. **Cálculo de predicciones**: una vez identificados los ítems similares, se predice la valoración que el usuario activo podría asignar al ítem objetivo. Esto se logra calculando un promedio ponderado de las calificaciones que el usuario ha otorgado a los ítems similares, ponderando cada calificación en función del grado de similitud.

2.2.2. Basados en modelos

El enfoque basado en modelos emplea métodos algorítmicos para generar recomendaciones. Estos algoritmos se han creado para superar ciertas limitaciones de los sistemas de recomendación comerciales que dependen en gran medida de los datos, como la velocidad en las predicciones y el riesgo de sobreajuste presente en los algoritmos basados en memoria. En lugar de trabajar directamente con enormes volúmenes de datos o valoraciones, estos algoritmos condensan dicha información en un modelo, lo que mejora notablemente la eficiencia predictiva. Podemos distinguir dos enfoques dentro de este tipo de sistemas:

1. **Aprendizaje Automático (*ML*) y Aprendizaje Profundo (*DL*)**: Numerosos estudios aplican modelos de *ML* y *DL* para desarrollar sistemas de recomendación. Entre los métodos tradicionales se destacan los algoritmos bayesianos, los árboles de decisión, las técnicas basadas en vecinos, las redes neuronales, los algoritmos de *clustering* y los métodos basados en descenso de gradiente. En particular, el algoritmo de *clustering K-means* se utiliza ampliamente en el ámbito de las recomendaciones de cursos, ya que permite agrupar a estudiantes con preferencias y comportamientos similares, facilitando la recomendación de objetos de aprendizaje afines mediante métricas de similitud previamente definidas.

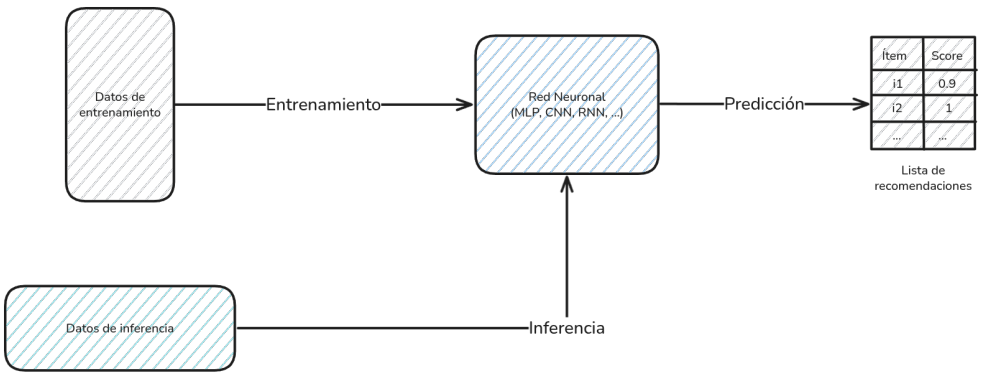


Figura 3: Sistemas basados en aprendizaje automático/profundo

Salau et al. [44] señalan que el auge del aprendizaje profundo ha impulsa-

do el uso de modelos basados en redes neuronales para mejorar los sistemas de recomendación. Entre los más utilizados se encuentran enfoques clásicos como el Perceptrón Multicapa (*Multi-Layer Perceptron*) (*MLP*) [12, 18] y los *Auto-Encoders* (*AE*) [9], que permiten extraer representaciones de los datos de forma eficiente. También destacan las Redes Neuronales Convolucionales (*Convolutional Neural Networks*) (*CNN*) [40], especialmente útiles para analizar información multidimensional, y las Redes Neuronales Recurrente (*Recurrent Neural Networks*) (*RNN*) [1], que resultan eficaces para modelar secuencias de datos, como el historial de interacciones de los usuarios. El funcionamiento de este tipo de modelos se puede apreciar en la Figura 3.

Además de los ya comentados, en los últimos años se han incorporado a la lista los Largos Modelos del Lenguaje (*Large Language Models*) (*LLM*) [21], que han demostrado una gran mejora a la hora de conseguir recomendaciones personalizadas para el usuario.

2. **Minería de reglas asociativas**: La minería de reglas de asociación es una técnica utilizada para descubrir relaciones significativas en grandes conjuntos de datos, generalmente en forma de reglas de la estructura *si-entonces*. Su objetivo es identificar patrones de co-ocurrencia entre ítems, determinando qué combinaciones de variables aparecen juntas con mayor frecuencia. Matemáticamente, este tipo de reglas se expresan como:

$$X \Longrightarrow Y \tag{1}$$

donde X e Y son conjuntos de ítems y se interpreta como: "Si X, entonces Y".

Existen tres categorías dentro de este tipo de enfoques:

- **Tradicional** [7]: Utiliza algoritmos como *A Priori* para extraer patrones frecuentes entre los datos.

- **Asociación con Soporte Adaptativo (*ASARM*)** [35]: Ajusta dinámicamente el umbral de soporte, permitiendo generar reglas más flexibles.
- **Evolutivo** [14]: Emplea algoritmos evolutivos para optimizar la extracción de reglas, aunque con algunos desafíos.

2.3. Sistemas basados en conocimiento

Como se observa en la Figura 4, los sistemas de recomendación basados en el conocimiento emplean información detallada sobre un dominio específico, como los datos relativos a los usuarios y los productos. Esto implica que utilizan conocimiento especializado para generar recomendaciones que se ajusten a las preferencias y necesidades individuales de cada usuario. En Uta et al. [47] se presenta un análisis más profundo sobre este tipo de sistemas, incluyendo estudios que proponen nuevos algoritmos fundamentados en este enfoque.

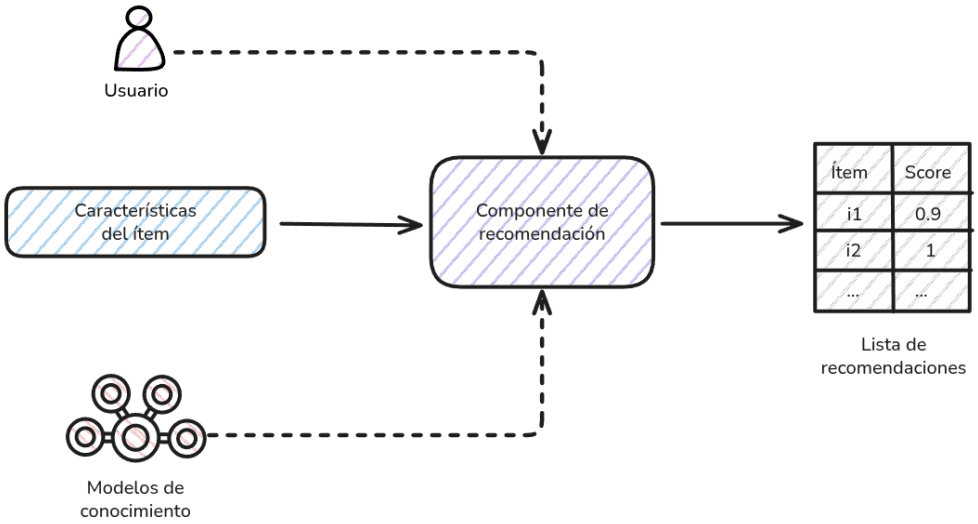


Figura 4: Sistemas basados en conocimiento

Sin embargo, uno de los grandes retos de este enfoque es que recopilar y organizar todo ese conocimiento no es sencillo. Es necesario transformar esa información en un formato que las computadoras puedan entender y utilizar, lo que implica un trabajo considerable en la extracción y estructuración de los datos.

A pesar de estos desafíos, estos sistemas son especialmente útiles para la recomendación de cursos, donde se requiere un entendimiento profundo y detallado del dominio para ofrecer sugerencias precisas y pertinentes.

2.4. Sistemas híbridos

Esta técnica, como su nombre sugiere, combina los distintos enfoques mencionados anteriormente para superar las limitaciones que tendría cada uno por separado. De este modo, se crea un sistema unificado que logra mejores resultados en comparación con el uso individual de estas técnicas.

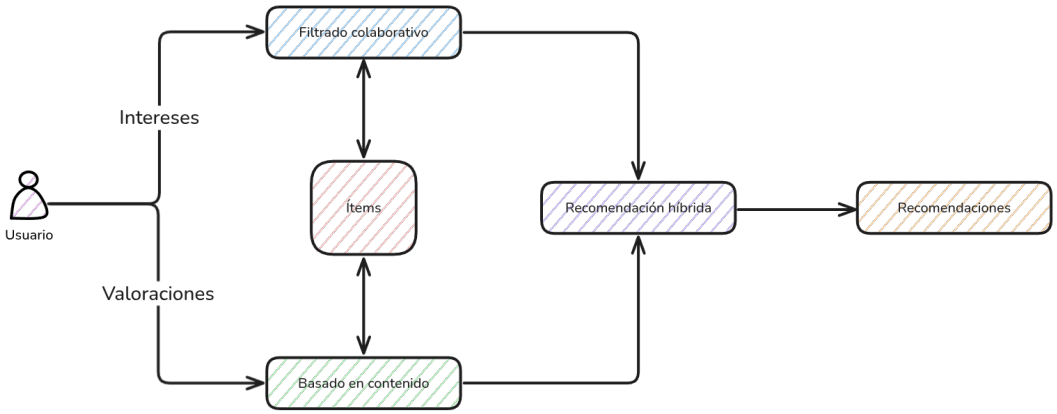


Figura 5: Sistemas híbridos

Dentro del ámbito de la recomendación de cursos, las estrategias que han demostrado ser más efectivas son las siguientes:

- **Filtrado colaborativo y basado en contenido**: La combinación de estos dos enfoques permite considerar tanto la información del usuario como las características de los ítems, lo que mejora la precisión y personalización de las recomendaciones, cómo se puede comprobar en la Figura 5. Trabajos como Do et al. [6] implementan sistemas de recomendación híbridos usando esta combinación.

- **Minería de datos y filtrado colaborativo**: El sistema emplea técnicas de minería de datos para analizar grandes volúmenes de información y descubrir patrones ocultos en el comportamiento de los estudiantes o sus estilos de aprendizaje. Esto permite generar recomendaciones aún más personalizadas y adaptadas a cada usuario [45, 51].

2.5. Métricas de evaluación

La evaluación de los sistemas de recomendación es esencial para garantizar que las sugerencias proporcionadas sean precisas, relevantes y alineadas con los objetivos del usuario. Este proceso implica el uso de diversas métricas que permiten analizar diferentes aspectos del rendimiento del sistema. A continuación, se presentan algunas de las métricas más utilizadas, su definición y cómo se agrupan según el aspecto que evalúan, siguiendo la taxonomía propuesta por Jadon and Patil [30].

2.5.1. Principios de evaluación

Antes de introducir las métricas específicas utilizadas en la evaluación de sistemas de recomendación, es importante comprender los principios fundamentales sobre los que se basan la mayoría de estas métricas. A continuación, se presentan los dos aspectos clave que permiten contextualizar y aplicar correctamente las evaluaciones:

1. **Ranking de predicciones**: Como se explica al comienzo de la sección 2, el objetivo principal de un sistema de recomendación es generar un *ranking* ordenado de ítems que podrían resultar relevantes para un usuario. Para evaluar la calidad de dicho ranking, se debe determinar si los ítems están correctamente ordenados según su relevancia real. Esta relevancia suele establecerse a partir de un umbral en las valoraciones reales: si el sistema predice la puntuación que un usuario daría a un ítem, consideramos relevante cualquier predicción cuya valoración real sea igual o superior a dicho umbral (por ejemplo, 8 sobre una escala de 0 a 10). Una vez definida la relevancia, se puede construir un ranking binario sobre el que aplicar las métricas de evaluación, un ejemplo de *ranking* sería el mostrado en la Tabla 2.

2. **Top-K de recomendaciones**: Muchas métricas se centran únicamente en los primeros k ítems del ranking generado por el sistema, ya que en la práctica los usuarios suelen interactuar solo con las recomendaciones mejor posicionadas. Esta evaluación parcial permite medir la capacidad del sistema para priorizar correctamente los ítems más relevantes, sin necesidad de analizar la totalidad del conjunto.

Usuario	Ítem	Predicción	Real	Relevancia real
u_1	i_1	8.2	7	0
u_1	i_2	7.22	9	1
u_1	i_3	2.68	3	0
u_1	i_4	8.81	8	1
...	...	...	...	...

Tabla 2: Ejemplo de ranking de predicciones y su relevancia

2.5.2. Métricas predictivas

Las métricas predictivas se emplean para evaluar la precisión con la que un sistema de recomendación anticipa las preferencias o valoraciones de los usuarios. Estas medidas permiten determinar qué tan bien el sistema acierta al estimar las calificaciones que los usuarios darían a determinados ítems. Entre las principales métricas utilizadas para este propósito se encuentran las denominadas métricas de valoración.

- ***Mean Squared Error* (*MSE*)**: Cuantifica la diferencia entre los valores estimados y los reales. Se calcula promediando los cuadrados de los errores, es decir, las diferencias entre las predicciones ($\hat{y}_i$) y los valores reales (y_i) sobre n predicciones:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2}$$

  Un valor más bajo indica que las predicciones del modelo se ajustan mejor a los valores reales, es decir, el error promedio es menor.

- ***Root Mean Squared Error* (*RMSE*)**: Cuantifica las diferencias entre las calificaciones que el modelo predice y las que los usuarios realmente asignan. Se calcula como:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3}$$

  Al igual que en él MSE, un valor más bajo indica que las predicciones del modelo se ajustan mejor a los valores reales, es decir, el error promedio es menor.

2.5.3. Métricas de calidad del *ranking*

Estos tipos de métricas cuantifican qué tan bien un sistema de recomendación ordena los ítems según su relevancia para el usuario. A diferencia de las métricas de predicción, que se centran en la exactitud de los valores estimados, las métricas de ranking evalúan la posición y la visibilidad de los ítems relevantes en la lista final.

- $Precision@K$: Indica qué proporción de los K ítems recomendados son realmente relevantes para el usuario. Se calcula como:

$$Precision@K = \frac{\text{Número de ítems relevantes en el top-}K}{K} \in [0, 1] \tag{4}$$

  Un valor más cercano a 1 indica que una mayor proporción de los primeros resultados son relevantes.

  - *Ventaja*: Es fácil de interpretar y refleja la exactitud de las recomendaciones prioritarias.
  - *Limitación*: Depende del número total de ítems relevantes por usuario. Por ejemplo, si un usuario solo considera relevantes 3 ítems y elegimos $K = 10$, la precisión máxima alcanzable será $3/10 = 0.3$, lo que dificulta la comparación entre usuarios con distinto número de ítems relevantes.

- *Recall@K*: Mide la proporción de todos los ítems relevantes que el sistema consigue recuperar dentro de las K recomendaciones principales. Se calcula como:

$$Recall@K = \frac{|I \cap T|}{|I|} \in [0, 1] \tag{5}$$

donde I es el conjunto de todos los ítems relevantes y T es el conjunto del top-K de ítems recomendados, por ende, $|I|$ se define cómo el número total de ítems relevantes y $|I \cap T|$ cómo el número de ítems relevantes en el top-K de recomendaciones. Al igual que $Precision@K$, un valor más cercano a 1 indica que el modelo es capaz de recuperar una mayor proporción de ítems relevantes.

  - *Ventaja*: Refleja la capacidad del sistema para cubrir todas las preferencias del usuario.
  - *Limitación*: No penaliza el orden dentro del ranking y puede dar valores altos incluso si los ítems relevantes aparecen en posiciones lejanas.

- $F_1@K$ **score**: Combina las métricas de $Precision@K$ y $Recall@K$ en un único valor para proporcionar una evaluación equilibrada. Se define cómo:

$$F_\beta@K = \frac{(1 + \beta^2) \cdot Precision@K \cdot Recall@K}{(\beta^2 \cdot Precision@K) + Recall@K} \in [0, 1] \tag{6}$$

donde $\beta \in [0, 1]$ representa la importancia de $Recall@K$ en relación con $Precision@K$, cómo queremos obtener la media armónica entre estas dos métricas, establecemos $\beta = 1$, por lo que la definición final sería:

$$F_1@K = 2 \cdot \frac{Precision@K \cdot Recall@K}{Precision@K + Recall@K} \in [0, 1] \tag{7}$$

  - *Ventaja*: Ofrece un balance entre precisión y cobertura, penalizando fuertemente los casos donde alguna de las dos métricas es baja, lo que ayuda a evitar soluciones que solo optimicen una de ellas.

- *Limitación*: Al ser una media armónica, puede ser difícil interpretar o comparar directamente los valores cuando Precision y Recall varían mucho.

- **Recall y Precision promedio**: Las métricas $Precision@K$ y $Recall@K$ mostradas anteriormente se consideran para un único usuario, si queremos tener en cuenta el total de usuarios U, podemos calcular la media de cada una de estas métricas respectivamente ($Average\ Precision@K$ y $Average\ Recall@K$):

$$Average\ Precision@K = \frac{1}{U}\sum_{u=1}^{U} Precision_u@K \in [0,1] \tag{8}$$

$$Average\ Recall@K = \frac{1}{U}\sum_{u=1}^{U} Recall_u@K \in [0,1] \tag{9}$$

- ***Mean Reciprocal Rank* (*MRR*)**: Corresponde al promedio de los recíprocos de las posiciones del primer ítem correcto en los resultados. Es decir, si el ítem correcto aparece en la posición k, su rango recíproco es $1/k$. Para un conjunto de interacciones Q, el MRR se calcula de la siguiente manera:

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i} \in [0,1] \tag{10}$$

  donde $rank_i$ es la posición del primer ítem relevante para el usuario u en el top-K de resultados. Un valor más cercano a 1 indica que, en promedio, los ítems relevantes se encuentran más cerca del inicio de las recomendaciones.

  - *Ventaja*: Beneficia cuando la primera posición del *ranking* tiene una gran importancia.
  - *Limitación*: Solo se limita al primer ítem relevante y no toma en cuenta el resto de ítems.

- ***Mean Average Precision* (*MAP*)**: Combina precisión y posición en una única métrica. Para cada usuario, se calcula la precisión en cada punto donde

aparece un ítem relevante, también denominada como *Average Precision* (AP), y luego se promedia.

$$AP@K = \frac{1}{|I|} \sum_{k=1}^{T} Precision(k) \times rel(k) \tag{11}$$

$$rel(k) = \begin{cases} 1 \text{ si } k \text{ es relevante} \\ 0 \text{ en caso contrario} \end{cases} \tag{12}$$

$$MAP@K = \frac{1}{U} \sum_{u=1}^{U} AP_u@K \in [0, 1] \tag{13}$$

Un valor más cercano a 1 indica una mejor capacidad del sistema para posicionar los ítems relevantes en los primeros lugares de la lista.

- *Ventaja*: Evalúa tanto la exactitud de las recomendaciones como qué tan bien el sistema puede ordenar los ítems relevantes dentro de la lista.
- *Limitación*: Puede ser difícil de comunicar y no tiene una explicación intuitiva inmediata.

- ***Hit Rate* (*HR*)**: Mide la proporción de usuarios que reciben al menos un ítem relevante.

$$HR@k = \begin{cases} 1 \text{ si hay al menos un ítem relevante en el top-}K \\ 0 \text{ en caso contrario} \end{cases} \tag{14}$$

Debido a su naturaleza, este valor de esta métrica aumentará conforme aumente el valor de k.

- *Ventaja*: Es una métrica intuitiva y sencilla que refleja si el sistema es capaz de recomendar al menos un *ítem* relevante.
- *Limitación*: No considera la cantidad ni la posición de los *ítems* relevantes dentro del top-K.

- ***Normalized Discounted Cumulative Gain* (*NDCG*)**: Mide la calidad de un sistema de *ranking*, considerando la posición de los ítems relevantes en la lista, dando mayor peso a los ítems que se encuentran en posiciones más altas. Para calcularlo, primero debemos calcular el DCG, que se define como:

$$DCG@K = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}, \tag{15}$$

donde rel_i es la relevancia del ítem en la posición i del *ranking* y p es el número de ítems ordenados.

De esta manera, la expresión final sería:

$$NDCG@K = \frac{DGC@K}{IDGC@K}, \tag{16}$$

donde $IDGC$ representa el DGC máximo cuando los ítems están perfectamente ordenados en orden descendente de relevancia. Indica una mejor capacidad del sistema para posicionar más ítems relevantes en las primeras posiciones del *ranking*.

  - *Ventaja*: Tiene en cuenta tanto la relevancia de los *ítems* como su posición, penalizando fuertemente los *ítems* relevantes que aparecen en posiciones bajas del *ranking*.
  - *Limitación*: Su cálculo puede ser más costoso computacionalmente y depende de una definición adecuada de la relevancia de los *ítems*.

3. Formulación del problema y Objetivos

El objetivo principal de este *TFG* es desarrollar un sistema de recomendación para plataformas de *e-learning*, utilizando conjuntos de datos públicos. Este proyecto abarcará el preprocesamiento y análisis de la información disponible y llevará a cabo un estudio experimental con diferentes modelos de recomendación, adaptados tanto a los datos disponibles como al contexto educativo. La finalidad es evaluar las ventajas y limitaciones de diversos enfoques en el ámbito de los sistemas de recomendación aplicados a *e-learning*, contribuyendo a mejorar la personalización y efectividad de estas plataformas.

3.1. Planteamiento del problema

A diferencia de otros ámbitos como el entretenimiento audiovisual o el comercio electrónico, donde abundan los conjuntos de datos públicos y privados proporcionados por grandes corporaciones, el acceso a recursos específicamente orientados al contexto educativo sigue siendo limitado. No obstante, este trabajo recopila los datos disponibles públicamente que se han usado en otros proyectos y que se consideren relevantes para el desarrollo de sistemas de recomendación educativa, de todos estos conjuntos de datos, serán seleccionados aquellos que se consideren más completos y serán sometidos a un proceso de preprocesamiento, que incluirá la detección y corrección de valores atípicos, la normalización de formatos y la construcción de estructuras fundamentales como las matrices usuario-ítem y los perfiles de contenido, con el fin de dejarlos preparados para la fase de entrenamiento y evaluación de los algoritmos.

Por otro lado, se cuenta con implementaciones estándar de algoritmos de recomendación en librerías como *Surprise* [27]. Estas herramientas servirán como referencia para establecer un marco comparativo sólido. Con tal fin, se ha desarrollado un sistema que calcula de forma coherente las mismas métricas de evaluación para todos los algoritmos, asegurando así la compatibilidad con la mencionada librería y

facilitando la comparación directa entre modelos.

Finalmente, se definirá y ejecutará un estudio experimental exhaustivo que permita evaluar de forma objetiva y cuantificable el rendimiento de nuestra propuesta en comparación con las implementaciones consideras el estado del arte. Esta evaluación se realizará siguiendo las métricas descritas en la sección 2.5. Los resultados obtenidos proporcionarán una base sólida para valorar el funcionamiento de los enfoques desarrollados y su aplicabilidad en escenarios reales de enseñanza virtual.

