GRADO EN INGENIERÍA MATEMÁTICA E INTELIGENCIA ARTIFICIAL

TRABAJO FIN DE GRADO

DESARROLLO DE UNA LIBRERÍA EN PYTHON DE SISTEMAS DE RECOMENDACIÓN

Autor: Catalina Royo-Villanova Seguí

Director: Pablo Sánchez Pérez

Madrid

Junio de 2025

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Desarrollo de una librería en Python de sistemas de recomendación

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2024/25 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Firmado por ROYO-VILLANOVA SEGUI CATALINA - ***1334** el día 14/06/2025 con un certificado emitido por AC FNMT Usuarios

Fdo.: Catalina Royo-Villanova Seguí Fecha: 14 / 06 / 2025

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Firmado por SANCHEZ PEREZ PABLO - ***6329** el día 15/06/2025 con un certificado emitido por AC FNMT Usuarios

Fdo.: Pablo Sánchez Pérez Fecha: / / .

Quiero agradecer a mis amigos y a mi familia, quienes han estado a mi lado y me han apoyado durante todo el proceso de elaboración de este Trabajo de Fin de Grado. Especialmente a mis padres, mis hermanos y mis compañeras de piso, que me han visto avanzar y superar este reto día a día. Su ánimo me ha impulsado a seguir adelante cuando he dudado de mí misma, tanto en este proyecto como a lo largo de toda la carrera.

También me gustaría agradecer a ICAI por la excelente formación recibida durante estos cuatro años; al jefe de estudios, por organizar este grado que tanto me alegro de haber estudiado; y a mis profesores que han puesto todo de su parte para enseñarnos no solo el material de las asignaturas, sino también por empujarnos a perseverar y a mejorar tanto como alumnos como como personas. A mis compañeros de clase que han compartido esta experiencia conmigo y han contribuido a que me sintiera en clase como en casa.

Finalmente, quiero agradecer a mi tutor, que me ha guiado durante este trabajo, resolviendo mis dudas y orientándome en la dirección correcta.

DESARROLLO DE UNA LIBRERÍA EN PYTHON DE SISTEMAS DE RECOMENDACIÓN

Autor: Royo-Villanova Seguí, Catalina.
Director: Sánchez Pérez, Pablo.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

Resumen

Este trabajo presenta el desarrollo de una librería en Python diseñada para implementar, comparar y evaluar sistemas de recomendación de manera modular y extensible. Se han programado distintos modelos, desde enfoques clásicos como popularidad, vecinos próximos (KNN) o factorización de matrices, hasta métodos basados en redes neuronales como Multi-Layer Perceptron (MLP) y Graph Neural Networks (GNN). La herramienta ha sido validada sobre dos conjuntos de datos reales y contrastados (MovieLens 100K [1] y Foursquare New York [2], [3]) con métricas que evalúan no solo la precisión de las recomendaciones, sino también su novedad y diversidad.

Palabras clave: Sistemas de Recomendación, Python, Librería, Evaluación, Filtrado Colaborativo

1. Introducción

En la actualidad, los usuarios se enfrentan a un volumen de información abrumador, lo que dificulta la toma de decisiones y reduce la eficiencia en el consumo de contenidos. Los sistemas de recomendación surgen como una solución clave en plataformas como Netflix, Amazon o Spotify, donde permiten ofrecer contenidos personalizados en función de las preferencias del usuario.

Este Trabajo de Fin de Grado se centra en el desarrollo de una librería en Python para la implementación, entrenamiento y evaluación de sistemas de recomendación, permitiendo la comparación rigurosa de distintos algoritmos. A diferencia de librerías existentes, que pueden ser complejas, poco extensibles o dependientes de formatos rígidos de datos, la solución desarrollada busca la máxima flexibilidad, facilitando tanto