3.2. Objetivos planteados

Hemos estructurado los objetivos del trabajo en una serie de pasos que abordan desde el estudio del estado del arte hasta la implementación y validación experimental de diferentes algoritmos de recomendación:

1. **Análisis del estado del arte en sistemas de recomendación para *e-learning***: Estudiaremos y analizaremos las investigaciones más recientes y relevantes sobre sistemas de recomendación aplicados al ámbito educativo.

2. **Revisión y preprocesamiento de conjuntos de datos que estén disponible y puedan actuar como *benchmarks***: Seleccionaremos algunos conjuntos de datos públicos de plataformas de *e-learning* que puedan servir como referencia para evaluar la efectividad del sistema de recomendación.

3. **Selección de sistemas de recomendación adecuada para los entornos *e-learning***: Se seleccionarán diferentes algoritmos para sistemas de recomendación, ya sean basados en contenido o de filtrado colaborativo, estos pueden estar implementados en una librería o por nosotros mismos, dependiendo de como se dé el caso.

4. **Desarrollo, adaptación e implementación de sistemas de recomendación adecuados para las características disponibles en los entornos**

e-learning: En esta fase, se lleva a cabo la construcción del sistema de recomendación, adaptándolo específicamente a las características y limitaciones del entorno deseado.

5. **Comparativa de las distintas técnicas seleccionadas para evaluar su rendimiento en el problema educativo**: Se realizará una evaluación exhaustiva de los sistemas de recomendación implementados para medir su efectividad en el contexto educativo.

.

4. Metodología de trabajo

La metodología empleada en este proyecto es *Cross-Industry Standard Process for Data Mining* (*CRISP-DM*), un enfoque estructurado y ampliamente adoptado para guiar el desarrollo de proyectos de minería de datos. Esta metodología, desarrollada a finales de los años 90, se ha consolidado como un estándar "de facto" en el sector, gracias a su versatilidad y eficacia en la extracción de valor a partir de datos, tal y como se expone en Martínez-Plumed et al. [37]. Por ello, se presenta como la opción más adecuada para abordar este trabajo.

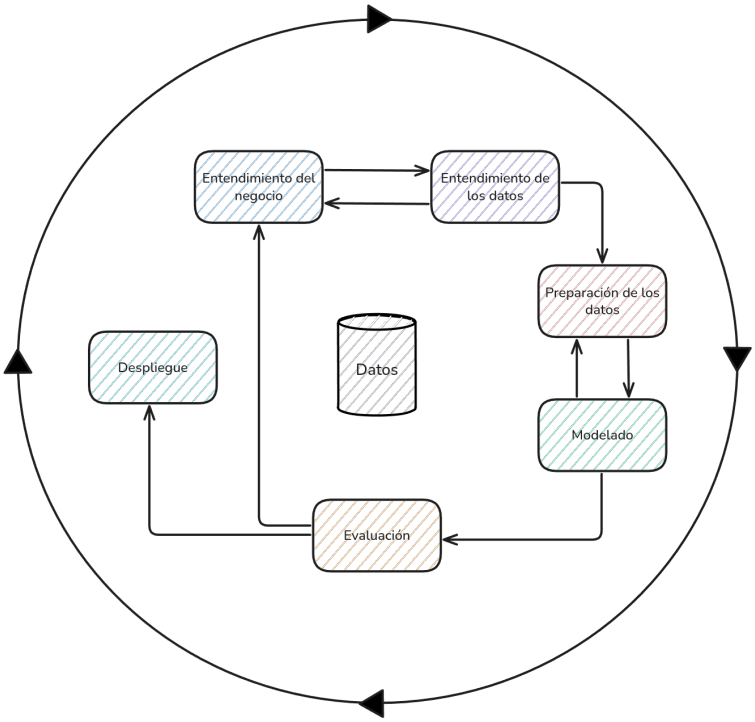


Figura 6: Fases de la metodología *CRISP-DM*

La Figura 6 ilustra las seis fases fundamentales en las que se estructura esta metodología: entendimiento del negocio, entendimiento de los datos, preparación de los datos, modelado, evaluación y despliegue. Cada una de estas fases proporciona una guía clara para avanzar de manera sistemática, desde la definición del problema hasta la implementación práctica de la solución.

4.1. Entendimiento del negocio

Esta fase representa el punto de partida fundamental del proyecto, ya que tiene como propósito principal comprender en profundidad la naturaleza del problema que se busca resolver. No se trata únicamente de identificar una necesidad técnica, sino de alinear de manera coherente los objetivos funcionales del sistema de recomendación con las metas estratégicas y operativas del entorno en el que será desplegado. Esta alineación es crucial para garantizar que el sistema no solo funcione correctamente desde un punto de vista técnico, sino que también genere un impacto positivo y medible en la plataforma y en la experiencia de sus usuarios.

En el ámbito específico del *e-learning*, los objetivos de negocio pueden abarcar diversas dimensiones que van desde mejorar la retención de los estudiantes en los cursos, fomentar una participación más activa y constante, personalizar la experiencia de aprendizaje en función de las características y necesidades de cada estudiante, hasta incrementar la tasa de finalización de los programas formativos ofrecidos. Estas metas reflejan la necesidad de adoptar soluciones tecnológicas que no solo sean innovadoras, sino también efectivas para abordar los retos específicos del aprendizaje en línea.

En este trabajo, como se detalló previamente en la Sección 3, se plantea el diseño e implementación de un sistema de recomendación que sea capaz de ofrecer sugerencias precisas y personalizadas sobre los contenidos disponibles. El objetivo es que cada estudiante reciba recomendaciones adaptadas a sus intereses, historial de interacciones y nivel de progreso, lo cual se espera que tenga un efecto directo en su motivación y compromiso con la plataforma. A su vez, se anticipa que una mayor implicación de los estudiantes contribuirá de forma significativa a reducir las tasas de abandono y mejorar la eficacia general del proceso de aprendizaje. De este modo, se busca establecer una sinergia entre la tecnología de recomendación y las estrategias pedagógicas de la plataforma para alcanzar resultados sostenibles y alineados con los objetivos institucionales.

4.2. Entendimiento de los datos

Esta fase se enfoca en recolectar los datos disponibles, explorar sus características fundamentales y evaluar su idoneidad para el análisis. La comprensión incluye examinar la estructura de los datos, los tipos de variables, la distribución de valores, la presencia de valores atípicos, duplicados o datos faltantes, así como inconsistencias o errores. Esta evaluación permite al equipo entender si los datos son suficientes y confiables para continuar con el proceso, o si es necesario conseguir más fuentes o mejorar la calidad de la información existente.

En consonancia con este enfoque, se ha llevado a cabo una búsqueda y recopilación de diferentes conjuntos de datos con el fin de contrastar su desempeño en contextos reales. En esta sección se describe en detalle los tres datasets utilizados y se analizará su estructura y las características relevantes que conforman su contenido.

4.2.1. *MARS* dataset

El conjunto de datos *Mandarine Academy Recommender System* (*MARS*) [15], ofrece datos provenientes de la empresa Mandarine Academy, dedicada a la creación de contenido pedagógico en línea (videos, cuestionarios, documentos, entre otros) con el objetivo de facilitar la digitalización de los entornos laborales y adaptarse a las tendencias actuales.

Los datos se estructuran en torno a tres entidades principales, cada una con sus respectivos archivos en formato Valores Separados por Comas (*CSV*): *users*, *items* y *ratings* (tanto implícitos como explícitos). Cada archivo incluye información relativa a usuarios de habla inglesa y francesa.

La entidad *users* está compuesta por su identificador (*User ID*) y su categórica de trabajo (*Job*) por motivos de seguridad, sin embargo, debido a que este último atributo no es obligatorio, la gran mayoría de usuarios no lo ha indicado, lo que supone una gran cantidad de datos faltantes.

Característica	Descripción	Tipo	Contador (FR-EN)	Falta (FR-EN)
User ID	Identificador único	Entero	121345 - 9902	0 % - 0 %
Job	Categórica de trabajo	Categórica	9057 - 1409	92.53 % - 85.7 %

Tabla 3: Características de la entidad *users* para el conjunto de datos *MARS*.

Tal como se muestra en la Tabla 3, la versión francesa del conjunto de datos hay un mayor número de usuarios (121345) que en la inglesa (9902).

Por otro lado, la entidad *items* se refiere a los diferentes recursos que se pueden encontrar en los cursos. En la Tabla 4 se muestran los atributos de los recursos, que se entregan en formato video y pueden tener subtítulos en 11 idiomas, encontrados en la descripción.

Característica	Descripción	Tipo	Contador (FR-EN)	Falta (FR-EN)
Item ID	Identificador único	Entero	1451 - 1167	0 % - 0 %
Language	Categórica de trabajo	Categórica	1451 - 1167	0 % - 0 %
Title	Título del contenido	Texto	1451 - 1167	0 % - 0 %
Views	Número de vistas	Entero	1392 - 1119	4 % - 4 %
Description	Descripción del contenido	Texto	1221 - 1009	15.85 % - 13.53 %
Creation Date	Fecha de carga del contenido	Fecha	1451 - 1167	0 % - 0 %
Duration	Duración en segundos	Entero	1451 - 1167	0 % - 0 %
Type	Tutorial, Caso de Uso o Webcast	Categórica	1451 - 1167	0 % - 0 %
Level	Principiante, Intermedio, Avanzado o Indefinido	Categórica	349 - 475	75.94 % - 59.29 %
Job	Profesiones relacionadas	Categórica	326 - 1167	77.53 % - 74.46 %
Software	Software relacionado	Categórica	1349 - 1167	7 % - 5.31 %
Theme	Tema relacionado	Categórica	1269 - 1167	12.54 % - 25.44 %

Tabla 4: Características de la entidad *items* para el conjunto de datos *MARS*.

Existen tres tipos de recursos: tutoriales (2-3 minutos), casos de uso (2-3 minutos) y *webcasts* (30 minutos). Pueden etiquetarse por *job*, *software* y *theme*, aunque faltan muchas etiquetas de nivel y trabajo. La versión francesa de la plataforma tiene más contenido que la versión en inglés, con 1451 y 1167 ítems respectivamente.

La entidad *ratings* se divide en *ratings* explícitos e implícitos, en la vida real, ambos datasets se considerarían implícitos, pero en el contexto del sistema de recomendación, el tiempo de visualización (implícito) se trata como explícito. El archivo `explicit_ratings.csv` detalla la cantidad de tiempo que cada usuario dedicó a ver un video.

Característica	Descripción	Tipo	Contador Único (FR-EN)	Contador Total (FR-EN)
Item ID	Identificador único	Entero	1350 (93 %) - 776 (66.49 %)	85339 - 3659
User ID	Identificador único	Entero	9789 (8 %) - 822 (8.3 %)	-
Watch %	Porcentaje de visualización del video	Flotante	-	-
Rating	Porcentaje de visualización escalado en 1-10	Categórica	-	-
Creation Date	Fecha del evento	Fecha	-	-

Tabla 5: Características de la entidad *explicit ratings* para el conjunto de datos *MARS*.

En la Tabla 5, la columna de Contador Único muestra el número de usuarios/*items*, así como el porcentaje del número total de usuarios e *items* en la plataforma. El atributo *Watch %* indica un valor de 0 (el video no ha comenzado) a 100 (el video ha finalizado), mientras que el atributo *Rating* convierte los valores de *Watch %* a una escala de 1 a 10 para facilitar la comparación. Ambos atributos pueden considerarse una forma de determinar el grado de interés de los usuarios.

El archivo `implicit_ratings.csv` muestra el historial de navegación de los usuarios en la plataforma. Los datos almacenados se pueden observar en la Tabla 6.

Característica	Descripción	Tipo	Contador Único (FR-EN)	Contador Total (FR-EN)
Item ID	Identificador único	Entero	1377 (94.90 %) - 957 (82 %)	253827 - 21908
User ID	Identificador único	Entero	18519 (15.26 %) - 3007 (30.36 %)	-
Creation Date	Fecha del evento	Fecha	-	-

Tabla 6: Características de la entidad *implicit ratings* para el conjunto de datos *MARS*.

Este archivo almacena esencialmente información sobre qué usuario vio qué *item*. En ambos archivos existen algunas observaciones duplicadas de pares (usuario, ítem), sin embargo, el atributo *Creation Date* distingue estas observaciones.

4.2.2. *ITM-Rec* dataset

Este conjunto de datos fue introducido por Zheng [50] y recopilado mediante un cuestionario realizado con la plataforma *Qualtrics*. Los participantes fueron estudiantes de posgrado matriculados en la especialización de gestión de datos y análisis en el departamento de *ITM* del Illinois Tech. El objetivo del cuestionario era registrar las preferencias de los estudiantes respecto a los temas de los proyectos finales en tres asignaturas distintas: Bases de Datos (*Data Bases*) (*DB*), Análisis de datos (*Data Bases*) (*DA*) y Ciencia de datos (*Data Science*) (*DS*). Cabe destacar que

este dataset tiene en cuenta el periodo de cuarentena debido a la pandemia por el *COVID-19*.

El cuestionario también recopilaba información demográfica de los estudiantes (edad, género, estado civil) y características del contenido de los ítems (como título, URL y descripción textual). Toda esta información se encuentra almacena en varios archivos *CSV*.

Los estudiantes podían realizar sus proyectos de forma individual o en grupo. En este último caso, completaban el cuestionario de nuevo como grupo, entregando una sola copia consensuada tras la discusión. Estas respuestas reflejan las preferencias grupales y complementan los datos individuales, enriqueciendo el conjunto de datos para el desarrollo y evaluación de sistemas de recomendación educativa. Sin embargo, para este trabajo solamente se han empleado las valoraciones individuales de los alumnos, de esta manear simplificamos el proceso de evaluación.

El dataset cuenta con un total de 476 usuarios únicos y su información está compuesta por su género (*Genre)*, edad (*Age*) y si este está casado o no (*Married*), junto con un identificador único para cada usuario (*UserID)*, toda esta información se puede ver más detalladamente en la Tabla 7.

Característica	**Descripción**	**Tipo**
User ID	Identificador único	Entero
Genre	Género del usuario	Categórica
Married	Estado civil del usuario (está o no casado)	Booleana

Tabla 7: Características de la entidad *users* para el conjunto de datos *ITM*.

Por otro lado, la Tabla 8 recoge información sobre los 70 ítems únicos que componen el dataset, está formada por un identificador único (*Item ID*), su nombre o título (*Title*), URL y descripción (*Description*).

Por último, el *dataset* ofrece un total de 5230 interacciones de usuario-ítem, co-

mo se puede comprobar en la Tabla 9. Estas interacciones se componen, además de sus respectivos identificadores (*User ID* e *Item ID*), de otra serie de información contextual cómo el tipo de clase (*Class*), el semestre en el que se ha realizado (*Semester*), el periodo de cuarentena debido al *COVID-19* (*Lockdown*), interés por el dominio de la aplicación (*App*), interés en el procesamiento o almacenamiento de los datos (*Data*), y facilidad de uso de los datos (*Ease*) y, por último, la valoración dada por el usuario al ítem (*Rating*), en total, el conjunto de datos cuenta con 5230 valoraciones.

Característica	**Descripción**	**Tipo**
Item ID	Identificador único	Entero
Title	Nombre del curso	Texto
URL	URL del ítem	Texto
Descripción	Descripción del ítem	Texto

Tabla 8: Características de la entidad *item* para el conjunto de datos *ITM*.

Característica	**Descripción**	**Tipo**
User ID	Identificador único	Entero
Item ID	Identificador único	Entero
Rating	Valoración del usuario al ítem	Entero
App	Interés por el dominio de la aplicación	Entero
Data	Interés en el procesamiento o almacenamiento de los datos	Entero
Ease	Facilidad de uso de los datos	Entero
Class	Tipo de clase (*DB*, *DA* o *DS*)	Categórica
Semester	Semestre en el que se realiza la interacción (Spring o Fall)	Categórica
Lockdown	Periodo de cuarentena debido al *COVID-19* (PRE, DUR, POS)	Categórica

Tabla 9: Características de entidad *rating* para el conjunto de datos *ITM*.

4.3. Preparación de los datos

Esta etapa del proyecto tiene como finalidad transformar los datos iniciales, en su estado bruto, en un conjunto estructurado, limpio y adecuado para el entrenamiento de modelos de recomendación. Se trata de una fase fundamental en la que se aplican diversas técnicas de preprocesamiento para asegurar que la información

utilizada refleje de forma precisa y consistente el dominio del problema. El objetivo es eliminar inconsistencias, reducir el ruido, homogeneizar los formatos y adaptar las variables para que sean compatibles con los requerimientos del modelo.

Durante este proceso, se realizan tareas como la limpieza de datos (eliminación de valores nulos o erróneos), la selección de atributos relevantes, la codificación de variables categóricas, la normalización de variables numéricas y la integración de distintas fuentes de información. La decisión sobre qué técnicas aplicar depende en gran medida de la naturaleza de las variables presentes en cada conjunto de datos y del uso que se pretende hacer de ellas dentro del sistema.

Tal y como se expuso en la sección 4.2, todos los conjuntos de datos utilizados en este trabajo han sido sometidos a un proceso de preprocesamiento cuidadosamente diseñado. Este proceso fue ajustado según las características específicas de cada variable. En particular, se presta especial atención a la distinción entre variables numéricas y categóricas, dado que el modelo, como se describirá más adelante en la sección 5.2, está preparado para trabajar exclusivamente con estos dos tipos de variables, además de esto, también se procesan de manera independiente los identificadores únicos asociados a los usuarios y los ítems. Por último, también ha sido necesario refactorizar diferentes tablas en un formato común para que puedan ser utilizadas en los tipos de algoritmos evaluados.

4.3.1. Variables numéricas

Las variables numéricas utilizadas en el modelo incluyen métricas cuantitativas derivadas de la interacción del usuario con el contenido, tales como el porcentaje de visualización, el número total de vistas o la duración de la actividad. Estas variables suelen tener escalas y rangos muy diferentes entre sí, lo que puede generar problemas durante el entrenamiento, especialmente en redes neuronales profundas.

Para mitigar estos efectos, se ha aplicado una técnica de normalización min-max, que transforma cada variable numérica escalando sus valores al rango cerrado $[0, 1]$.

La variable x_s obtenida tras esta transformación se puede expresar con la fórmula:

$$x_s = \frac{x - x_{min}}{x_{max} - x_{min}} \cdot (max - min) + min, \tag{17}$$

donde x es la variable en su escala original, x_{max} y x_{min} son el máximo y mínimo valor de x respectivamente, de la misma manera, max y min representan el máximo y mínimo valor que queremos obtener cuando apliquemos el escalado, en nuestro caso, $max = 1$ y $min = 0$. Por lo que la expresión final sería:

$$x_s = \frac{x - x_{min}}{x_{max} - x_{min}} \in [0, 1]. \tag{18}$$

Esta normalización es crucial para evitar que variables con magnitudes elevadas dominen el proceso de aprendizaje y para asegurar que las funciones de activación, como $ReLU$, trabajen con entradas adecuadas. Además, esta práctica mejora la estabilidad numérica durante el entrenamiento y acelera la convergencia, facilitando que el modelo generalice mejor sobre nuevos datos.

4.3.2. Variables categóricas

Las variables categóricas describen atributos cualitativos y discretos del contenido y el contexto, como el nivel de dificultad del material, el tipo de recurso (video, texto, test), o el semestre académico en que se publicó. Estas variables no tienen una magnitud numérica ni un orden inherente, por lo que es necesario convertirlas en una representación que pueda ser procesada por el modelo.

Para ello, se ha realizado un análisis exhaustivo para identificar todas las categorías presentes en cada variable. Posteriormente, se ha aplicado una codificación por asignación numérica, que consiste en asignar un número entero único a cada categoría distinta dentro de la variable. Por ejemplo, la dificultad podría codificarse como 0 para “baja”, 1 para “media” y 2 para “alta”. Esta técnica permite que el modelo identifique y diferencie entre categorías sin asumir ninguna relación ordinal, ya que el número asignado actúa simplemente como una etiqueta.

4.3.3. Identificadores de usuarios e ítems

Los identificadores únicos de usuario e ítem representan una categoría especial dentro del conjunto de datos. Aunque a menudo se presentan como valores numéricos, no contienen una relación matemática entre ellos, es decir, no existe un orden o magnitud entre dos IDs consecutivos. A pesar de que puedan parecer ya aptos para el modelo por ser enteros, es fundamental tratarlos adecuadamente para evitar que se interpreten como variables numéricas con significado ordinal.

Por este motivo, todos los identificadores han sido transformados mediante una codificación que asigna a cada valor único un número entero consecutivo, sin que este número refleje una relación semántica entre ellos. Esta técnica garantiza que el modelo no infiera patrones incorrectos debido a una aparente estructura numérica en los identificadores.

Además, esta transformación unifica el formato de entrada y asegura que los IDs de usuarios e ítems se procesen de la misma manera que otras variables categóricas, permitiendo al modelo aprender representaciones específicas y diferenciadas para cada entidad. Esta representación es esencial para sistemas de recomendación, donde tanto la identidad del usuario como del ítem son claves para personalizar las sugerencias.

4.4. Modelado

Es en esta etapa donde se desarrollan y entrenan los algoritmos de recomendación necesarios para resolver el problema. La elección de modelos dependerá de los datos y objetivos definidos. Cómo ya hemos explicado en la sección 2, existen una gran variedad de paradigmas de recomendación para escoger. La elección de la técnica dependerá tanto de los objetivos definidos en la fase de negocio como del tipo y calidad de los datos disponibles.

4.4.1. Objetivo del modelo

Existen numerosas implementaciones modernas de algoritmos para sistemas de recomendación. Algunas de ellas generan un "score" que indica cuán recomendable es un determinado ítem para un usuario [18], mientras que otras están orientadas a predecir la Tasa de Clics (*Click-Through Rate*) (*CTR*) [12, 36], es decir, la probabilidad de que un usuario haga clic en un ítem. Este tipo de predicciones resulta especialmente útil en contextos donde los cursos se ofrecen a través de plataformas online.

Por otro lado, la forma más básica de implementación en este tipo de algoritmos es la predicción de valoraciones o *rating prediction*. Como su nombre indica, este enfoque busca estimar la puntuación que un usuario daría a un determinado ítem. Es el método más habitual en algoritmos clásicos de filtrado colaborativo, los cuales se tratarán en mayor detalle en la sección 4.5.3.

Teniendo esto en cuenta, es necesario decidir qué tipo de salida generará el modelo propuesto. Teniendo en cuenta que las implementaciones de la librería *Surprise* [27], con las que se pretende hacer un estudio comparativo, abordan el problema como uno de predicción de valoración, se ha optado por seguir este mismo enfoque para el modelo propuesto. Esta elección permitirá realizar comparaciones más equitativas entre modelos.

Para generar la recomendación final, el sistema identificará cuáles de las valoraciones predichas superan un determinado umbral. Por ejemplo, si las valoraciones se encuentran en una escala de 0 a 10, se podría fijar el umbral en 8. Si la valoración estimada para un usuario u sobre un ítem i supera este umbral, entonces i se considerará recomendable para u.

4.4.2. Enfoque seleccionado

El modelo desarrollado se fundamenta en un enfoque híbrido que integra dos fuentes de información complementarias: las interacciones entre usuarios e ítems y

las características intrínsecas de los propios ítems (metadatos, descripciones, factores de contenido, etc.). Por un lado, el análisis de interacciones permite capturar patrones de comportamiento colectivo y preferencias latentes dentro de la comunidad de usuarios. Por otro lado, la incorporación de atributos específicos de cada ítem enriquece el contexto a la hora de formular recomendaciones, especialmente en situaciones de inicio (*cold-start*) donde los nuevos usuarios o ítems carecen de historial suficiente.

En la práctica, esta combinación se implementa mediante la fusión de dos componentes de predicción:

- **Componente colaborativo**: Eficaz para descubrir afinidades globales, pero presenta limitaciones cuando existen usuarios o ítems con pocas interacciones registradas.
- **Componente de contenido**: Evalúa la compatibilidad entre el perfil del usuario, construido a partir de su historial de interacciones y preferencias explícitas, y los atributos de cada ítem, lo cual reduce el impacto del problema de *cold-start* para nuevos contenidos.

Una vez generadas las predicciones parciales en ambos módulos, el sistema combinará dichos resultados para obtener la predicción final.

Para consultar detalles específicos sobre la arquitectura interna, las decisiones de diseño y el proceso de entrenamiento de cada submódulo, se remite al lector a la sección 5.2, donde se describen cada uno de los componentes anteriormente mencionados.

4.5. Evaluación

El objetivo de esta fase es valorar el rendimiento del sistema desde una doble perspectiva: técnica y de negocio. Se analizan los resultados obtenidos mediante métricas cuantitativas y, si es posible, también se considera la experiencia del usuario

final. La evaluación rigurosa permite verificar si el sistema cumple con los objetivos establecidos y si está listo para su implementación.

4.5.1. Validación cruzada

La validación cruzada o *Cross Validation* (*CV*) es una técnica muy utilizada en aprendizaje automático para evaluar el rendimiento de un modelo predictivo. Su objetivo es estimar cómo se comportará el modelo al hacer predicciones sobre datos no vistos, proporcionando una medida más fiable de su generalización.

Consiste en dividir el conjunto de datos en varias particiones (o *folds*) y entrenar el modelo múltiples veces, cada vez usando un *fold* distinto como conjunto de validación y el resto como conjunto de entrenamiento. Al final, se promedian los resultados obtenidos en cada iteración para obtener una estimación global del rendimiento.

Existen varios tipos de validaciones cruzadas, sin embargo, las más utilizadas en el campo de los sistemas de recomendación son las siguientes:

- *k-fold Cross Validation* (*KFCV*): Consiste en dividir el conjunto de datos disponible en k subconjuntos (llamados *folds*) de tamaño similar. El modelo se entrena y se evalúa k veces, utilizando en cada iteración $k-1$ *folds* para entrenamiento y el *fold* restante para validación. De esta forma, cada observación del conjunto de datos se utiliza una vez como conjunto de prueba y $k-1$ veces como parte del conjunto de entrenamiento.

  Este procedimiento permite obtener una estimación más robusta del rendimiento del modelo, ya que considera distintos subconjuntos de datos para la evaluación. Al finalizar las k iteraciones, se calcula el promedio de todas las métricas de evaluación obtenidas en cada una de ellas, proporcionando una medida global del rendimiento del modelo. El valor de k suele elegirse entre 5 y 10. Un valor de k muy bajo puede generar estimaciones con alta varianza, mientras que un valor muy alto incrementa el coste computacional.

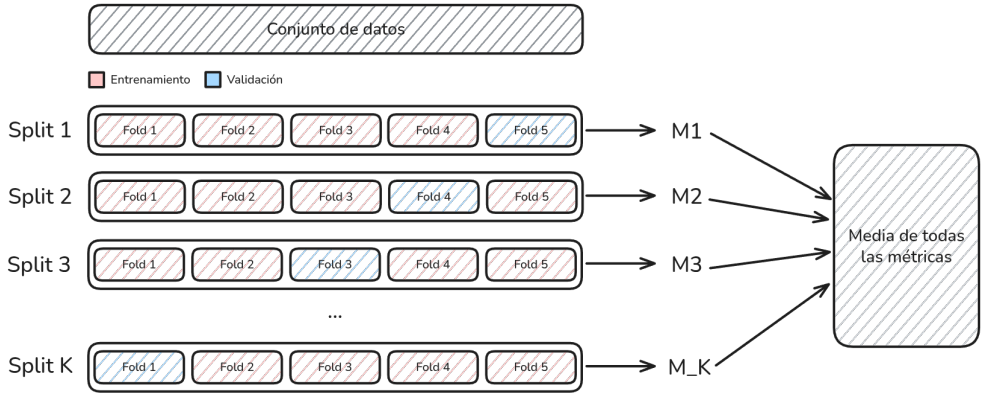


Figura 7: Diagrama de funcionamiento de *KFCV*.

- *Leave One Out Cross Validation (LOOCV)*: Es un caso extremo de *KFCV* donde k es igual al número total de ejemplos en el conjunto de datos. Es decir, para un conjunto de N muestras, se realizan N entrenamientos. En cada uno, se utiliza una sola muestra como conjunto de validación y las $N-1$ restantes para entrenamiento.

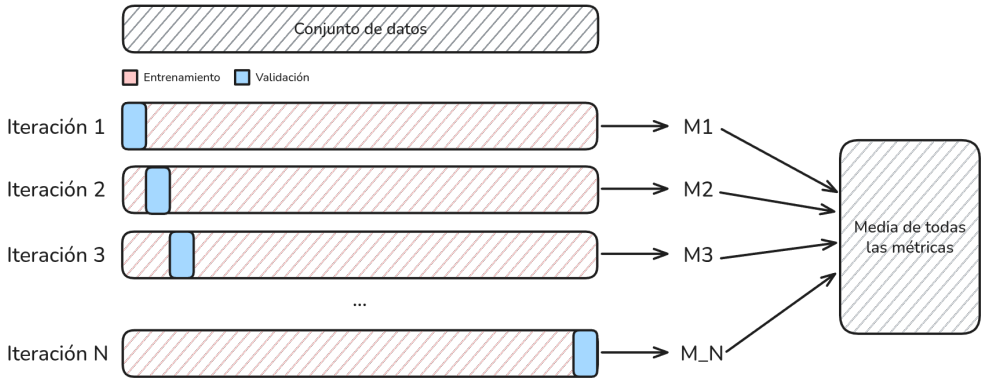


Figura 8: Diagrama de funcionamiento de *LOOCV*.

LOOCV es ampliamente utilizada en la evaluación de sistemas de recomendación, dado que permite evaluar cómo se comportaría el sistema al recomendar un único ítem no visto, lo cual se alinea con el objetivo práctico de muchos de estos algoritmos. Sin embargo, dada la complejidad computacional de este enfoque, en este trabajo se ha optado por utilizar *KFCV*, que ofrece un compromiso más razonable entre coste computacional y calidad de la estimación del rendimiento del modelo.

4.5.2. Análisis estadístico

Los test estadísticos son herramientas muy útiles para la evaluación de modelos de *DL* y *ML*, pues nos permiten determinar si las diferencias observadas en el rendimiento de los modelos son estadísticamente significativas o simplemente fruto del azar.

Un test estadístico parte de la formulación de dos hipótesis: la hipótesis nula H_0 y la hipótesis alternativa H_1. Un ejemplo podría ser:

- Hipótesis nula H_o: Todos los modelos tienen el mismo rendimiento medio en los todos los conjuntos de datos evaluados:

$$H_0 : F_1 = F_2 = \cdots = F_n, \tag{19}$$

  donde F_i es la distribución de la métrica estudiada para el modelo i y n el número total de modelos evaluados.

- Hipótesis alternativa H_1: Al menos un par de modelos difieren en su rendimiento medio, lo que indicaría que el modelo se diferencia del resto:

$$H_1 : \exists\, i, j \text{ tal que } F_i \neq F_j \tag{20}$$

A partir de los datos experimentales se realizan diversos tests estadísticos dependiendo de qué conclusiones se quieran sacar. El resultado de esta comparación es el *p-value*, que indica la probabilidad de observar unos resultados tan extremos como los obtenidos si H_0 fuera cierta. Si el *p-value* es menor que un umbral α predefinido (generalmente $\alpha = 0.05$), se considera que hay evidencia suficiente para rechazar H_0 y aceptar H_1.

En concreto, para este *TFG* se ha utilizado la denominada prueba de *Friedman*, una prueba estadística no paramétrica utilizada para comparar el rendimiento de

varios algoritmos evaluados en los varios conjuntos de datos.

Dado un conjunto de resultados $\{x_{ij}\}_{m\times n}$, que representa una matriz en las que se representan los resultados de m *datasets* para n modelos, se le asigna a cada valor x_{ij} se le asigna un rango r_{ij}, de manera que el mejor valor recibe el rango 1, el segundo el rango 2 y así sucesivamente, lo que nos genera una nueva matriz $\{r_{ij}\}_{m\times n}$. A continuación, se calcula el valor del estadístico Q correspondiente a este test, que se define como:

$$Q = \frac{12m}{n(n+1)} \sum_{j=1}^{n} \left(\bar{r}_j - \frac{n+1}{2} \right)^2, \tag{21}$$

donde $\bar{r}_j$ es la media de todos los rangos calculados en la matriz $\{r_{ij}\}_{m\times n}$:

$$\bar{r}_j = \frac{1}{m} \sum_{i=1}^{n} r_{ij}. \tag{22}$$

Cuando m o n son grandes, la distribución de probabilidad de Q puede aproximarse a una distribución chi-cuadrado (χ^2), por tanto, el valor del *p-value* viene dado por:

$$p\text{-}value = P(\chi^2_{k-1} \geq Q) \tag{23}$$

En caso de que se rechace la hipótesis nula H_0, es decir, cuando $p\text{-}value < \alpha$, entonces se realizarán unas pruebas *post-hoc*, más concretamente las llamdas pruebas de *Nemenyi*, para determinar cuáles son los modelos que difieren entre sí, generando una Diagrama de Diferencias Críticas (*CDD*), que nos permite tener una representación visual de estas representaciones con base en los *rankings* generados por *Friedman*.

4.5.3. Descripción de los modelos

Además de la evaluación del modelo propuesto, también se pretende realizar una comparativa con los modelos previos, para ello, se ha seleccionado la librería *Sur-*

prise [27], que cuenta con un total de 11 algoritmos de diferentes tipos dedicados a sistemas de recomendación. Esta comparación permite contextualizar el rendimiento del sistema desarrollado, evidenciando sus fortalezas y debilidades frente a soluciones bien establecidas. A través de este análisis, se busca identificar en qué aspectos el modelo propuesto ofrece mejoras y cuáles son las áreas que podrían beneficiarse de técnicas alternativas.

Para ello, en esta sección se explicará detalladamente cada uno de estos modelos, su funcionamiento y los criterios utilizados para la evaluación conjunta. Cabe destacar, que todos estos algoritmos están basados en filtrado colaborativo, pues la librería no ofrece la implementación de algoritmos que no sigan este paradigma.

4.5.3.1. Algoritmos de referencia: A pesar de la simplicidad de estos algoritmos, proporcionan puntos de referencia esenciales que permiten establecer una comparativa de rendimiento para cualquier modelo más avanzado. Se conserva el nombre original establecido en la librería, el cual suele coincidir con el especificado por los autores del algoritmo, con el fin de mantener la coherencia y evitar posibles problemas en las traducciones.

- **Normal Predictor** [22]: Este algoritmo predice un rating aleatorio basado en la distribución del conjunto de entrenamiento, que se asume que es normal.

$$\hat{\mu} = \frac{1}{|R_{train}|} \sum_{r_{vi} \in R_{train}} r_{vi} \tag{24}$$

$$\hat{\sigma} = \sqrt{\sum_{r_{vi} \in R_{train}} \frac{(r_{vi} - \hat{\mu})^2}{|R_{train}|}} \tag{25}$$

  La predicción $\hat{r}_{ui}$ es generada a partir de una distribución normal $\mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$, dónde $\hat{\mu}$ y $\hat{\sigma}^2$ se estiman a partir de los datos de entrenamiento usando la Estimación de Máxima Verosimilitud (*Maximum Likelihood Estimation*) [38].

- **Baseline Only** [22]: Predice la estimación base para un usuario y un ítem dados. Aquí, μ representa la media de todos los ratings.

$$\hat{r}_{ui} = b_{ui} = \mu + b_u + b_i \tag{26}$$

Si el usuario u es desconocido, se asume que $b_u = 0$, lo mismo aplica para el ítem i con b_i.

4.5.3.2. Algoritmos basados en *KNN*: Estos algoritmos derivan directamente de un paradigma básico de vecinos más cercanos (nearest neighbors) [20]. Para cada uno de estos algoritmos, el número real de vecinos que se agregan para calcular una estimación es necesariamente $\leq k$. En primer lugar, puede que simplemente no existan suficientes vecinos y, en segundo lugar, los conjuntos $N_i^k(u)$ y $N_u^k(i)$, que se definen como los k vecinos más cercanos del usuario u que ha valorado el ítem i, y los k vecinos más cercanos del ítem i que ha sido valorado por el usuario u respectivamente, solo incluyen vecinos para los cuales la medida de similitud es positiva, pues no tendría sentido agregar valoraciones de usuarios (o ítems) que estén negativamente correlacionados.

- **Basic *KNN*** [24]: Algoritmo básico de filtrado colaborativo, la predicción $\hat{r}_{ui}$ se define como:

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} sim(u,v) \cdot r_{vi}}{\sum_{v \in N_i^k(u)} sim(u,v)}, \tag{27}$$

donde r_{vi} representa la valoración real dada por el usuario $v \in N_i^k(u)$, también se puede expresar como:

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u^k(i)} sim(i,j) \cdot r_{uj}}{\sum_{j \in N_u^k(i)} sim(i,j)}, \tag{28}$$

donde r_{uj} es la valoración real dada para el item $j \in N_u^k(i)$.

- ***KNN* with Means** [24]: Algoritmo básico de filtrado colaborativo, pero tomando en cuenta la media de ratings por cada usuario, las predicciones se expresan como:

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in N_i^k(u)} sim(u,v) \cdot (r_{vi} - \mu_v)}{\sum_{v \in N_i^k(u)} sim(u,v)}, \tag{29}$$

donde μ_u representa la media de todas las valoraciones dadas por el usuario u y μ_v la media de todas las valoraciones del usuario $v \in N_i^k(u)$, también se puede expresar como:

$$\hat{r}_{ui} = \mu_i + \frac{\sum_{j \in N_u^k(i)} sim(i,j) \cdot (r_{uj} - \mu_j)}{\sum_{j \in N_u^k(i)} sim(i,j)} \tag{30}$$

donde μ_i representa la media de todas las valoraciones dadas al ítem i y μ_j la media de todas las valoraciones para el ítem $j \in N_u^k(i)$.

- ***KNN* with z-score** [24]: Algoritmo básico de filtrado colaborativo, pero tomando en cuenta la normalización z-score para cada usuario, el cálculo de las predicciones de las valoraciones se realiza de la siguiente manera:

$$\hat{r}_{ui} = \mu_u + \sigma_u + \frac{\sum_{v \in N_i^k(u)} sim(u,v) \cdot (r_{vi} - \mu_v)/\sigma_v}{\sum_{v \in N_i^k(u)} sim(u,v)}, \tag{31}$$

donde σ_u representa la desviación estándar de todas las valoraciones dados por el usuario u, de la misma manera, σ_v equivale a la desviación estándar de todas las valoraciones dadas por el usuario $v \in N_i^k(u)$, esta predicción también se puede expresar como:

$$\hat{r}_{ui} = \mu_i + \sigma_i \frac{\sum_{j \in N_u^k(i)} sim(i,j) \cdot (r_{uj} - \mu_j)/\sigma_j}{\sum_{j \in N_u^k(i)} sim(i,j)} \tag{32}$$

donde σ_i representa la desviación estándar de todas las valoraciones dadas al ítem i y σ_j equivale a la desviación estándar de todas las valoraciones dadas al ítem $j \in N_u^k(i)$.

- ***KNN* baseline** [24]: Algoritmo básico de filtrado colaborativo, pero tomando en cuenta una valoración de referencia, aquí el cálculo de la predicción se expresa como:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{v \in N_i^k(u)} sim(u,v) \cdot (r_{vi} - b_{vi})}{\sum_{v \in N_i^k(u)} sim(u,v)}, \tag{33}$$

también puede expresarse de la forma:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in N_u^k(i)} sim(i,j) \cdot (r_{uj} - b_{uj})}{\sum_{j \in N_u^k(i)} sim(i,j)} \tag{34}$$

En estas dos expresiones, b_{ui} representa una estimación (*baseline*) de referencia de un usuario u a un ítem i:

$$b_{ui} = \mu + b_u + b_i \tag{35}$$

4.5.3.3. Algoritmos basados en factorización de matrices: Este tipo de algoritmos son una técnica de filtrado colaborativo que permite predecir las preferencias de los usuarios mediante el proceso de *Matrix Factorization* (*MF*), que descompone la matriz de interacciones usuario-ítem en dos matrices de menor dimensión y, tal y como comentan en Koren et al. [33], son uno de los métodos más exitosos en sistemas de recomendación.

- ***Singular Value Decomposition* (*SVD*)** [25]: Se utiliza para aproximar la matriz usuario-ítem mediante la descomposición en matrices de menor rango que capturan las características latentes de usuarios e ítems. Si no consideramos *baselines*, este algoritmo es equivalente al *Probabilistic Matrix Factorization* [43]. La predicción $\hat{r}_{ui}$ se establece como:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \tag{36}$$

Si el usuario u es desconocido, entonces $b_u = p_u = 0$, la misma lógica se aplica para el ítem i con b_i y q_i.

Para estimar todo lo desconocido, minimizamos la siguiente expresión correspondiente al error cuadrático regularizado (*regularized squared error*):

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda \left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2 \right) \tag{37}$$

La minimización se realiza mediante una simple implementación del Descenso del Gradiente Estocástico *Stochastic Gradient Descent* (*SGD*):

$$\begin{aligned} b_u &\leftarrow b_u + \gamma(e_{ui} - \lambda b_u) \\ b_i &\leftarrow b_i + \gamma(e_{ui} - \lambda b_i) \\ p_u &\leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda p_u) \\ q_i &\leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda q_i) \end{aligned} \tag{38}$$

donde $e_{ui} = r_{ui} - \hat{r}_{ui}$, γ es el *learning rate* y λ es el término de regularización.

- ***SVD++*** [25]: Se trata de una extensión del *SVD* que tiene en cuenta las valoraciones implícitas. En este caso, la predicción $\hat{r}_{ui}$ se define como:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j \right) \tag{39}$$

donde los términos y_j son un nuevo conjunto de factores de ítems que capturan valoraciones implícitas e I_u es el conjunto de todos los ítems valorados por el usuario u. Si el usuario u es desconocido, entonces $b_u = p_u = 0$, la misma lógica se aplica para el ítem i con b_i y q_i.

- ***Non-negative Matrix Factorization* (*NMF*)** [25]: Este algoritmo está basado en la Factorización de Matrices No Negativas (*NMF*, por sus siglas en

inglés). Es bastante similar al *SVD*, la predicción r_{ui} se expresa como:

$$\hat{r}_{ui} = q_i^T p_u, \tag{40}$$

el procedimiento de optimización es un descenso de gradiente estocástico (regularizado) con una elección específica del tamaño de paso que garantiza la no negatividad de los factores, siempre que sus valores iniciales también sean positivos.

En cada paso del procedimiento de *SGD*, los factores f del usuario u o del ítem i se actualizan de la siguiente manera:

$$p_{uf} \leftarrow p_{uf} \cdot \frac{\sum_{i \in I_u} q_{if} \cdot r_{ui}}{\sum_{i \in I_u} q_{if} \cdot \hat{r_{ui}} + \lambda_u |I_u| p_{uf}}, \tag{41}$$

$$q_{if} \leftarrow q_{if} \cdot \frac{\sum_{u \in U_i} p_{uf} \cdot r_{ui}}{\sum_{u \in U_i} p_{uf} \cdot \hat{r_{ui}} + \lambda_i |U_i| q_{if}}, \tag{42}$$

donde λ_u y λ_i son los parámetros de regularización. También existe una versión con sesgos, que hace que la definición de r_{ui} cambie a:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \tag{43}$$

4.5.3.4. Otros métodos de filtrado colaborativo: Finalmente, se incluyen algunos métodos adicionales que no pertenecen a las categorías anteriores y han demostrado ser eficaces en distintos contextos. Estos algoritmos capturan regularidades mediante estructuras como clústeres o diferencias estadísticas entre ítems.

- **Slope One** [34]: Se trata de una técnica de filtrado colaborativo, utilizando las diferencias promedio de valoraciones entre pares de ítems para hacer las predicciones. En este caso, y cómo se puede apreciar en Hug [26], la predicción $\hat{r}_{ui}$ se calcula como:

$$\hat{r}_{ui} = \mu_u + \frac{1}{|R_i(u)|} \sum_{j \in R_i(u)} \text{dev}(i, j), \tag{44}$$

donde $R_i(u)$ es el conjunto de ítems relevantes, es decir, el conjunto de ítems valorados por u que también tienen al menos un usuario en común con i. La función $\text{dev}(i, j)$ es definida como la diferencia media entre las valoraciones de i y las de j:

$$\text{dev}(i, j) = \frac{1}{|U_{ij}|} \sum_{u \in U_{ij}} r_{ui} - r_{uj}, \tag{45}$$

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

- **Co-Clustering** [10]: Los usuarios e ítems son asignados a los *clusters* C_u y C_i respectivamente y a algunos *co-clusters* C_{ui}, siguiendo entonces las definiciones de [23], la predicción $\hat{r}_{ui}$ se calcula entonces como:

$$\hat{r}_{ui} = \overline{C_{ui}} + (\mu_u - \overline{C_u}) + (\mu_i - \overline{C_i}), \tag{46}$$

donde $\overline{C_{ui}}$ es la valoración promedia del *co-cluster* C_{ui}, $\overline{C_u}$ es la valoración promedia de los *clusters* de u y $\overline{C_i}$ es la valoración promedia de los *clusters* de i. Si el usuario no es conocido, la predicción $\hat{r}_{ui} = \mu_i$. Si el ítem no es conocido, entonces $\hat{r}_{ui} = \mu_u$, si ambos usuario e ítem son desconocidos, $\hat{r}_{ui} = \mu$.

4.6. Despliegue

La fase final de implementación consiste en trasladar el sistema de recomendación desde un entorno de desarrollo controlado hacia un entorno de producción operativo. En el ámbito del *e-learning*, estos entornos pueden corresponderse con plataformas ampliamente utilizadas como *Moodle* o *Google Classroom*. Esta etapa representa la puesta en práctica de todo el trabajo previo de investigación y desarrollo, transformando un prototipo experimental en una solución funcional capaz de

prestar servicio a usuarios reales bajo condiciones de operación continua.

Asimismo, la implementación definitiva implica integrar el sistema de recomendación con la arquitectura tecnológica existente en la organización, lo que suele conllevar desafíos relacionados con la compatibilidad y la sincronización. Es fundamental establecer protocolos de comunicación bien definidos y mecanismos eficientes de intercambio de datos, como *API*s *REST*, que permitan una interacción fluida entre el sistema de recomendación y el resto de los sistemas.

En el contexto específico de este *TFG*, esta fase no se ha abordado en el resultado final, ya que los objetivos del proyecto se centran en la realización de un estudio experimental del modelo propuesto, más que en su integración en un entorno real. Esta decisión responde al carácter académico y de investigación del trabajo, en el que la prioridad es la propuesta algorítmica y la evaluación rigurosa de su rendimiento bajo condiciones controladas.

5. Desarrollo y Experimentación

En esta sección se abordará de forma detallada el desarrollo completo del modelo que sustenta nuestro sistema de recomendación. Se comentarán las herramientas, librerías y tecnologías utilizadas durante la implementación, destacando su papel en cada fase del desarrollo.

Posteriormente, se describirá la arquitectura elegida, justificando su diseño y explicando cómo opera internamente para generar recomendaciones personalizadas. A continuación, se explicarán todos los hiperparámetros del modelo, detallando su función y relevancia. Con base en ello, se establecerá la configuración final utilizada durante las evaluaciones experimentales.

Finalmente, se presentarán los resultados obtenidos durante el periodo de evaluación para cada uno de los conjuntos de datos seleccionados. Estos serán analizados e interpretados a la luz de los objetivos planteados, destacando tanto los aciertos del sistema como las posibles áreas de mejora. Este análisis permitirá valorar la eficacia del modelo y orientar futuros trabajos o ajustes sobre el mismo.

Todo el código desarrollado para este trabajo se encuentra disponible en un repositorio público alojado en la plataforma *GitHub*, de esta manera, se podrá tener disponible para su uso en futuros estudios.

5.1. Tecnologías utilizadas

Para el desarrollo del sistema de recomendación se ha utilizado el lenguaje de programación *Python* en su versión 3.12 [42], junto con un conjunto de bibliotecas ampliamente reconocidas en el ámbito del análisis de datos y el aprendizaje automático. Python ha sido elegido por su sintaxis clara y concisa, su extensa comunidad de usuarios y desarrolladores, y su ecosistema robusto de librerías especializadas, lo que ha permitido una implementación eficiente, estructurada y fácilmente

mantenible del modelo.

Durante la fase de preprocesamiento y análisis exploratorio de datos, se ha empleado la librería *Pandas* [49], que ofrece herramientas para realizar operaciones de limpieza, transformación, filtrado y agregación de datos de forma eficiente. Para complementar este procesamiento, se ha utilizado *NumPy* [17], una biblioteca fundamental para la computación científica que proporciona soporte para arrays multidimensionales y operaciones algebraicas vectorizadas, contribuyendo a una mejora sustancial del rendimiento en operaciones de bajo nivel.

En lo que respecta a la generación de visualizaciones, se ha utilizado *Matplotlib* [28], una biblioteca versátil para la creación de gráficos estáticos, animados e interactivos. Esta herramienta ha permitido representar gráficamente distribuciones de datos, evolución del entrenamiento y comparativas de métricas, facilitando el análisis visual y la interpretación de los resultados obtenidos.

Para la implementación de técnicas de aprendizaje automático tradicionales, la biblioteca *scikit-learn* [41] ha desempeñado un papel clave. Esta herramienta proporciona una amplia gama de algoritmos para clasificación, regresión y reducción de dimensionalidad, así como utilidades para la normalización de datos, validación cruzada, división de conjuntos de datos y evaluación de modelos base.

Además, como se ha ido mencionando en las secciones anteriores, se ha utilizado *Surprise* [27], una biblioteca especializada en sistemas de recomendación, que facilita la implementación y evaluación de algoritmos como *SVD* o *KNN*, basados en técnicas de filtrado colaborativo. Gracias a esta librería, se han podido establecer comparaciones sólidas entre modelos clásicos y el enfoque propuesto en este trabajo.

La propuesta desarrollada ha sido implementada utilizando *PyTorch* [39], un *framework* de deep learning de código abierto que permite construir y entrenar redes neuronales de manera dinámica y eficiente. Su diseño basado en grafos computacio-

nales dinámicos proporciona una gran flexibilidad para definir arquitecturas personalizadas y controlar el flujo del entrenamiento. Además, su compatibilidad con aceleración por Unidades de Procesamiento Gráfico (*GPU*) ha permitido aprovechar al máximo los recursos disponibles.

Sobre esta base, se ha integrado *Torchmetrics* [5], una biblioteca diseñada para el cálculo estandarizado de métricas durante el entrenamiento y la evaluación de modelos. Esta integración ha asegurado la precisión y consistencia en la medición del rendimiento, al tiempo que ha reducido el esfuerzo necesario para implementar métricas personalizadas.

Asimismo, el uso de *PyTorch Lightning* [8] ha supuesto una mejora sustancial en la organización del código. Este *framework* ligero abstrae gran parte de la lógica repetitiva del entrenamiento, permitiendo separar de forma clara la definición del modelo, la preparación de datos y los bucles de entrenamiento y validación. Además, su integración nativa con *Torchmetrics* han facilitado la escalabilidad y reproducibilidad de los experimentos.

Para comparar el rendimiento de los modelos seleccionados sobre varios conjuntos de datos, se ha utilizado la librería *SciPy* [48], que proporciona herramientas para realizar tests estadísticos como el de *Friedman*, permitiendo así evaluar si existen diferencias significativas en los resultados obtenidos por los distintos algoritmos evaluados.

Finalmente, para la gestión de dependencias y entornos virtuales del proyecto, se ha utilizado el gestor de paquetes *uv* [2], una herramienta moderna que combina rapidez y eficiencia. La herramienta *uv* permite instalar paquetes de forma extremadamente rápida y gestionar entornos de forma aislada, garantizando la coherencia del entorno de ejecución.

5.2. Arquitectura del modelo propuesto

El algoritmo de recomendación propuesto se trata, como se habló en la sección 4.4, de un modelo de *DL* basado en un enfoque híbrido que combina componentes *CF* y *CBF*, lo que permite aprovechar tanto las interacciones históricas entre usuarios y cursos como la información adicional asociada a ambos, como ya se explica en la sección 2.4.

El enfoque seleccionado para el modelo es el de *rating prediction*, cuyo objetivo principal es estimar, con la mayor precisión posible, la valoración $\hat{y}_{ui}$ que un usuario u daría a un ítem determinado i.

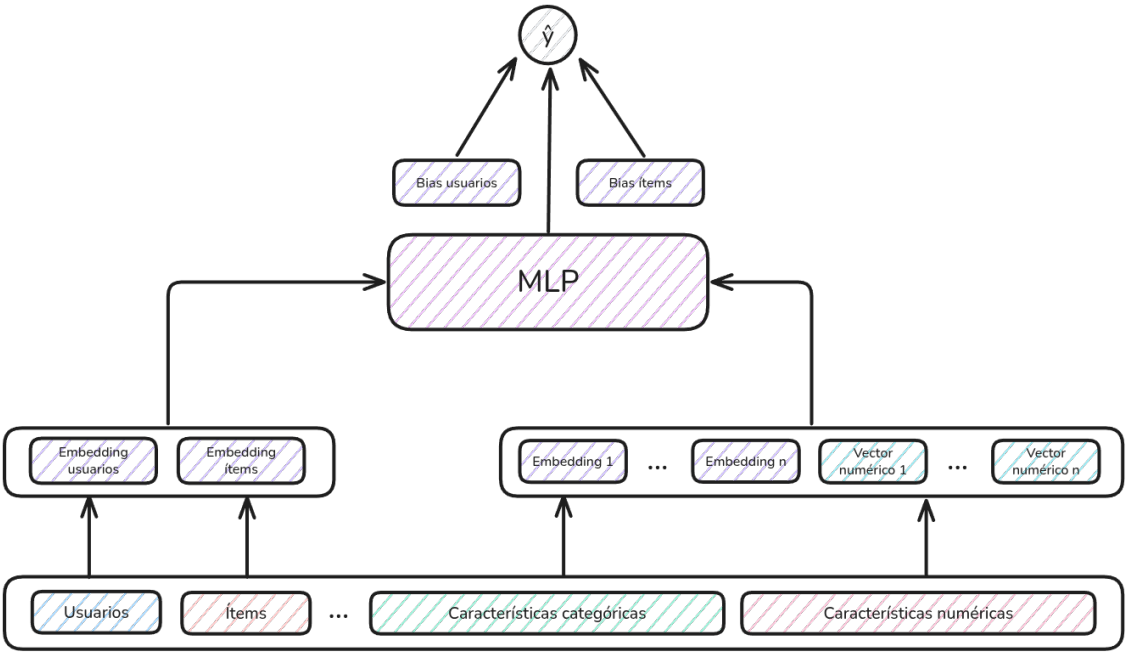


Figura 9: Arquitectura del modelo propuesto

La Figura 9 muestra la arquitectura propuesta para el modelo. Por un lado, cada usuario e ítem se codifican mediante *embeddings* que capturan su identidad en un espacio continuo. Por otro, la información adicional cómo los atributos categóricos también se transforma en vectores y se concatena con dichos *embeddings*, mientras que los vectores que forman las características numéricas que también concatenan a

estos *embeddings*. Esa combinación sirve como entrada para un *MLP* que, junto a dos términos de sesgo (uno para usuario y otro para ítem), produce como salida la valoración estimada de la interacción.

La arquitectura propuesta se compone de tres elementos fundamentales: el componente colaborativo, el componente basado en contenido y el componente híbrido. Cada uno de ellos desempeña un rol específico dentro del modelo y resulta esencial para garantizar su correcto funcionamiento.

A continuación, se describirá en detalle la función de cada uno de estos componentes y cómo procesan los datos de manera secuencial, desde su entrada inicial hasta la obtención del resultado final.

5.2.1. Componente colaborativo

El componente colaborativo se encarga de capturar las preferencias latentes de los usuarios a partir de sus interacciones históricas con los ítems. Para ello, tal como se ilustra en la Figura 11, se emplea una técnica de aprendizaje de representaciones latentes en la que tanto usuarios como cursos quedan representados mediante vectores densos de dimensión d, denominados *embeddings*, de modo que la proximidad entre estos vectores indica afinidad entre el usuario y el ítem correspondiente.

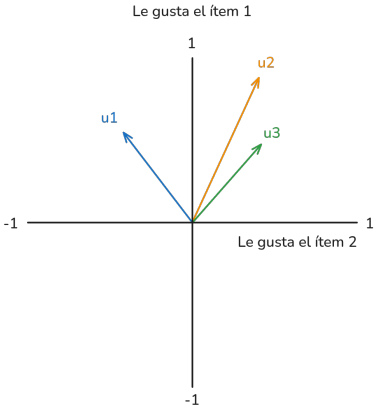


Figura 10: Espacio latente de los usuarios u_1, u_2 y u_3, y los ítems i_1 e i_2 para $n = 2$.

La Figura 10 muestra una representación simple del espacio latente de dimensión $d = 2$ para los usuarios u_1, u_2 y u_3, tomando en cuenta únicamente dos ítems del conjunto de datos i_1 e i_2. Gracias a esta representación podemos intuir que u_1 está interesado en i_1 pero no en i_2, mientras que u_2 está más interesado en i_1, pero también tiene interés en i_2. Para el caso que dos usuarios tengan un mismo interés en un ítem, como son los usuarios u_2 y u_3 con i_2, las distancia de sus vectores de *embeddings* en el espacio latente será menor que para usuarios que no tengan el mismo interés.

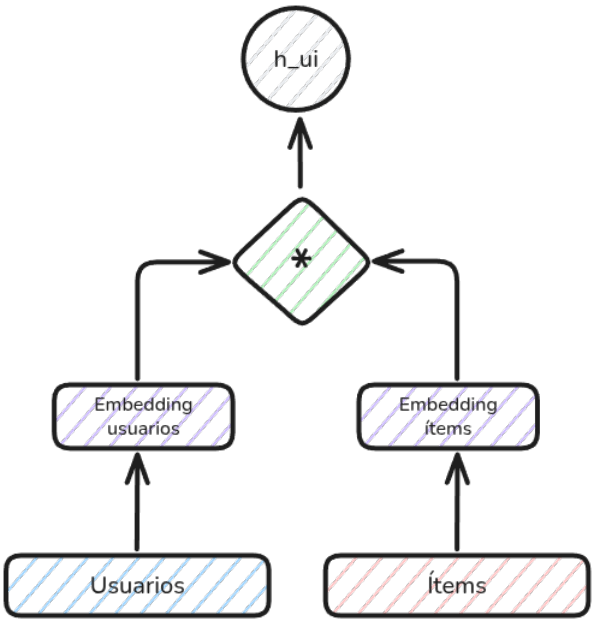


Figura 11: Componente colaborativo del modelo

La interacción h_{ui} entre un usuario y un ítem se representa mediante el producto elemento a elemento, también conocido como producto Hadamard, entre sus vectores de *embedding*:

$$h_{ui} = u_i \odot v_j = [u_{i1}v_{j1},\ u_{i2}v_{j2},\ \ldots,\ u_{id}v_{jd}] \in \mathbb{R}^d, \tag{47}$$

donde $u_i = (u_{i1}, u_{i2}, \ldots, u_{id}) \in \mathbb{R}^d$ y $v_j = (v_{j1}, v_{j2}, \ldots, v_{jd}) \in \mathbb{R}^d$ representan los *embeddings* de dimensión d del usuario $i \in \mathcal{U}$ y del ítem $j \in \mathcal{I}$, respectivamente.

Este tipo de operación se inspira en las técnicas de factorización matricial (*Matrix Factorization, MF*) presentadas en la sección 4.5.3.3 y muy utilizadas en otros modelos basados en *DL* como *DeepFM* [12], *NeuralFM* [18] o *DORIS* [36]. A diferencia del enfoque tradicional de MF, que emplea el producto punto para obtener un valor escalar:

$$s_{ij} = u_i \cdot v_j = \sum_{k=1}^{d} u_{ik} v_{jk} \in \mathbb{R}, \tag{48}$$

aquí se conserva el vector completo resultante del producto de *Hadamard*. Esta decisión permite enriquecer la representación combinándola con otras características adicionales (como información de contenido), lo que ofrece a la red neuronal subsiguiente una entrada más expresiva y detallada. En consecuencia, el modelo puede capturar relaciones más complejas y sutiles entre usuarios e ítems, fusionando de manera más eficaz las perspectivas colaborativa y basada en contenido.

5.2.2. Componente basado en contenido

El componente basado en contenido, representado en la Figura 12, se encarga de capturar información explícita y observable sobre los usuarios y los cursos, que no depende directamente de las interacciones pasadas, donde los usuarios o cursos tienen pocas o ninguna interacción registrada, lo que nos permite evitar el problema del *cold-start*.

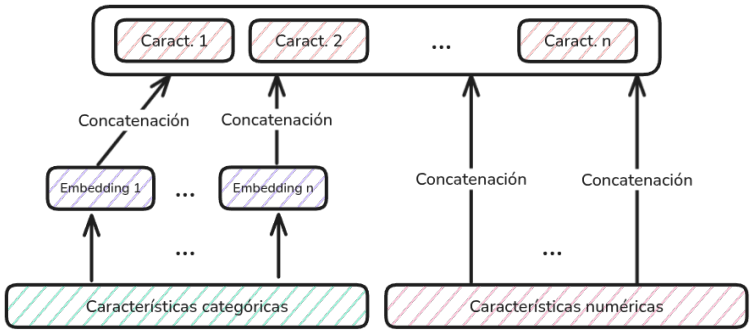


Figura 12: Componente basado en contenido del modelo

Como se puede apreciar en la Figura 12, y cómo ya se menciona previamente en la Sección 4.3, en este modelo se consideran dos tipos de variables de contenido:

- Variables categóricas, como su nivel de dificultad, o el tipo de contenido. Estas variables se transforman mediante capas de *embedding* independientes, que aprenden a representar cada categoría como un vector denso en un espacio semántico. Los *embeddings* para estas variables tienen una dimensión menor que los *embeddings* usados en el componente colaborativo, lo que reduce la complejidad del modelo y evita redundancia.
- Variables numéricas, como el número de visitas del curso o el porcentaje de visionado. Estas se incorporan directamente como vectores de valores continuos.

Los vectores de *embedding* generados para cada variable categórica se concatenan junto con los valores numéricos, dando lugar a una representación vectorial enriquecida que captura la información de contenido de manera compacta y diferenciada. Esta representación permite al modelo reconocer patrones comunes entre cursos similares y usuarios con perfiles específicos, y generalizar mejor cuando se presentan ejemplos nuevos o poco frecuentes.

5.2.3. Componente híbrido

El componente híbrido integra la información implícita capturada por el componente colaborativo con la información explícita del contenido, generando un vector de entrada que recoge tanto el comportamiento del usuario como las características de los ítems. Formalmente, este vector de entrada x de dimensión d_x se construye concatenando:

$$x = \begin{bmatrix} h_{ui} \\ cat_i \\ num_i \end{bmatrix} \in \mathbb{R}^{d_x}, \quad \text{con} \quad d_x = d + K \cdot d_c + N, \tag{49}$$

donde cat_i son las representaciones densas de las variables categóricas pertenecientes a los ítems, num_i los valores normalizados de las variables numéricas, d

representa la dimensión utilizada para representar los *embeddings* de usuario e ítems, K representa el número total de variables categóricas dadas al modelo, d_c es la dimensión asignada a los *embeddings* de las variables categóricas y N representa el número total de variables numéricas introducidas al modelo.

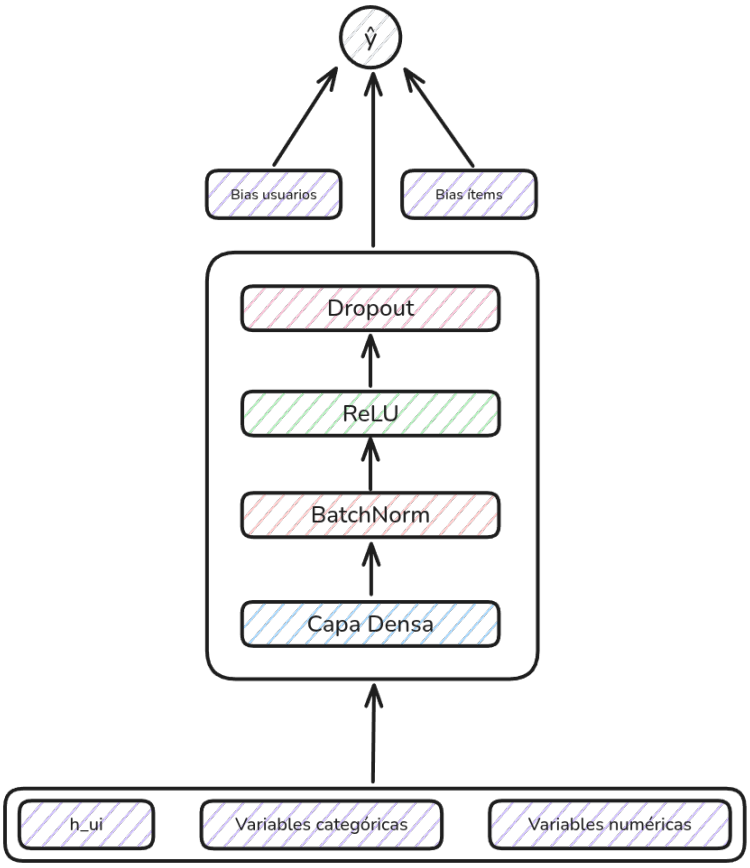


Figura 13: Componente híbrido del modelo

Tal y como muestra la Figura 13, este vector se introduce en un *MLP* formado por varias capas densas. Cada una de estas capas aplica la función de activación *ReLU* (*Rectified Linear Unit*), cuya definición se expresa como:

$$ReLU(x) = max(0, x), \tag{50}$$

lo que permite al modelo capturar relaciones no lineales complejas entre las variables de entrada. *Rectified Linear Unit* (*ReLU*) es además computacionalmente

eficiente y ayuda a evitar el desvanecimiento del gradiente, un problema común en redes profundas.

Tras cada capa densa, se aplican dos técnicas fundamentales para mejorar el rendimiento del modelo: la normalización por lotes (*batch normalization*) [29] y *dropout* [19]. La normalización por lotes consiste en normalizar las salidas de una capa utilizando la media y la desviación estándar del mini-lote actual. Al mantener la distribución de los datos intermedios más constante, se permite usar tasas de aprendizaje más altas y se facilita la convergencia del modelo.

Por otro lado, la técnica de *dropout* introduce aleatoriedad durante el entrenamiento, desactivando de forma temporal (es decir, durante una iteración) un subconjunto de neuronas, seleccionadas con una probabilidad p. Esto impide que ciertas neuronas se vuelvan demasiado dependientes entre sí, lo que ayuda a prevenir el sobreajuste y mejora la capacidad de generalización del modelo. En la fase de inferencia (evaluación), todas las neuronas están activas, y sus salidas se escalan adecuadamente para compensar la desactivación realizada durante el entrenamiento.

La salida final del *MLP*, denotada como $\text{mlp}(x)$, es un escalar que representa la predicción de la puntuación. A esta salida se le suman los sesgos individuales del usuario b_u y del curso b_i, aprendidos previamente por el componente colaborativo. Finalmente, se aplica una operación de recorte para garantizar que la predicción resultante $\hat{y}_{ui}$ se mantenga dentro de un rango válido:

$$\hat{y}_{ui} = \text{mlp}(x) + b_u + b_i \in [r_{min},\, r_{max}], \tag{51}$$

donde r_{min} y r_{max} indican respectivamente el valor mínimo y máximo del rango permitido para las valoraciones, por ejemplo, de 1 a 10.

Por último, como se comenta en la Sección 2.5.1, para determinar el *ranking* final de los ítems recomendados, se utiliza un valor umbral τ que permite distinguir entre ítems relevantes o no relevantes para un usuario. Así se considera que un ítem i es

recomendable para un usuario u si se cumple que:

$$\hat{y}_{ui} \geq \tau. \tag{52}$$

Esta comparación permite transformar una predicción continua en una decisión binaria, lo cual es esencial para calcular posteriormente las métricas de *ranking* vistas en la Sección 2.5.3.

5.3. Hiperparámetros del modelo propuesto

El modelo propuesto cuenta con una serie de hiperparámetros que son necesarios para definir su estructura. Estos son útiles tanto para crear las representaciones latentes de los usuarios y los ítems, como para las características categóricas y numéricas que utilizará. Cada uno de estos hiperparámetros forman parte de un componente diferente del modelo, a continuación se definirán cada uno de ellos.

5.3.1. Hiperparámetros colaborativos

Estos hiperparámetros son los que dan forma al componente basado en filtrado colaborativo del modelo, y, por ende, los que definen el espacio latente para representar correctamente las interacciones de los usuarios con los ítems.

Sea $|\mathcal{U}|$ el número total de usuarios únicos en el conjunto de datos, este valor determina el tamaño del vocabulario para las representaciones vectoriales de usuarios. Análogamente, $|\mathcal{I}|$ representa la cardinalidad del conjunto de ítems, definiendo el espacio de latente para los productos del sistema. Estos parámetros son críticos porque establecen directamente las dimensiones de las matrices de *embeddings* E_u para el usuario u y E_i para el ítem i que el modelo debe aprender durante la optimización.

La dimensionalidad de las representaciones latentes se controla mediante el hiperparámetro d, que define la dimensión del espacio vectorial donde se proyectan tanto usuarios como ítems. Este parámetro determina la capacidad expresiva del modelo para capturar patrones complejos en las preferencias. Una dimensionalidad

d elevada permite representaciones más ricas en el espacio latente $\mathbb{R}^d$, pero incrementa el riesgo de sobreajuste debido al aumento en el número de parámetros libres. Conversamente, un valor de d reducido puede limitar la capacidad del modelo para capturar relaciones sutiles en los datos.

5.3.2. Hiperparámetros basados en contenido

Son los que recibe el modelo para instanciar cada uno de los elementos necesarios para el componente basado en contenido.

El tratamiento de las características categóricas introduce un conjunto de parámetros de cardinalidad $|\mathcal{C}_k|_{k=1}^{K}$, donde cada $|\mathcal{C}_k|$ representa el número de categorías únicas para la k-ésima característica categórica, y K denota el número total de características categóricas disponibles.

Las características numéricas continuas se representan mediante un vector $f \in \mathbb{R}^N$, donde N es el número de variables continuas incorporadas directamente al modelo. Estas características mantienen su naturaleza continua y aportan información cuantitativa sin transformación adicional al proceso de predicción.

5.3.3. Hiperparámetros para el componente híbrido

Estos hiperparámetros definen la forma del *MLP*, que se encarga de combinar los resultados de los otros dos componentes.

Su arquitectura se parametriza mediante una secuencia de tamaños de entrada para cada capa $H = [h_1, h_2, \ldots, h_L]$, con $h_1 > h_2 > \cdots > h_L$ y donde L representa el número de capas ocultas y cada h_ℓ especifica la dimensión de la ℓ-ésima capa oculta.

Por otro lado, el hiperparámetro $p \in [0, 1]$ representa la probabilidad de desactivación aleatoria de neuronas durante el entrenamiento usando la técnica *dropout*, que ya se mencionó en la sección anterior 5.2.

Por último, es necesario especificar los valores mínimos y máximos en los que se puede encontrar el valor de predicción del modelo, estos se definen mediante los parámetros r_{min} y r_{max} respectivamente, lo que permite establecer un rango válido $[r_{min}, r_{max}]$ para las salidas del modelo.

5.4. Configuración de hiperparámetros para evaluación

A continuación explicaremos la configuración de hiperparámetros utilizada para el proceso de evaluación del modelo, esta configuración es importante, pues será la misma para todas las pruebas realizadas para los diferentes conjuntos de datos, de esta manera aseguramos la consistencia en las pruebas que se realicen al modelo.

5.4.1. Configuración seleccionada para el modelo

Para la elección de esta configuración se ha adoptado por no utilizar métodos de optimización automática como la búsqueda en cuadrícula (*Grid Search*), la búsqueda aleatoria (*Random Search*) o la optimización bayesiana, técnicas que, aunque pueden producir configuraciones óptimas, requieren recursos computacionales considerables y tiempo de ejecución que puede extenderse bastante dependiendo del espacio de búsqueda y la complejidad del modelo. En su lugar, se ha optado por una configuración manual, fundamentada en prácticas habituales en la literatura y en pruebas exploratorias previas.

La dimensionalidad de los *embeddings* utilizados para representar a los usuarios e ítems en el espacio latente ha sido establecida en $d = 128$, un valor que proporciona suficiente capacidad expresiva para capturar patrones complejos en las preferencias de usuarios sin incurrir en sobreajuste excesivo. Por otro lado, la dimensión para los vectores de *embedding* asignados a procesar las características categóricas ha sido establecida en $d_c = d/2$.

Para la arquitectura del *MLP* se ha optado por establecer el tamaño de las capas densas de lar red siguiendo la siguiente estructura: primeramente se establece una

capa de tamaño de entrada de 256, este tamaño de entrada es reducido a la mitad hasta llegar a un valor de 16, lo que nos deja una configuración con $L = 5$ capas ocultas: $H = [256, 128, 64, 32, 16]$. Esta progresión permite que las capas iniciales capturen patrones más generales, mientras que las capas posteriores se especializan en relaciones más específicas. Estos mismos valores serán aplicados a las capas de regularización de *batch normalization*.

El hiperparámetro de regularización para *dropout* se ha fijado en $p = 0.5$, un valor estándar que indica, que, durante el entrenamiento, aproximadamente la mitad de las neuronas en cada capa oculta se desactivan aleatoriamente, forzando al modelo a desarrollar representaciones más robustas y distribuidas.

Para finalizar, se ha decidido utilizar el *MSE*, definida previamente en la sección 2.5.2, como función de perdida para el entrenamiento del modelo.

5.4.2. Configuración de modelos para comparación

Los algoritmos de con los que se comparará el modelo propuesto fueron configurados utilizando los hiperparámetros estándar definidos en sus implementaciones de referencia, sin aplicar procesos de optimización adicionales que pudieran introducir sesgos en la comparación. Esta aproximación conservadora asegura que las diferencias observadas en el rendimiento reflejen las características intrínsecas de cada algoritmo en lugar de ventajas derivadas de una sintonización específica.

Para el algoritmo *Baselines Only* la configuración por defecto utiliza el método Mínimos Cuadrados Alternativos (*Alternating Least Squares*) (*ALS*) para estimar las líneas base y establece los valores de los parámetros de regularización para usuarios $\lambda_u = 10$ y para ítems $\lambda_i = 15$.

Para toda la familia de algoritmos basados en *KNN* se establece valor de $k = 40$ vecinos, y se ha calculado la similitud de Diferencia Cuadrática Media (*Mean Squared Difference*) (*MSD*) entre todos los pares de usuarios, su definición es la siguiente:

$$msd(u,v) = \frac{1}{|I_{uv}|} \cdot \sum_{i \in I_{uv}} (r_{ui} - r_{vi})^2, \tag{53}$$

y la similitud se calcula como:

$$\text{msd\_sim}(u,v) = \frac{1}{msd(u,v)+1} \tag{54}$$

Para los algoritmos basados en *MF* la configuración por defecto establece un número de factores = 100, un valor de tasa de aprendizaje $\gamma = 0.005$ y un término de regularización $\lambda = 0.02$.

El algoritmo *Co-Clustering* ha sido configurado con un número de tres clústeres para los usuarios e ítems.

Además, como se menciona más adelante en la Sección 5.4.4, para los algoritmos no determinista se ha decidido modificar el valor de la semilla (*seed*) utilizada para generar la configuración de pesos iniciales.

Para finalizar, es necesario mencionar que algoritmos como *Normal predictor* o *Slope One* no cuentan con hiperparámetros modificables.

5.4.3. Configuración de entrenamiento

La configuración dedicada al entrenamiento busca reducir todo lo posible la posibilidad de sobreajuste del modelo, para ellos se emplean diversas técnicas de regularización, junto a las ya mencionadas como *dropout* o *batch normalization*.

Para la actualización de los pesos, se utiliza el optimizador *ADAM* [32] con una tasa de aprendizaje inicial establecida en $\eta = 0.001$. Además, se aplica regularización $L2$, una técnica que tiene como objetivo mitigar el sobreajuste penalizando los pesos excesivamente grandes. Esta penalización se implementa añadiendo un término adicional a la función de pérdida original:

$$L_{new}(w) = L_{original}(w) + \lambda w^T w \tag{55}$$

donde λ es un hiperparámetro que controla la intensidad de la regularización, y w_i son los pesos del modelo. Este término adicional incentiva que los pesos tomen valores pequeños, contribuyendo a una mejor generalización en datos no vistos. En nuestro caso, se ha establecido un valor de $\lambda = 1 \times 10^{-6}$.

Además, se incorpora un planificador de tasa de aprendizaje (*learning rate scheduler*), que ajusta dinámicamente la tasa de aprendizaje durante el entrenamiento. Este planificador monitorea el error de validación y reduce la tasa de aprendizaje a través de un factor dado si dicha métrica no mejora tras un número épocas consecutivas (definido por el parámetro patience), en nuestro caso, factor = 0.5, por lo que la nueva tasa de aprendizaje será $\eta/2$ y patience = 3. Esta estrategia permite afinar las actualizaciones del modelo en fases avanzadas del entrenamiento, facilitando la convergencia hacia mínimos más estables y precisos.

Durante el entrenamiento del modelo, se aplica una técnica de regularización muy parecida al *learning rate scheduler* conocida como Detención Temprana (*Early Stopping*), cuyo objetivo es interrumpir el proceso de entrenamiento cuando una métrica clave, en este caso el error de validación, deja de mejorar significativamente.

En concreto, esta técnica compara el valor actual del error de validación con el mejor valor obtenido hasta el momento. Si durante un número determinado de épocas no se observa una mejora mínima establecida (definida por un umbral δ), el entrenamiento se detiene automáticamente para evitar el sobreajuste y el uso innecesario de recursos computacionales.

Para este modelo, se ha configurado un umbral de mejora $\delta = 0.001$ y una paciencia = 5. Esto significa que si el error de validación no mejora en al menos 0.001 durante cinco épocas consecutivas, el entrenamiento se interrumpe de forma anticipada.

Por último, se ha empleado un tamaño de batch de 128 para dividir el conjunto de entrenamiento en bloques más pequeños de datos. Esta técnica, conocida como aprendizaje por mini-batches, permite actualizar los pesos del modelo varias veces por época utilizando una fracción del conjunto de datos completo. De esta forma, se mejora la eficiencia computacional, se estabiliza el proceso de entrenamiento y se facilita la convergencia al evitar oscilaciones bruscas que pueden surgir al entrenar con todo el conjunto de datos o con muestras individuales.

5.4.4. Configuración de evaluación

Como se explicó en la Sección 4.5, el procedimiento utilizado para evaluar la calidad del modelo es $KFCV$, con $k = 5$, una elección habitual en este tipo de pruebas.

Por otro lado, para el cálculo de las métricas de evaluación de ranking, se ha decidido evaluar rankings de top-5, top-10 y top-15, esto nos permitirá comprobar la capacidad del modelo para priorizar correctamente los ítems más relevantes entre sus primeras sugerencias. Esta comparación ofrece una visión más completa del comportamiento del sistema de recomendación en escenarios con diferentes requisitos de precisión y cobertura.

Puede darse el caso de que en alguno de los conjuntos de datos, el número de interacciones de un usuario h_u sea menor que el número de elementos seleccionados para formar el top-K de recomendaciones, si esto ocurre, entonces el valor de K cambiará a:

$$K = min(K, h_u), \tag{56}$$

esta condición afecta sobre todo a la métrica Precision@K, pues es la única que depende exclusivamente de h_u, como se puede comprobar en su definición en la sección 2.5.3.

El valor de r_{min} y r_{max}, que forman el rango de valores en los que se puede

encontrar la predicción $[r_{min}, r_{max}]$, se establecerá utilizando los valores mínimo y máximo respectivamente de todas las valoraciones que se encuentren en el conjunto de datos seleccionado.

Además, el umbral τ, que determina el valor mínimo de *rating* en el que el ítem se considera recomendable para el usuario, se ha establecido como la media global del conjunto R de todos los *ratings* disponibles en el *dataset*, matemáticamente hablando, esto se define como:

$$\tau = \frac{1}{|R|} \sum_{r_{ui} \in R} r_{ui}. \tag{57}$$

Cabe destacar que no existe una regla general para la elección de este parámetro, el hecho de establecerlo esta regla ha sido para, de alguna manera, automatizar la elección de este valor a la hora de realizar las pruebas de evaluación para los diferentes conjuntos de datos.

Para finalizar, se han establecido distintas semillas (*seeds*) para los algoritmos no deterministas implementados en *Surprise* [27], como *SVD* o *NMF*. El objetivo es analizar si los resultados obtenidos por estos modelos dependen en gran medida del valor de la semilla utilizada. Para ello, se han realizado pruebas con los valores 0, 1 y 42, que son algunos de los más comúnmente empleados.

5.4.5. Configuración de *hardware*

Todas estas pruebas han sido evaluadas utilizando *GPU NVIDIA GeForce GTX 1060* con 6 GB de *VRAM* y una *CPU Intel I5 6800k*. Además, el sistema contaba con 16 GB de memoria *RAM* y 2 TB de almacenamiento.

6. Resultados y Discusión

Esta sección presenta un análisis de los resultados experimentales obtenidos. El estudio se estructura en dos fases complementarias que permitirán establecer conclusiones sólidas sobre el rendimiento de los diferentes enfoques evaluados.

En primer lugar, se examinarán detalladamente los resultados de evaluación del modelo propuesto, analizando tanto su comportamiento general como sus características específicas en diferentes escenarios de prueba. Paralelamente, se realizará una evaluación comparativa rigurosa aplicando las mismas métricas y protocolos de prueba a los algoritmos de referencia descritos en la Sección 4.5.3. Esta evaluación se llevará a cabo sobre los dos conjuntos de datos caracterizados en la Sección 4.2, garantizando así la consistencia metodológica y la validez de las comparaciones.

La segunda fase del análisis consistirá en la aplicación de pruebas estadísticas apropiadas que permitan determinar, con rigor científico, qué modelos ofrecen un rendimiento superior para cada conjunto de datos específico. Este análisis estadístico proporcionará la base empírica necesaria para establecer conclusiones definitivas sobre la efectividad relativa de cada enfoque e identificar los contextos en los que cada algoritmo demuestra sus fortalezas particulares.

6.1. Resultados experimentales

En esta sección se recogen las métricas obtenidas tras la evaluación del modelo propuesto sobre los dos conjuntos de datos empleados: *MARS* [15] e *ITM-Rec* [50]. Para cada *dataset* se muestran los resultados obtenidos al recomendar listas de tamaño $K = \{5, 10, 15\}$.

6.1.1. Resultados del modelo propuesto

La Tabla 10 resume las métricas de predicción y *ranking* alcanzadas sobre el conjunto de datos *MARS*. Cada columna corresponde a uno de los tamaños de lista

evaluados.

Métrica	**Top-5**	**Top-10**	**Top-15**
MSE	0.4002 ± 0.2589	0.2902 ± 0.0352	0.4376 ± 0.2534
RMSE	0.6062 ± 0.1748	0.5324 ± 0.0312	0.6373 ± 0.1667
Precision	0.7406 ± 0.0029	0.7324 ± 0.0034	0.7301 ± 0.0034
Recall	0.7429 ± 0.0037	0.7758 ± 0.0044	0.7860 ± 0.0045
F_1	0.7418 ± 0.0029	0.7535 ± 0.0036	0.7570 ± 0.0037
HR	0.7985 ± 0.0044	0.7985 ± 0.0044	0.7985 ± 0.0044
MAP	0.7945 ± 0.0040	0.7954 ± 0.0033	0.7934 ± 0.0064
MRR	0.7949 ± 0.0039	0.7958 ± 0.0034	0.7943 ± 0.0058
NDCG	0.7951 ± 0.0037	0.7962 ± 0.0037	0.7948 ± 0.0060

Tabla 10: Métricas obtenidas para el conjunto de datos *MARS*.

Los errores de predicción (*MSE* y *RMSE*) son menores en el *top*-10, lo que indica mayor fiabilidad a la hora de estimar las puntuaciones cuando la lista contiene diez elementos. El *top*-5 presenta mayor error y varianza, mientras que en el *top*-15 los errores vuelven a aumentar debido, previsiblemente, a la inclusión de ítems menos relevantes.

El F_1-*score* crece de forma estable con el tamaño de la lista porque el *Recall* aumenta a costa de una ligera pérdida de *Precision*. Este patrón es habitual cuando la lista de recomendaciones se amplía.

El resto de métricas de *ranking* confirman que el *top*-10 constituye el mejor compromiso entre relevancia y orden: *MAP*, *MRR* y *NDCG* alcanzan sus valores máximos en ese corte, mientras que *HR* se mantiene invariable en los tres escenarios.

Cómo se puede apreciar en la Tabla 11, que recoge las métricas obtenidas tras realizar los experimentos con el conjunto *ITM-Rec*, los valores de *MSE* y *RMSE* disminuyen conforme crece la longitud de la lista, señal de que el modelo es capaz de refinar sus predicciones cuando puede recomendar más cursos. Aun así, los errores son sensiblemente mayores que en *MARS*, lo que sugiere una menor correlación entre las características disponibles y la valoración de los usuarios, o bien que el modelo no captura todas las interacciones usuario–ítem.

Métrica	**Top-5**	**Top-10**	**Top-15**
MSE	1.5675 ± 0.1084	1.5183 ± 0.1365	1.4681 ± 0.0439
RMSE	1.2486 ± 0.0430	1.2288 ± 0.0571	1.2092 ± 0.0178
Precision	0.5551 ± 0.0111	0.5453 ± 0.0103	0.5448 ± 0.0102
Recall	0.7622 ± 0.0112	0.7719 ± 0.0109	0.7720 ± 0.0109
F_1	0.6423 ± 0.0110	0.6391 ± 0.0106	0.6388 ± 0.0105
HR	0.7720 ± 0.0109	0.7720 ± 0.0109	0.7720 ± 0.0109
MAP	0.7402 ± 0.0138	0.7406 ± 0.0138	0.7393 ± 0.0120
MRR	0.7477 ± 0.0149	0.7481 ± 0.0126	0.7456 ± 0.0111
NDCG	0.7480 ± 0.0128	0.7510 ± 0.0127	0.7502 ± 0.0111

Tabla 11: Métricas obtenidas para el conjunto de datos *ITM-Rec*.

Las métricas de *ranking* muestran un comportamiento muy parecido al observado en *MARS*. El índice *HR* permanece constante y los máximos de *MAP*, *MRR* y *NDCG* vuelven a aparecer en el *top*-10, aunque con valores algo más bajos. Se mantiene también el desequilibrio entre *Precision* y *Recall*, reflejado en un F_1-*score* estable pero inferior al del primer *dataset*.

Una vez analizados todos los resultados obtenidos por el modelo, podemos concluir que, en parte, las métricas de predicción son las que más afectadas se ven por

los cambios de tamaño en el *ranking* de recomendaciones, mientras que, por el otro lado, las métricas de *ranking* no se ven afectadas por este cambio de tamaño.

Para finalizar, podemos concluir que es con el conjunto de datos *MARS* con el que el modelo ofrece los mejores resultados, esto tiene sentido si tomamos en cuenta que es el conjunto de datos más grande de los tres evaluados y que, cómo ya se comentó en 4.2.1, cuenta con características fuertemente correlacionadas entre sí.

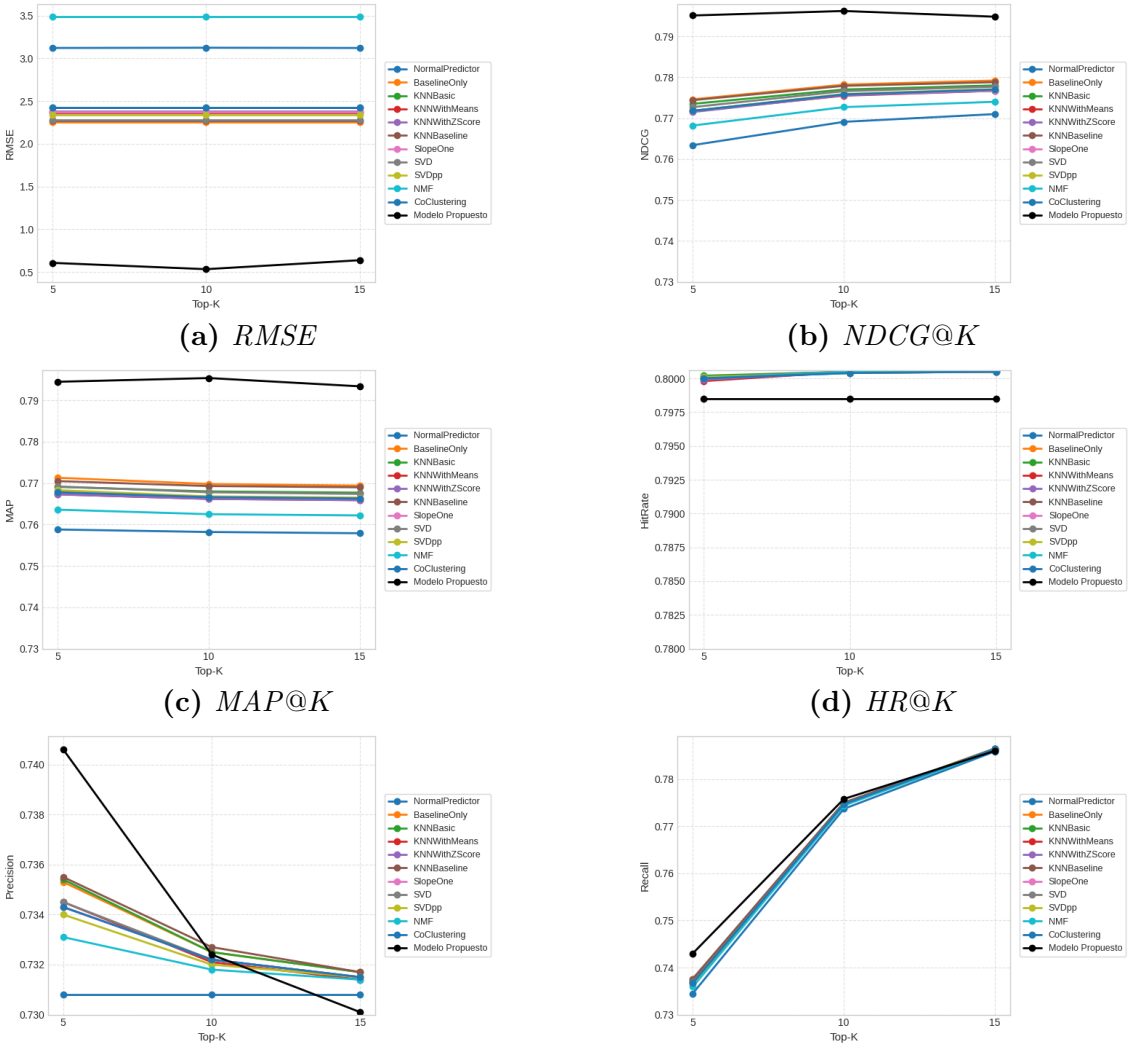
6.1.2. Comparación con otros modelos

En esta sección se presentan los resultados obtenidos tras evaluar los modelos implementados en la librería *Surprise* [27]. Las pruebas realizadas a estos modelos son equivalentes a las empleadas para evaluar el modelo propuesto, garantizando así que todos han sido analizados bajo las mismas condiciones experimentales.

Cabe destacar, que, para estas comparaciones, se ha decidido utilizar la media de los resultados obtenidos por los algoritmos basados en *SGD* para cada valor de semilla establecido para homogeneizar los resultados, pese a esto, en el repositorio de *Github* del trabajo se encuentran disponibles archivos *CSV* para cada uno de los resultados obtenidos. De entre todas las métricas disponibles, se han decidido estudiar aquellas que pueden ser más relevantes para evaluar sistemas de recomendación, aunque si se desea, se pueden consultar las Tablas correspondientes a dichas métricas en el Anexo A.

Para el conjunto de datos *MARS*, se puede observar gracias a la Figura 14 que los valores de *RMSE* obtenidos por el modelo propuesto son mínimos en casi todos los casos en comparación a los otros algoritmos, lo que nos indica que el modelo es capaz de predecir un valor de *rating* mucho más cercano al real.

Siguiendo la misma tendencia, el modelo obtiene los valores más altos en las métricas *NDCG@K* y *MAP@K*, lo que nos indica que, de entre todos los otros algoritmos, es el que consigue los mejores *rankigs* para los usuarios.



(a) *RMSE* **(b)** *NDCG@K*

(c) *MAP@K* **(d)** *HR@K*

(e) *Precision@K* **(f)** *Recall@K*

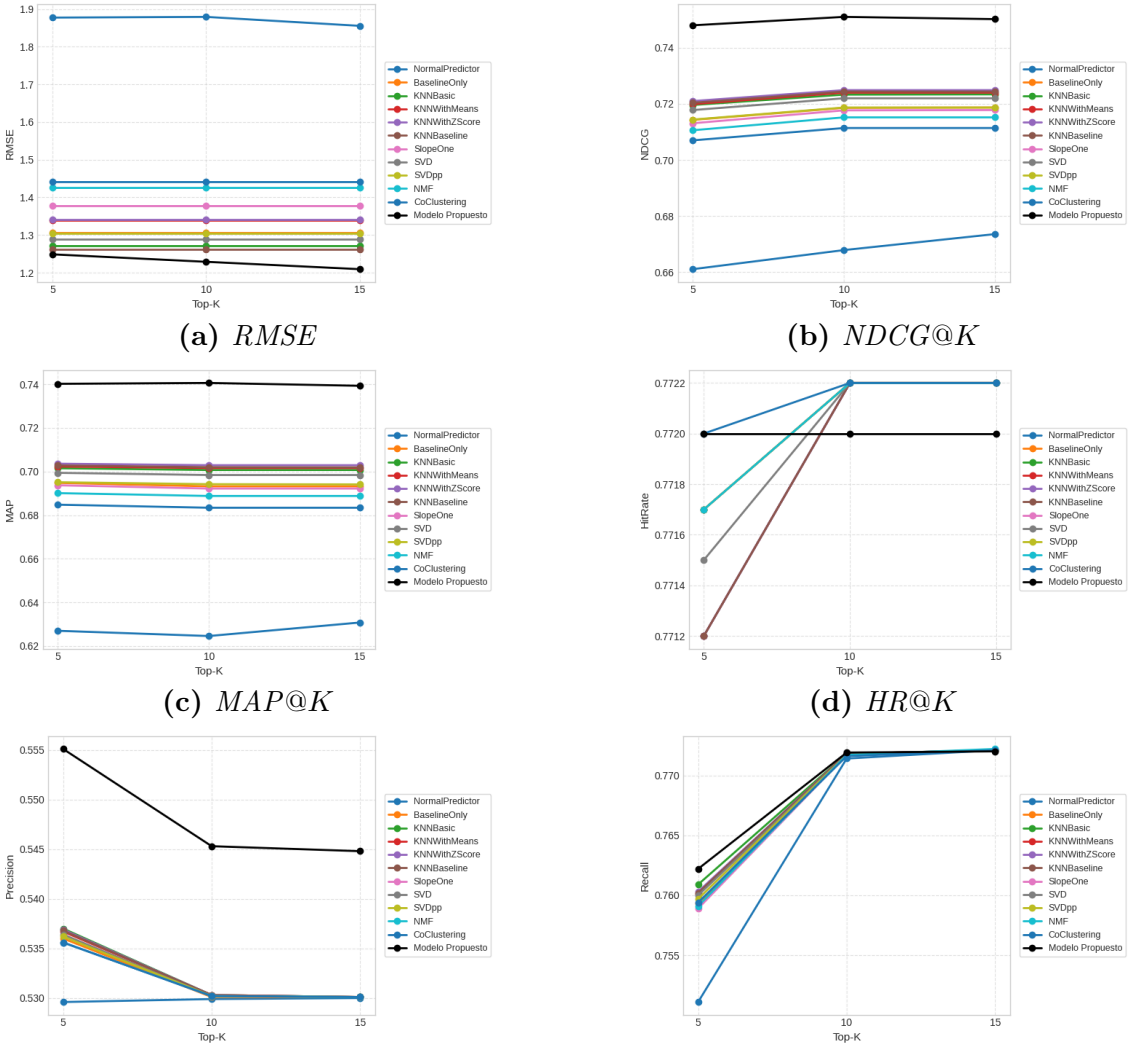
Figura 14: Comparación de métricas entre los algoritmos de estado del arte y el modelo propuesto (línea negra) en el *dataset MARS* para *top-5*, *top-10* y *top-15*.

La cosa cambia cuando observamos los valores de $Precision@K$ y $Recall@K$, pues todos los modelos acaban convergiendo a los mismos valores conforme aumentamos K, esto nos indica que, pese a diferir en algunas métricas, todos los modelos acaban consiguiendo más o menos la misma proporción de ítems relevantes en los *rankings*.

El caso de la métrica *HR* es el más interesante, pues podemos comprobar que el modelo propuesto ofrece un valor menor que el resto de modelos, aunque la diferencia es mínima (≈ 0.002), sin embargo, a diferencia del resto de algoritmos, este valor no aumenta conforme aumentamos K, es decir, se mantiene contante algo resalta bastante dada la naturaleza de *HR*. Este comportamiento sugiere que el modelo es capaz de identificar siempre al menos un ítem relevante en las primeras posiciones del ranking, pero también puede interpretarse como una limitación: el modelo podría estar acertando de forma temprana, pero no mejorando cuando se añaden más ítems al ranking, lo que justificaría su comportamiento.

Podemos observar gracias a la Figura 15 un comportamiento análogo al evaluar los modelos sobre el conjunto de datos *ITM-Rec*, el modelo vuelve a ser el que obtiene mejores resultados en casi todas las métricas, aunque, la gran mayoría de los resultados obtenidos son peores con respecto a los vistos en el *dataset* anterior.

El hecho de que ningún modelo mejore los resultados en este *dataset* en comparación con los obtenidos en el anterior puede deberse a diversos factores, como la falta de suficientes interacciones usuario-ítem, lo que hace que los modelos no tengan suficiente información para hacer recomendaciones precisas. Además, datos no representativos o sesgados pueden llevar a que los modelos no generalicen bien a nuevos datos o que solo recomienden ítems populares, ignorando ítems menos conocidos pero potencialmente relevantes. Esto puede tener sentido cuando comparamos el número de interacciones en el *dataset ITM-Rec* (5230) con respecto a las que tiene *MARS* (88998).



(a) *RMSE* **(b)** *NDCG@K*

(c) *MAP@K* **(d)** *HR@K*

(e) *Precision@K* **(f)** *Recall@K*

Figura 15: Comparación de métricas entre los algoritmos de estado del arte y el modelo propuesto (línea negra) en el *dataset ITM-Rec* para *top-5*, *top-10* y *top-15*.

6.2. Resultados de las pruebas estadísticas

Una vez obtenidos todos los resultados de las comparaciones, es momento de realizar tests estadísticos para determinar si realmente existe una diferencia de rendimiento entre los modelos evaluados.

Como se explicó anteriormente en la Sección 4.5.2, se han analizado los resultados usando en la prueba de Friedman, seguido de la prueba *post-hoc* de *Nemenyi* para visualizar de una manera clara si realmente existe una diferencia entre los resultados de los modelos a trabes del *CDD*. El valor umbral establecido para rechazar la hipótesis nula es $\alpha = 0.005$, un valor estándar en casi todos los estudios que se realizan.

Las pruebas buscan comprobar cuál de todos los modelos evaluados consiguen mejores puntuaciones en las diferentes métricas que se han calculado en los procesos de evaluación para cada valor de K. Por ende, podemos definir H_0 y H_1 de la siguiente manera:

- Hipótesis nula H_0: Todos los modelos tienen el mismo rendimiento medio para cada una de las métricas.
- Hipótesis nula H_1: Al menos un par de modelos difieren en el valor de las métricas obtenidas.

Dataset	**Estadísticos**	$K = 5$	$K = 10$	$K = 15$
MARS	*p-value*	1.2975×10^{-15}	2.8384×10^{-14}	2.5752×10^{-13}
	Q	112.0636	104.8503	99.6426
ITM-Rec	*p-value*	2.5242×10^{-14}	3.6050×10^{-15}	9.5196×10^{-9}
	Q	105.1261	109.6833	73.9685

Tabla 12: Resultados de las pruebas de *Friedman* para $K = \{5, 10, 15\}$.

Cómo se puede comprobar en la Tabla 12, al cumplirse que ($p\text{-}value < \alpha$) en todos los resultados para cada K, podemos concluir que ambos en todos los modelos

existe una gran diferencia entre los valores de las métricas obtenidos para cada uno, por lo que rechazamos la hipótesis nula H_0 y aceptamos H_1.

Como hemos aceptado H_0, a continuación se realizarán la prueba *post-hoc* de *Nemenyi*, cómo se explicó en la Sección 4.5.2, esta prueba nos permitirá obtener el *CDD* para poder visualizar de una mejor manera las diferencias entre cada uno de los modelos. Las Figuras 16 y 17 muestran estos diagramas para cada conjunto de datos evaluado.

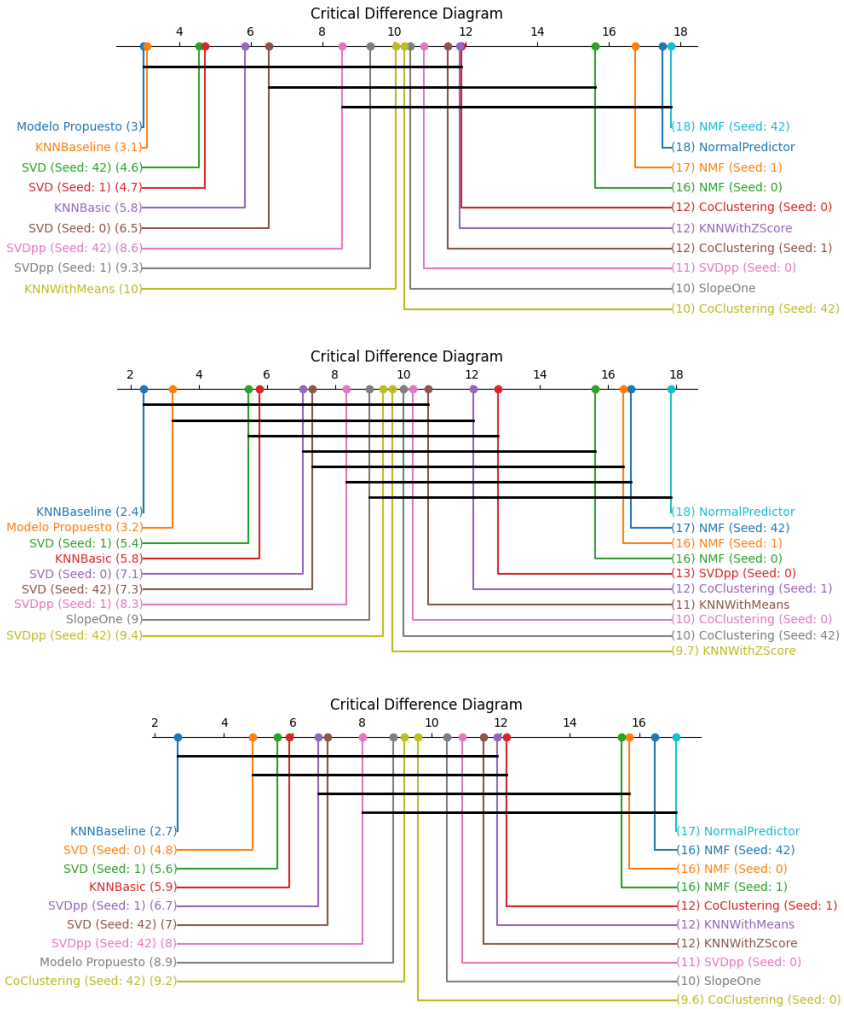


Figura 16: *CDD* para cada el conjunto de datos *MARS* por cada valor de K.

En este caso, los resultados muestran algo curioso, pues conforme aumentamos K, el modelo propuesto parece obtener menos puntuación, lo que sugiere que el modelo alcanza su máximo rendimiento en configuraciones de *top-K* reducido, donde la precisión de las recomendaciones resulta más elevada. A medida que se amplía el número de elementos considerados en el *ranking*, la capacidad del modelo para mantener la calidad de sus predicciones se ve comprometida, reflejándose en puntuaciones estadísticamente inferiores según los tests de diferencias críticas aplicados.

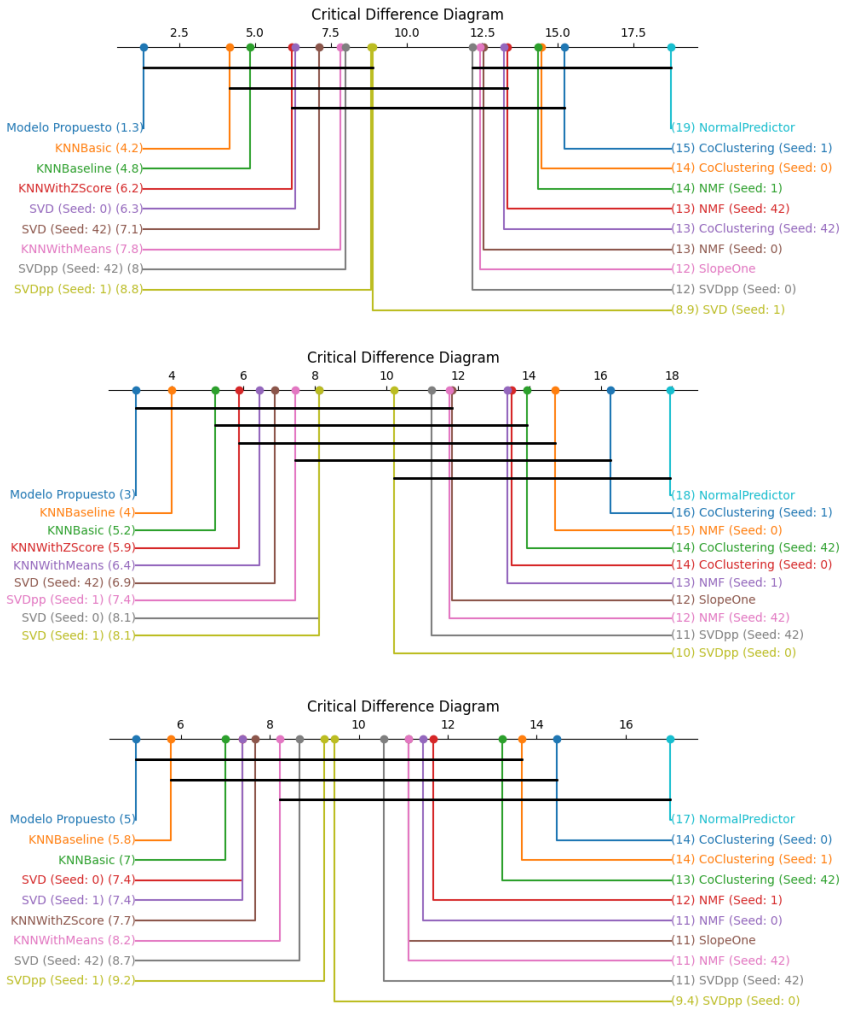


Figura 17: *CDD* para cada el conjunto de datos *ITM-Rec* por cada valor de K.

La situación parece cambiar cuando evaluamos las métricas obtenidas con el *da-*

taset ITM-Rec, en este caso, el modelo propuesto se establece como la mejor opción para cada valor de K, lo que sugiere que el modelo es capaz de obtener mejores recomendaciones en todos los casos, sin importar el valor de K, algo que también podemos visualizar en la Figura 15.

Esta variabilidad en el rendimiento puede significar que el modelo tiene una gran capacidad de adaptación a diferentes características de datos, posicionándolo como una alternativa interesante a la hora de ser utilizado en un entorno de producción.

7. Conclusiones y Recomendaciones

Este *TFG* ha logrado desarrollar y evaluar exitosamente un sistema de recomendación híbrido especializado para plataformas de *e-learning*, cumpliendo de manera integral con los objetivos planteados al inicio de la investigación. El trabajo ha demostrado la viabilidad de mejorar significativamente la personalización del aprendizaje en entornos educativos digitales mediante la implementación de técnicas avanzadas de recomendación.

Se ha realizado una revisión exhaustiva de las investigaciones más relevantes y actuales en el campo de los sistemas de recomendación. Esta fundamentación teórica ha proporcionado las bases necesarias para identificar las oportunidades de mejora y las direcciones más prometedoras para el desarrollo del modelo propuesto.

La selección cuidadosa de datasets públicos representativos del dominio educativo ha permitido validar la efectividad del sistema desarrollado bajo condiciones realistas.

Se ha desarrollado con éxito una nueva propuesta basada en un modelo híbrido robusto que combina eficazmente diferentes paradigmas de los sistemas de recomendación. La arquitectura resultante ha demostrado funcionar de manera exitosa en varios conjuntos de datos, especialmente valiosa en el contexto este contexto, donde la disponibilidad de datos puede ser limitada.

La evaluación experimental ha revelado un comportamiento diferenciado del modelo según las características específicas de los conjuntos de datos utilizados, demostrando su capacidad de adaptación a diversos contextos educativos. Pese a en ocasiones no obtener los mejores resultados, el modelo ha demostrado ser capaz de generar recomendaciones exitosas con respecto al resto de implementaciones.

Es importante reconocer que el sistema desarrollado aún presenta oportunidades

de mejora, pues no se ha podido someter a procesos de calibración de hiperparámetros que permitan obtener la mejor versión de este, lo que quizás pueda suponer una mejora en los resultados de las evaluaciones.

No obstante, los resultados obtenidos establecen una base sólida que confirma la validez del enfoque adoptado y sugiere un potencial considerable para futuras mejoras. El modelo desarrollado representa una contribución valiosa al campo de los sistemas de recomendación educativos, ofreciendo una alternativa competitiva que combina eficacia en la personalización del aprendizaje con robustez técnica demostrada.

7.1. Futuras mejoras

Pese a que, en general, se han conseguido cumplir todos los objetivos para este *TFG*, aún existen muchas mejoras que se pueden realizar en el futuro y que pueden significar un gran avance para el desarrollo del sistema de recomendación.

La implementación de técnicas de optimización de hiperparámetros constituye una de las mejoras más inmediatas y prometedoras. El desarrollo de procesos sistemáticos de calibración mediante técnicas como búsqueda en cuadrícula (*Grid Search*) o algoritmos evolutivos podría mejorar sustancialmente el rendimiento del modelo. Esta optimización permitiría explorar de manera exhaustiva el espacio de parámetros e identificar configuraciones óptimas que permitan al modelo realizar mejores predicciones.

Por otro lado, la falta de procesamiento de variables textuales como descripciones o nombres es sin duda una de las grandes mejoras que se le deben realizar al modelo, pues de esta manera el modelo podrá tener mucho más contexto de los ítems a recomendar, lo que podría mejorar el rendimiento de este, a esto se le pueden añadir otras características como las temporales, lo que pueden conseguir que el sistema tenga en cuenta aquellas valoraciones más recientes.

El desarrollo de interfaces de usuario más intuitivas y funcionalidades podría ser una mejora significativa en cuanto a experiencia del usuario se refiere. También permitirían comprobar que tal se comportaría el sistema en un entorno real.

Estas mejoras futuras no solo representan oportunidades para el perfeccionamiento técnico del sistema, sino que también abren nuevas posibilidades para contribuir de manera más significativa al avance de la personalización educativa y al desarrollo de tecnologías de aprendizaje más efectivas y centradas en el estudiante.

Bibliografía

[1] Ahmadian Yazdi, H., Seyyed Mahdavi, S. J., and Ahmadian Yazdi, H. (2024). Dynamic educational recommender system based on improved lstm neural network. *Scientific Reports*, 14(1):4381.

[2] Astral SH (2024). uv.

[3] Croon, R. D., Houdt, L. V., Htun, N. N., Štiglic, G., Abeele, V. V., and Verbert, K. (2019). Health recommender systems: Systematic review.

[4] da Silva, F. L., Slodkowski, B. K., da Silva, K. K. A., and Cazella, S. C. (2023). A systematic literature review on educational recommender systems for teaching and learning: research trends, limitations and opportunities. *Education and Information Technologies*, 28(3):3289–3328.

[5] Detlefsen, N. S., Borovec, J., Schock, J., Jha, A. H., Koker, T., Liello, L. D., Stancl, D., Quan, C., Grechkin, M., and Falcon, W. (2022). Torchmetrics - measuring reproducibility in pytorch. *Journal of Open Source Software*, 7(70):4101.

[6] Do, H.-Q., Le, T.-H., and Yoon, B. (2020). Dynamic weighted hybrid recommender systems. In *2020 22nd International Conference on Advanced Communication Technology (ICACT)*, pages 644–650.

[7] Dol Aher, S. and Lobo, L. M. R. (2012). A comparative study of association rule algorithms for course recommender system in e-learning. *International Journal of Computer Applications*, 39:48–52.

[8] Falcon, W. and The PyTorch Lightning team (2019). PyTorch Lightning.

[9] Ferreira, D., Silva, S., Abelha, A., and Machado, J. (2020). Recommendation system using autoencoders. *Applied Sciences*, 10:5510.

[10] George, T. and Merugu, S. (2005). A scalable collaborative filtering framework based on co-clustering. pages 4 pp.–.

[11] Gomez-Uribe, C. A. and Hunt, N. (2016). The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4).

[12] Guo, H., TANG, R., Ye, Y., Li, Z., and He, X. (2017). Deepfm: A factorization-machine based neural network for ctr prediction. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 1725–1731.

[13] Guruge, D. B., Kadel, R., and Halder, S. J. (2021). The state of the art in methodologies of course recommender systems—a review of recent research. *Data*, 6(2).

[14] Hafsa, M., Wattebled, P., Jacques, J., and Jourdan, L. (2022). A Multi-Objective E-learning Recommender System at Mandarine Academy. In *Proceedings of the 2nd Workshop on Multi-Objective Recommender Systems co-located with 16th ACM Conference on Recommender Systems (RecSys 2022)*, Seatlle (virtual), United States.

[15] Hafsa, M., Wattebled, P., Jacques, J., and Jourdan, L. (2023). E-learning recommender system dataset.

[16] Hardesty, L. (2021). The history of amazon's recommendation algorithm.

[17] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

[18] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. (2017). Neural collaborative filtering.

[19] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors.

[20] Hssina, B., Grota, A., and Erritali, M. (2021). Recommendation system using the k-nearest neighbors and singular value decomposition algorithms. *International Journal of Electrical and Computer Engineering (IJECE)*, 11:5541.

[21] Hua, W., Li, L., Xu, S., Chen, L., and Zhang, Y. (2023). Tutorial on large language models for recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems*, pages 1281–1283.

[22] Hug, N. Basic algorithms 2014; Surprise 1 documentation — surprise.readthedocs.io. [Accessed 02-06-2025].

[23] Hug, N. Co-clustering algorithms 2014; Surprise 1 documentation — surprise.readthedocs.io. [Accessed 02-06-2025].

[24] Hug, N. KNN inspired algorithms 2014; Surprise 1 documentation — surprise.readthedocs.io. [Accessed 02-06-2025].

[25] Hug, N. Matrix factorization-based algorithms 2014; Surprise 1 documentation — surprise.readthedocs.io. [Accessed 02-06-2025].

[26] Hug, N. Slope one-based algorithms 2014; Surprise 1 documentation — surprise.readthedocs.io. [Accessed 02-06-2025].

[27] Hug, N. (2020). Surprise: A python library for recommender systems. *Journal of Open Source Software*, 5(52):2174.

[28] Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

[29] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

[30] Jadon, A. and Patil, A. (2024). A comprehensive survey of evaluation techniques for recommendation systems.

[31] Khanal, S. S., Prasad, P. W. C., Alsadoon, A., and Maag, A. (2020). A systematic review: machine learning based recommendation systems for e-learning. *Education and Information Technologies*, 25(4):2635–2664.

[32] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

[33] Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

[34] Lemire, D. and Maclachlan, A. (2008). Slope one predictors for online rating-based collaborative filtering.

[35] Lin, W., Alvarez, S., and Ruiz, C. (2002). Efficient adaptive-support association rule mining for recommender systems. *Data Min. Knowl. Discov.*, 6:83–105.

[36] Ma, Y., Ouyang, R., Long, X., Gao, Z., Lai, T., and Fan, C. (2023). Doris: Personalized course recommendation system based on deep learning. *PLOS ONE*, 18(6):e0284687.

[37] Martínez-Plumed, F., Contreras-Ochando, L., Ferri, C., Hernández-Orallo, J., Kull, M., Lachiche, N., Ramírez-Quintana, M. J., and Flach, P. (2021). Crisp-dm twenty years later: From data mining processes to data science trajectories. *IEEE Transactions on Knowledge and Data Engineering*, 33(8):3048–3061.

[38] Miura, K. (2011). An introduction to maximum likelihood estimation and information geometry. *Interdisciplinary Information Sciences (IIS)*, 17.

[39] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library.

[40] Patel, R., Thakkar, P., and Ukani, V. (2024). Cnnrec: Convolutional neural network based recommender systems - a survey. *Engineering Applications of Artificial Intelligence*, 133:108062.

[41] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[42] Python Core Team (2023). *Python: A dynamic, open source programming language*. Python Software Foundation. Python version 3.12.

[43] Salakhutdinov, R. and Mnih, A. (2007). Probabilistic matrix factorization. In *Proceedings of the 21st International Conference on Neural Information Processing Systems*, NIPS'07, page 1257–1264, Red Hook, NY, USA. Curran Associates Inc.

[44] Salau, L., Hamada, M., Prasad, R., Hassan, M., Mahendran, A., and Watanobe, Y. (2022). State-of-the-art survey on deep learning-based recommender systems for e-learning. *Applied Sciences*, 12(23).

[45] Sami, A., Adrousy, W. E., Sarhan, S., and Elmougy, S. (2024). A deep learning based hybrid recommendation model for internet users. *Scientific Reports*, 14(1):29390.

[46] Steck, H., Baltrunas, L., Elahi, E., Liang, D., Raimond, Y., and Basilico, J. (2021). Deep learning for recommender systems: A netflix case study. *AI Magazine*, 42(3):7–18.

[47] Uta, M., Felfernig, A., Le, V.-M., Tran, T. N. T., Garber, D., Lubos, S., and Burgstaller, T. (2024). Knowledge-based recommender systems: overview and research directions. *Frontiers in Big Data*, Volume 7 - 2024.

[48] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A.,

Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

[49] Wes McKinney (2010). Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61.

[50] Zheng, Y. (2023). Itm-rec: An open data set for educational recommender systems.

[51] Çakır, M., Gunduz Oguducu, S., and Tugay, R. (2019). *A Deep Hybrid Model for Recommendation Systems*, pages 321–335.

A. Tablas de resultados

Dataset	**Modelo**	$K = 5$	$K = 10$	$K = 15$
MARS	NormalPredictor	9.7417 ± 0.1060	9.7593 ± 0.1452	9.7402 ± 0.1098
	BaselineOnly	5.0638 ± 0.0625	5.0638 ± 0.0625	5.0638 ± 0.0625
	KNNBasic	5.8896 ± 0.1354	5.8896 ± 0.1354	5.8896 ± 0.1354
	KNNWithMeans	5.5905 ± 0.0625	5.5815 ± 0.0605	5.5915 ± 0.0620
	KNNWithZScore	5.6906 ± 0.0607	5.6906 ± 0.0607	5.6906 ± 0.0607
	KNNBaseline	5.1420 ± 0.0811	5.1420 ± 0.0811	5.1420 ± 0.0811
	SlopeOne	5.6149 ± 0.0723	5.6149 ± 0.0723	5.6149 ± 0.0723
	SVD	5.1861 ± 0.0088	5.1861 ± 0.0088	5.1861 ± 0.0088
	SVDpp	5.4565 ± 0.0217	5.4565 ± 0.0217	5.4565 ± 0.0217
	NMF	12.1441 ± 0.0483	12.1441 ± 0.0483	12.1441 ± 0.0483
	CoClustering	5.8700 ± 0.0264	5.8700 ± 0.0264	5.8700 ± 0.0264
	Modelo Propuesto	**0.4002 ± 0.2589**	**0.2902 ± 0.0352**	**0.4376 ± 0.2534**
ITM-Rec	NormalPredictor	3.5264 ± 0.1738	3.5334 ± 0.1699	3.4437 ± 0.1226
	BaselineOnly	1.7067 ± 0.0620	1.7067 ± 0.0620	1.7067 ± 0.0620
	KNNBasic	1.6184 ± 0.0847	1.6184 ± 0.0847	1.6184 ± 0.0847
	KNNWithMeans	1.7944 ± 0.0918	1.7944 ± 0.0918	1.7944 ± 0.0918
	KNNWithZScore	1.7992 ± 0.0798	1.7992 ± 0.0798	1.7992 ± 0.0798
	KNNBaseline	1.5906 ± 0.0766	1.5906 ± 0.0766	1.5906 ± 0.0766
	SlopeOne	1.8991 ± 0.0823	1.8991 ± 0.0823	1.8991 ± 0.0823
	SVD	1.6612 ± 0.0139	1.6612 ± 0.0139	1.6612 ± 0.0139
	SVDpp	1.7031 ± 0.0161	1.7031 ± 0.0161	1.7031 ± 0.0161
	NMF	2.0327 ± 0.0216	2.0327 ± 0.0216	2.0327 ± 0.0216
	CoClustering	2.0775 ± 0.0090	2.0775 ± 0.0090	2.0775 ± 0.0090
	Modelo Propuesto	**1.5675 ± 0.1084**	**1.5183 ± 0.1365**	**1.4681 ± 0.0439**

Tabla 13: Comparación de *MSE* para el modelo propuesto para $K = \{5, 10, 15\}$.

Dataset	**Modelo**	$K = 5$	$K = 10$	$K = 15$
MARS	NormalPredictor	3.1211 ± 0.0170	3.1239 ± 0.0232	3.1209 ± 0.0176
	BaselineOnly	2.2503 ± 0.0139	2.2503 ± 0.0139	2.2503 ± 0.0139
	KNNBasic	2.4267 ± 0.0278	2.4267 ± 0.0278	2.4267 ± 0.0278
	KNNWithMeans	2.3644 ± 0.0132	2.3644 ± 0.0132	2.3644 ± 0.0132
	KNNWithZScore	2.3855 ± 0.0127	2.3855 ± 0.0127	2.3855 ± 0.0127
	KNNBaseline	2.2675 ± 0.0178	2.2675 ± 0.0178	2.2675 ± 0.0178
	SlopeOne	2.3695 ± 0.0153	2.3695 ± 0.0153	2.3695 ± 0.0153
	SVD	2.2773 ± 0.0019	2.2773 ± 0.0019	2.2773 ± 0.0019
	SVDpp	2.3359 ± 0.0047	2.3359 ± 0.0047	2.3359 ± 0.0047
	NMF	3.4848 ± 0.0070	3.4848 ± 0.0070	3.4848 ± 0.0070
	CoClustering	2.4228 ± 0.0054	2.4228 ± 0.0054	2.4228 ± 0.0054
	Modelo Propuesto	**0.6062 ± 0.1748**	**0.5324 ± 0.0312**	**0.6373 ± 0.1667**
ITM-Rec	NormalPredictor	1.8774 ± 0.0460	1.8793 ± 0.0451	1.8555 ± 0.0331
	BaselineOnly	1.3062 ± 0.0239	1.3062 ± 0.0239	1.3062 ± 0.0239
	KNNBasic	1.2718 ± 0.0338	1.2718 ± 0.0338	1.2718 ± 0.0338
	KNNWithMeans	1.3392 ± 0.0345	1.3392 ± 0.0345	1.3392 ± 0.0345
	KNNWithZScore	1.3411 ± 0.0299	1.3411 ± 0.0299	1.3411 ± 0.0299
	KNNBaseline	1.2609 ± 0.0308	1.2609 ± 0.0308	1.2609 ± 0.0308
	SlopeOne	1.3778 ± 0.0298	1.3778 ± 0.0298	1.3778 ± 0.0298
	SVD	1.2886 ± 0.0056	1.2886 ± 0.0056	1.2886 ± 0.0056
	SVDpp	1.3049 ± 0.0062	1.3049 ± 0.0062	1.3049 ± 0.0062
	NMF	1.4252 ± 0.0078	1.4252 ± 0.0078	1.4252 ± 0.0078
	CoClustering	1.4410 ± 0.0030	1.4410 ± 0.0030	1.4410 ± 0.0030
	Modelo Propuesto	**1.2486 ± 0.0430**	**1.2288 ± 0.0571**	**1.2092 ± 0.0178**

Tabla 14: Comparación de *RMSE* para el modelo propuesto para $K = \{5, 10, 15\}$.

Dataset	**Modelo**	$K = 5$	$K = 10$	$K = 15$
MARS	NormalPredictor	0.7634 ± 0.0094	0.7691 ± 0.0087	0.7710 ± 0.0092
	BaselineOnly	0.7746 ± 0.0086	0.7782 ± 0.0086	0.7792 ± 0.0086
	KNNBasic	0.7735 ± 0.0095	0.7770 ± 0.0093	0.7780 ± 0.0094
	KNNWithMeans	0.7717 ± 0.0092	0.7755 ± 0.0092	0.7767 ± 0.0092
	KNNWithZScore	0.7715 ± 0.0093	0.7756 ± 0.0093	0.7767 ± 0.0093
	KNNBaseline	0.7744 ± 0.0090	0.7779 ± 0.0090	0.7788 ± 0.0090
	SlopeOne	0.7718 ± 0.0093	0.7758 ± 0.0092	0.7768 ± 0.0093
	SVD	0.7727 ± 0.0002	0.7766 ± 0.0001	0.7777 ± 0.0001
	SVDpp	0.7718 ± 0.0003	0.7759 ± 0.0004	0.7771 ± 0.0003
	NMF	0.7682 ± 0.0005	0.7727 ± 0.0005	0.7740 ± 0.0005
	CoClustering	0.7718 ± 0.0002	0.7758 ± 0.0001	0.7770 ± 0.0001
	Modelo Propuesto	**0.7951 ± 0.0037**	**0.7962 ± 0.0037**	**0.7948 ± 0.0060**
ITM-Rec	NormalPredictor	0.6611 ± 0.0306	0.6679 ± 0.0267	0.6736 ± 0.0203
	BaselineOnly	0.7143 ± 0.0239	0.7186 ± 0.0230	0.7187 ± 0.0231
	KNNBasic	0.7196 ± 0.0231	0.7233 ± 0.0218	0.7234 ± 0.0219
	KNNWithMeans	0.7199 ± 0.0226	0.7240 ± 0.0212	0.7240 ± 0.0212
	KNNWithZScore	0.7210 ± 0.0217	0.7249 ± 0.0205	0.7249 ± 0.0205
	KNNBaseline	0.7204 ± 0.0220	0.7244 ± 0.0209	0.7244 ± 0.0209
	SlopeOne	0.7131 ± 0.0242	0.7177 ± 0.0231	0.7178 ± 0.0231
	SVD	0.7178 ± 0.0016	0.7220 ± 0.0017	0.7220 ± 0.0016
	SVDpp	0.7144 ± 0.0017	0.7187 ± 0.0016	0.7187 ± 0.0016
	NMF	0.7106 ± 0.0014	0.7152 ± 0.0017	0.7152 ± 0.0017
	CoClustering	0.7070 ± 0.0027	0.7114 ± 0.0025	0.7114 ± 0.0025
	Modelo Propuesto	**0.7480 ± 0.0128**	**0.7510 ± 0.0127**	**0.7502 ± 0.0111**

Tabla 15: Comparación de *NDCG* para el modelo propuesto para $K = \{5, 10, 15\}$.

Dataset	**Modelo**	$K = 5$	$K = 10$	$K = 15$
MARS	NormalPredictor	0.7588 ± 0.0087	0.7582 ± 0.0083	0.7579 ± 0.0093
	BaselineOnly	0.7713 ± 0.0081	0.7698 ± 0.0083	0.7694 ± 0.0082
	KNNBasic	0.7691 ± 0.0096	0.7680 ± 0.0096	0.7677 ± 0.0096
	KNNWithMeans	0.7673 ± 0.0097	0.7662 ± 0.0095	0.7659 ± 0.0095
	KNNWithZScore	0.7673 ± 0.0096	0.7662 ± 0.0097	0.7659 ± 0.0096
	KNNBaseline	0.7705 ± 0.0087	0.7693 ± 0.0088	0.7690 ± 0.0088
	SlopeOne	0.7678 ± 0.0090	0.7666 ± 0.0092	0.7663 ± 0.0091
	SVD	0.7692 ± 0.0001	0.7678 ± 0.0002	0.7674 ± 0.0001
	SVDpp	0.7683 ± 0.0005	0.7668 ± 0.0004	0.7665 ± 0.0004
	NMF	0.7636 ± 0.0007	0.7625 ± 0.0006	0.7622 ± 0.0006
	CoClustering	0.7678 ± 0.0003	0.7666 ± 0.0003	0.7663 ± 0.0003
	Modelo Propuesto	**0.7945 ± 0.0040**	**0.7954 ± 0.0033**	**0.7934 ± 0.0064**
ITM-Rec	NormalPredictor	0.6270 ± 0.0316	0.6246 ± 0.0304	0.6308 ± 0.0208
	BaselineOnly	0.6948 ± 0.0232	0.6932 ± 0.0238	0.6932 ± 0.0237
	KNNBasic	0.7015 ± 0.0219	0.7007 ± 0.0223	0.7007 ± 0.0223
	KNNWithMeans	0.7023 ± 0.0212	0.7016 ± 0.0216	0.7016 ± 0.0216
	KNNWithZScore	0.7036 ± 0.0199	0.7029 ± 0.0202	0.7029 ± 0.0202
	KNNBaseline	0.7028 ± 0.0204	0.7020 ± 0.0209	0.7019 ± 0.0209
	SlopeOne	0.6937 ± 0.0230	0.6922 ± 0.0238	0.6922 ± 0.0238
	SVD	0.6994 ± 0.0020	0.6984 ± 0.0020	0.6984 ± 0.0020
	SVDpp	0.6951 ± 0.0016	0.6942 ± 0.0019	0.6941 ± 0.0019
	NMF	0.6901 ± 0.0024	0.6888 ± 0.0022	0.6888 ± 0.0022
	CoClustering	0.6848 ± 0.0035	0.6834 ± 0.0034	0.6834 ± 0.0034
	Modelo Propuesto	**0.7402 ± 0.0138**	**0.7406 ± 0.0138**	**0.7393 ± 0.0120**

Tabla 16: Comparación de *MAP* para el modelo propuesto para $K = \{5, 10, 15\}$.

Dataset	**Modelo**	$K = 5$	$K = 10$	$K = 15$
MARS	NormalPredictor	0.7630 ± 0.0087	0.7639 ± 0.0081	0.7639 ± 0.0093
	BaselineOnly	0.7756 ± 0.0079	0.7756 ± 0.0079	0.7756 ± 0.0079
	KNNBasic	0.7731 ± 0.0092	0.7733 ± 0.0094	0.7733 ± 0.0094
	KNNWithMeans	0.7707 ± 0.0095	0.7709 ± 0.0095	0.7709 ± 0.0095
	KNNWithZScore	0.7710 ± 0.0099	0.7711 ± 0.0099	0.7710 ± 0.0100
	KNNBaseline	0.7744 ± 0.0085	0.7748 ± 0.0086	0.7748 ± 0.0086
	SlopeOne	0.7714 ± 0.0092	0.7716 ± 0.0093	0.7716 ± 0.0093
	SVD	0.7734 ± 0.0004	0.7734 ± 0.0005	0.7734 ± 0.0006
	SVDpp	0.7724 ± 0.0003	0.7725 ± 0.0002	0.7725 ± 0.0002
	NMF	0.7675 ± 0.0010	0.7676 ± 0.0009	0.7676 ± 0.0009
	CoClustering	0.7716 ± 0.0004	0.7717 ± 0.0003	0.7717 ± 0.0003
	Modelo Propuesto	**0.7949 ± 0.0040**	**0.7958 ± 0.0034**	**0.7943 ± 0.0058**
ITM-Rec	NormalPredictor	0.6363 ± 0.0320	0.6343 ± 0.0303	0.6434 ± 0.0202
	BaselineOnly	0.7060 ± 0.0250	0.7061 ± 0.0250	0.7061 ± 0.0250
	KNNBasic	0.7099 ± 0.0211	0.7100 ± 0.0210	0.7100 ± 0.0210
	KNNWithMeans	0.7099 ± 0.0205	0.7101 ± 0.0204	0.7101 ± 0.0204
	KNNWithZScore	0.7116 ± 0.0197	0.7118 ± 0.0196	0.7118 ± 0.0196
	KNNBaseline	0.7119 ± 0.0199	0.7121 ± 0.0198	0.7121 ± 0.0198
	SlopeOne	0.7035 ± 0.0242	0.7036 ± 0.0243	0.7036 ± 0.0243
	SVD	0.7094 ± 0.0015	0.7095 ± 0.0015	0.7095 ± 0.0015
	SVDpp	0.7041 ± 0.0028	0.7042 ± 0.0030	0.7042 ± 0.0030
	NMF	0.6999 ± 0.0030	0.6998 ± 0.0032	0.6998 ± 0.0032
	CoClustering	0.6948 ± 0.0037	0.6948 ± 0.0037	0.6948 ± 0.0037
	Modelo Propuesto	**0.7477 ± 0.0149**	**0.7481 ± 0.0126**	**0.7456 ± 0.0111**

Tabla 17: Comparación de *MRR* para el modelo propuesto para $K = \{5, 10, 15\}$.

Dataset	**Modelo**	$K = 5$	$K = 10$	$K = 15$
MARS	NormalPredictor	0.8000 ± 0.0098	0.8004 ± 0.0099	0.8005 ± 0.0099
	BaselineOnly	0.8001 ± 0.0099	0.8005 ± 0.0099	0.8005 ± 0.0099
	KNNBasic	0.8002 ± 0.0099	0.8005 ± 0.0099	0.8005 ± 0.0099
	KNNWithMeans	0.7998 ± 0.0099	0.8005 ± 0.0099	0.8005 ± 0.0099
	KNNWithZScore	0.7999 ± 0.0099	0.8005 ± 0.0099	0.8005 ± 0.0099
	KNNBaseline	0.8000 ± 0.0098	0.8005 ± 0.0099	0.8005 ± 0.0099
	SlopeOne	0.8000 ± 0.0098	0.8005 ± 0.0099	0.8005 ± 0.0099
	SVD	0.8001 ± 0.0001	0.8005 ± 0.0001	0.8005 ± 0.0000
	SVDpp	0.8001 ± 0.0002	0.8004 ± 0.0001	0.8005 ± 0.0000
	NMF	0.8000 ± 0.0002	0.8005 ± 0.0001	0.8005 ± 0.0000
	CoClustering	0.8000 ± 0.0002	0.8004 ± 0.0001	0.8005 ± 0.0000
	Modelo Propuesto	**0.7985 ± 0.0044**	**0.7985 ± 0.0044**	**0.7985 ± 0.0044**
ITM-Rec	NormalPredictor	0.7712 ± 0.0247	0.7722 ± 0.0234	0.7722 ± 0.0234
	BaselineOnly	0.7717 ± 0.0232	0.7722 ± 0.0234	0.7722 ± 0.0234
	KNNBasic	0.7717 ± 0.0240	0.7722 ± 0.0234	0.7722 ± 0.0234
	KNNWithMeans	0.7712 ± 0.0238	0.7722 ± 0.0234	0.7722 ± 0.0234
	KNNWithZScore	0.7712 ± 0.0238	0.7722 ± 0.0234	0.7722 ± 0.0234
	KNNBaseline	0.7712 ± 0.0238	0.7722 ± 0.0234	0.7722 ± 0.0234
	SlopeOne	0.7717 ± 0.0232	0.7722 ± 0.0234	0.7722 ± 0.0234
	SVD	0.7715 ± 0.0006	0.7722 ± 0.0000	0.7722 ± 0.0000
	SVDpp	0.7717 ± 0.0005	0.7722 ± 0.0000	0.7722 ± 0.0000
	NMF	0.7717 ± 0.0000	0.7722 ± 0.0000	0.7722 ± 0.0000
	CoClustering	0.7720 ± 0.0003	0.7722 ± 0.0000	0.7722 ± 0.0000
	Modelo Propuesto	**0.7720 ± 0.0109**	**0.7720 ± 0.0109**	**0.7720 ± 0.0109**

Tabla 18: Comparación de *HR* para el modelo propuesto para $K = \{5, 10, 15\}$.

Dataset	**Modelo**	$K = 5$	$K = 10$	$K = 15$
MARS	NormalPredictor	0.7308 ± 0.0078	0.7308 ± 0.0078	0.7308 ± 0.0079
	BaselineOnly	0.7353 ± 0.0079	0.7325 ± 0.0079	0.7317 ± 0.0079
	KNNBasic	0.7354 ± 0.0080	0.7325 ± 0.0079	0.7317 ± 0.0079
	KNNWithMeans	0.7345 ± 0.0080	0.7321 ± 0.0080	0.7314 ± 0.0079
	KNNWithZScore	0.7343 ± 0.0080	0.7322 ± 0.0080	0.7315 ± 0.0079
	KNNBaseline	0.7355 ± 0.0079	0.7327 ± 0.0079	0.7317 ± 0.0079
	SlopeOne	0.7343 ± 0.0078	0.7322 ± 0.0079	0.7315 ± 0.0079
	SVD	0.7345 ± 0.0003	0.7322 ± 0.0001	0.7315 ± 0.0001
	SVDpp	0.7340 ± 0.0002	0.7320 ± 0.0001	0.7315 ± 0.0000
	NMF	0.7331 ± 0.0003	0.7318 ± 0.0001	0.7314 ± 0.0000
	CoClustering	0.7343 ± 0.0001	0.7322 ± 0.0001	0.7315 ± 0.0001
	Modelo Propuesto	**0.7406 ± 0.0030**	**0.7324 ± 0.0034**	**0.7301 ± 0.0034**
ITM-Rec	NormalPredictor	0.5296 ± 0.0241	0.5299 ± 0.0228	0.5300 ± 0.0228
	BaselineOnly	0.5360 ± 0.0222	0.5301 ± 0.0229	0.5301 ± 0.0229
	KNNBasic	0.5370 ± 0.0225	0.5303 ± 0.0231	0.5301 ± 0.0229
	KNNWithMeans	0.5367 ± 0.0227	0.5303 ± 0.0231	0.5301 ± 0.0229
	KNNWithZScore	0.5369 ± 0.0227	0.5303 ± 0.0231	0.5301 ± 0.0229
	KNNBaseline	0.5368 ± 0.0223	0.5303 ± 0.0231	0.5301 ± 0.0229
	SlopeOne	0.5356 ± 0.0224	0.5302 ± 0.0230	0.5301 ± 0.0229
	SVD	0.5364 ± 0.0006	0.5302 ± 0.0001	0.5301 ± 0.0000
	SVDpp	0.5362 ± 0.0004	0.5302 ± 0.0001	0.5301 ± 0.0000
	NMF	0.5356 ± 0.0006	0.5302 ± 0.0001	0.5301 ± 0.0000
	CoClustering	0.5356 ± 0.0002	0.5302 ± 0.0002	0.5301 ± 0.0000
	Modelo Propuesto	**0.5551 ± 0.0111**	**0.5453 ± 0.0103**	**0.5448 ± 0.0102**

Tabla 19: Comparación de *Precision* para el modelo propuesto para $K = \{5, 10, 15\}$.

Dataset	**Modelo**	$K = 5$	$K = 10$	$K = 15$
MARS	NormalPredictor	0.7345 ± 0.0086	0.7737 ± 0.0093	0.7859 ± 0.0100
	BaselineOnly	0.7373 ± 0.0084	0.7749 ± 0.0094	0.7865 ± 0.0099
	KNNBasic	0.7376 ± 0.0084	0.7750 ± 0.0094	0.7865 ± 0.0099
	KNNWithMeans	0.7371 ± 0.0084	0.7748 ± 0.0095	0.7863 ± 0.0100
	KNNWithZScore	0.7369 ± 0.0085	0.7748 ± 0.0095	0.7863 ± 0.0100
	KNNBaseline	0.7376 ± 0.0083	0.7750 ± 0.0094	0.7865 ± 0.0099
	SlopeOne	0.7368 ± 0.0084	0.7747 ± 0.0094	0.7863 ± 0.0099
	SVD	0.7369 ± 0.0002	0.7746 ± 0.0001	0.7864 ± 0.0001
	SVDpp	0.7365 ± 0.0001	0.7746 ± 0.0000	0.7864 ± 0.0001
	NMF	0.7359 ± 0.0003	0.7744 ± 0.0001	0.7862 ± 0.0001
	CoClustering	0.7367 ± 0.0002	0.7747 ± 0.0001	0.7864 ± 0.0001
	Modelo Propuesto	**0.7430 ± 0.0037**	**0.7758 ± 0.0044**	**0.7860 ± 0.0045**
ITM-Rec	NormalPredictor	0.7511 ± 0.0274	0.7714 ± 0.0228	0.7721 ± 0.0233
	BaselineOnly	0.7593 ± 0.0247	0.7717 ± 0.0232	0.7721 ± 0.0234
	KNNBasic	0.7609 ± 0.0258	0.7718 ± 0.0232	0.7721 ± 0.0234
	KNNWithMeans	0.7601 ± 0.0260	0.7718 ± 0.0232	0.7721 ± 0.0234
	KNNWithZScore	0.7603 ± 0.0258	0.7718 ± 0.0232	0.7721 ± 0.0234
	KNNBaseline	0.7602 ± 0.0256	0.7718 ± 0.0232	0.7721 ± 0.0234
	SlopeOne	0.7589 ± 0.0253	0.7717 ± 0.0232	0.7721 ± 0.0234
	SVD	0.7600 ± 0.0015	0.7717 ± 0.0000	0.7721 ± 0.0000
	SVDpp	0.7597 ± 0.0003	0.7717 ± 0.0000	0.7721 ± 0.0000
	NMF	0.7591 ± 0.0007	0.7717 ± 0.0000	0.7722 ± 0.0000
	CoClustering	0.7594 ± 0.0005	0.7716 ± 0.0001	0.7721 ± 0.0000
	Modelo Propuesto	**0.7622 ± 0.0112**	**0.7719 ± 0.0109**	**0.7720 ± 0.0109**

Tabla 20: Comparación de *Recall* para el modelo propuesto para $K = \{5, 10, 15\}$.

Dataset	**Modelo**	$K = 5$	$K = 10$	$K = 15$
MARS	NormalPredictor	0.7326 ± 0.0081	0.7516 ± 0.0084	0.7573 ± 0.0088
	BaselineOnly	0.7363 ± 0.0080	0.7531 ± 0.0085	0.7581 ± 0.0087
	KNNBasic	0.7365 ± 0.0081	0.7531 ± 0.0085	0.7581 ± 0.0087
	KNNWithMeans	0.7358 ± 0.0081	0.7528 ± 0.0086	0.7579 ± 0.0088
	KNNWithZScore	0.7355 ± 0.0081	0.7529 ± 0.0086	0.7579 ± 0.0088
	KNNBaseline	0.7366 ± 0.0080	0.7532 ± 0.0085	0.7581 ± 0.0087
	SlopeOne	0.7355 ± 0.0080	0.7529 ± 0.0085	0.7579 ± 0.0087
	SVD	0.7357 ± 0.0002	0.7528 ± 0.0001	0.7580 ± 0.0001
	SVDpp	0.7352 ± 0.0001	0.7527 ± 0.0001	0.7579 ± 0.0000
	NMF	0.7345 ± 0.0002	0.7525 ± 0.0001	0.7578 ± 0.0000
	CoClustering	0.7355 ± 0.0002	0.7528 ± 0.0001	0.7580 ± 0.0001
	Modelo Propuesto	**0.7418 ± 0.0030**	**0.7535 ± 0.0036**	**0.7570 ± 0.0037**
ITM-Rec	NormalPredictor	0.6212 ± 0.0255	0.6282 ± 0.0232	0.6285 ± 0.0235
	BaselineOnly	0.6284 ± 0.0231	0.6285 ± 0.0234	0.6286 ± 0.0235
	KNNBasic	0.6296 ± 0.0238	0.6286 ± 0.0235	0.6286 ± 0.0235
	KNNWithMeans	0.6291 ± 0.0240	0.6286 ± 0.0235	0.6286 ± 0.0235
	KNNWithZScore	0.6293 ± 0.0239	0.6286 ± 0.0235	0.6286 ± 0.0235
	KNNBaseline	0.6292 ± 0.0236	0.6286 ± 0.0235	0.6286 ± 0.0235
	SlopeOne	0.6280 ± 0.0236	0.6285 ± 0.0234	0.6286 ± 0.0235
	SVD	0.6289 ± 0.0009	0.6285 ± 0.0000	0.6286 ± 0.0000
	SVDpp	0.6286 ± 0.0002	0.6285 ± 0.0000	0.6286 ± 0.0000
	NMF	0.6280 ± 0.0006	0.6285 ± 0.0001	0.6286 ± 0.0000
	CoClustering	0.6282 ± 0.0002	0.6285 ± 0.0001	0.6286 ± 0.0000
	Modelo Propuesto	**0.6423 ± 0.0110**	**0.6391 ± 0.0106**	**0.6388 ± 0.0105**

Tabla 21: Comparación de $F_1@K$ para el modelo propuesto para $K = \{5, 10, 15\}$.

B. Manual de Usuario

Este manual presenta una demostración desarrollada para este Trabajo de Fin de Grado, el sistema consiste en un programa que se ejecuta a través de línea de comandos que ofrece al usuario diferentes modos de ejecución para entrenar, evaluar y ejecutar un pequeño ejemplo de inferencia sobre el modelo propuesto, adicionalmente, ofrece la capacidad de ejecutar las evaluaciones usadas para los algoritmos implementados por la librería *Surprise* [27].

B.1. Instalación

Antes de comenzar la instalación, asegúrate de disponer de un Sistema de Control de Versiones (SCV). El más extendido hoy en día es *Git*, que puedes obtener desde su página oficial (`https://git-scm.com/`). Asimismo, debe tener instalado el gestor de paquetes *uv* (`https://docs.astral.sh/uv/`), responsable de administrar todas las dependencias del proyecto. Cabe destacar que esta guía de instalación es válida tanto para Windows, Mac y Linux.

El sistema se encuentra alojado en un repositorio de *GitHub* (`https://github.com/Pacatro/UcoRecSys`), y los pasos necesarios para su instalación son los siguientes:

1. Clonar el repositorio usando *Git* y acceder a él:

```
git clone https://github.com/Pacatro/UcoRecSys.git
cd UcoRecSys
```

2. Ejecutar el proyecto, este comando instalará automáticamente el entorno virtual y todas las dependencias del proyecto, una vez finalizadas las instalaciones se muestran todos los argumentos disponibles:

```
uv run src/main.py -h
```

B.2. Cómo usarlo

Como se ha comentado anteriormente, el programa ofrece varios modos de ejecución que deben ser seleccionados a través de la línea de comandos, cada modo representa una ejecución diferente para el modelo (entrenar, evaluar, inferencia, etc.).

B.2.1. Modos de ejecución

A continuación se explicarán todos los modos de ejecución disponibles:

- **Entrenamiento**: Esta funcionalidad permite el entrenamiento del modelo utilizando el conjunto de datos especificado y la configuración de parámetros establecida. Durante el proceso de entrenamiento, el sistema proporciona información detallada sobre el progreso a través de la interfaz de terminal, incluyendo métricas de pérdida y de validación. Una vez completado el entrenamiento, el sistema ejecuta automáticamente una evaluación sobre el conjunto de prueba, presentando las métricas de rendimiento obtenidas y almacenando el modelo entrenado en formato *PT*, estándar para modelos desarrollados con el *framework* PyTorch.

- **Inferencia**: Utiliza un modelo previamente entrenado para generar recomendaciones sobre datos no vistos durante el proceso de entrenamiento. El sistema requiere la especificación del conjunto de datos original utilizado para el entrenamiento del modelo, asegurando la coherencia en el procesamiento de características. Como resultado, el sistema genera *rankings* de recomendaciones personalizados basados en las predicciones calculadas para cada usuario del conjunto de prueba.

- **Evaluación**: Implementa un protocolo de validación cruzada para evaluar el rendimiento del modelo sobre un conjunto de datos determinado. Al finalizar el proceso de evaluación, todas las métricas calculadas se almacenan automáticamente en archivos con formato *CSV*, facilitando el análisis posterior y la comparación de resultados entre diferentes configuraciones del modelo.

- **Evaluación de *Surprise***: Esta funcionalidad extiende el proceso de evaluación a algoritmos implementados en la librería Surprise, aplicando los mismos protocolos de validación utilizados para el modelo propuesto. Esta capacidad permite establecer comparaciones directas y objetivas entre el modelo desarrollado y algoritmos de referencia reconocidos en la literatura.

- **Análisis estadísticos**: Esta funcionalidad ejecuta pruebas estadísticas formales sobre los resultados obtenidos en las evaluaciones previas, almacenando los análisis en archivos *CSV* estructurados. Esta modalidad requiere la disponibilidad previa de resultados de evaluación y proporciona validación estadística de las diferencias de rendimiento observadas entre diferentes modelos o configuraciones.

B.2.2. Argumentos de línea de comandos

En esta sección se detallarán todos los argumentos disponibles en el programa que permiten una configuración mucho más personalizada, estos comandos se dividen cuatro grupos dependiendo de su significado dentro del sistema:

- **Opciones principales**: Argumentos para indicar el modo de ejecución:
  - `-t`,`--train`: Activa el modo entrenamiento.
  - `-e`,`--eval`: Activa el modo evaluación del modelo.
  - `-s`,`--surprise`: Activa el modo evaluación de los modelos de *Surprise*.
  - `-i`,`--inference MODELO.pt`: Activa el modo inferencia usando el modelo proporcionado.
  - `-st`,`--stats_test`: Ejecuta la prueba de estadísticas (por defecto: False).

- **Opciones comunes**: Argumentos para configurar el conjunto de datos y la validación cruzada:
  - `-ds {mars, itm}`, `--dataset {mars, itm}`: Conjunto de datos a usar (por defecto: mars).

- `-cv {kfold, loo}`, `--cvtype {kfold, loo}`: Tipo de validación cruzada (por defecto: kfold).
- `--top_k TOP_K`: Valor Top-K para métricas de ranking (por defecto: 10).

- **Opciones de entrenamiento**: Argumentos para configurar el entrenamiento del modelo:
  - `--epochs EPOCHS`: Épocas de entrenamiento (por defecto: 50).
  - `--batch-size BATCH_SIZE`: Tamaño del lote de entrenamiento (por defecto: 128).
  - `--output-model MODEL_OUT`: Ruta para guardar el modelo entrenado (por defecto: model.pt).
  - `-lr LR`: Tasa de aprendizaje (por defecto: 0.001).
- **Opciones de evaluación**: Argumentos para configurar la evaluación del modelo:
  - `-k K_SPLITS`, `--k_splits K_SPLITS`: Número de divisiones de validación cruzada (por defecto: 5).
  - `--seeds SEEDS`: Semillas aleatorias para los algoritmos no deterministas de *Surprise* (por defecto: [0, 1, 42]).
- **Opciones varias**: Argumentos adicionales:
  - `-v`, `--verbose`: Activa la salida detallada haciendo que se muestre información adicional.

B.2.3. Ejemplo de uso:

A continuación se mostrará un pequeño ejemplo práctico que muestra el proceso de entrenamiento del modelo y las salidas correspondientes.

Para este ejemplo entrenaremos el modelo sobre el conjunto de datos *MARS*, utilizando 10 épocas de entrenamiento y una lista del *top*-15 de recomendaciones

para evaluar el modelo y lo guardaremos en el archivo `modelo.pt`. El comando utilizado para este propósito sería el siguiente:

```
uv run src/main.py -t -ds mars --epochs 10 --top_k 10 --output-
    model modelo.pt
```

La Figura 18 muestra el resultado final una vez finalizado el entrenamiento, donde se puede ver en detalle todo el proceso de entrenamiento, así como los valores de las métricas calculadas sobre el conjunto de pruebas. Al final de la ejecución, el programa muestra un mensaje indicando la ruta en la que se ha almacenado el modelo ya entrenado, en nuestro caso, `modelo.pt`

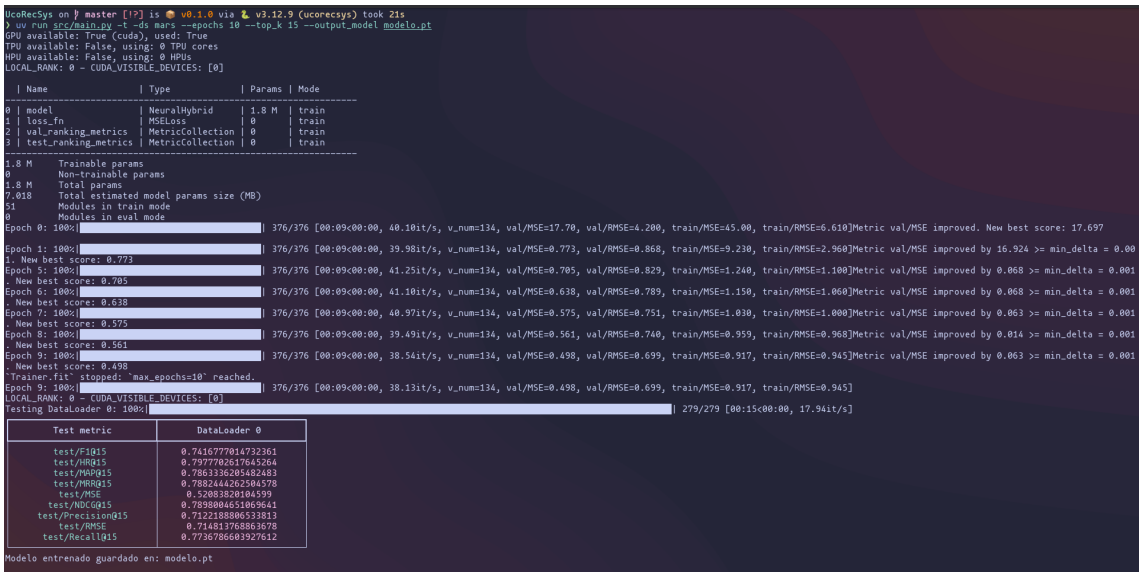


Figura 18: Ejemplo de salida del programa

C. Manual de código

Esta sección presenta una descripción detallada de la arquitectura y estructura del programa desarrollado para la demostración del sistema de recomendación implementado en este trabajo de fin de grado. El conjunto completo de código fuente, *datasets* utilizados y resultados experimentales se encuentra disponible públicamente en el repositorio del proyecto alojado en la plataforma *GitHub* (`https://github.com/Pacatro/UcoRecSys`).

C.1. Estructura del repositorio

El repositorio se organiza mediante una arquitectura modular que facilita la navegación y comprensión del proyecto. La estructura principal comprende tres directorios fundamentales que separan de manera lógica los diferentes componentes del sistema.

El directorio `data` constituye el repositorio central de información del proyecto, conteniendo todos los conjuntos de datos empleados durante el desarrollo y evaluación del sistema. Estos datasets se almacenan en formato *CSV* para garantizar la compatibilidad y facilitar el procesamiento mediante las herramientas de análisis implementadas.

El directorio `results` alberga la totalidad de los resultados experimentales generados durante el proceso de evaluación del sistema de recomendación. Esta sección se subdivide estratégicamente para organizar los diferentes tipos de resultados obtenidos. El subdirectorio `stats` contiene específicamente todos los resultados derivados del análisis estadístico realizado sobre el rendimiento del sistema, incluyendo métricas de evaluación, comparativas de algoritmos y análisis de significancia estadística.

El directorio `src` representa el núcleo técnico del proyecto, conteniendo la totalidad del código fuente desarrollado. Esta sección incluye la implementación de

los algoritmos de recomendación, los módulos de preprocesamiento de datos, las funciones de evaluación y todos los componentes auxiliares necesarios para el funcionamiento integral del sistema.

C.1.1. Estructura del código fuente (`src`)

La organización del código fuente sigue una arquitectura modular que separa las responsabilidades del sistema en componentes especializados, facilitando el mantenimiento y la escalabilidad del proyecto.

El módulo `args_parser.py` constituye el punto de entrada para la gestión de argumentos de línea de comandos, proporcionando una interfaz estandarizada para la configuración de parámetros del sistema durante la ejecución. Este componente permite la parametrización flexible del comportamiento del sistema sin necesidad de modificar el código fuente.

El archivo `config.py` centraliza todas las configuraciones globales del proyecto, incluyendo rutas de archivos, parámetros de algoritmos y constantes del sistema, lo que garantiza la consistencia de configuración a través de todos los módulos y simplifica la gestión de variables de entorno.

Los módulos `datasets.py` y `datamodule.py` están destinados al manejo de los conjuntos de datos. El primero define los métodos necesarios para cargar los ficheros *CSV* correspondientes a los conjuntos de datos, mientras que el segundo implementa las funcionalidades de transformación y preparación de dichos *datasets*.

El módulo `model.py` encapsula la implementación del modelo utilizado para elaborar el sistema de recomendación, definiendo la arquitectura de este, así como los hiperparámetros que lo conforman.

El componente `engine.py` representa el sistema de recomendación en su totalidad, en él se definen los procesos de entrenamiento y validación del modelo usando

las funcionalidades del *framework PyTorch Lightning*, en él se definen también el conjunto de métricas que serán utilizadas para el proceso de evaluación. Este módulo utiliza el modelo definido previamente en `model.py` para generar las predicciones finales

Los módulos `evaluation.py` y `stats.py` implementan los métodos necesarios para la evaluación del sistema. El primero contiene los algoritmos de validaciones cruzadas que serán aplicados a los modelos, mientras que el segundo se especializa en el análisis estadístico de los resultados obtenidos, incluyendo pruebas de significancia y generación de reportes estadísticos.

El componente `surprise_eval.py` proporciona funcionalidades especializadas para la evaluación del sistema implementado en la librería *Surprise*, permitiendo la comparación con implementaciones de referencia y la validación cruzada de los resultados obtenidos.

Finalmente, el archivo `main.py` funciona como el punto de entrada principal de la aplicación, integrando todos los componentes del sistema y proporcionando la interfaz de ejecución unificada.

C.2. Archivos adicionales

El repositorio cuenta con una serie de archivos y *scripts* adicionales que complementan al programa y que han sido muy útiles para realizar las evaluaciones y análisis de los resultados.

- `run_evals.sh`: Proporciona una implementación básica para ejecutar todas las pruebas que han sido usadas para evaluar los modelos.
- `run_stats.sh`: Al igual que el *script* anterior, este permite ejecutar todas las pruebas estadísticas utilizadas para el análisis de los resultados.
- `results_study.ipynb`: Es donde se encuentran los métodos utilizados para

generar las gráficas de visualización de resultados, aunque no pertenezca al programa principal, se ha decidido incluir para tener disponibles estos métodos.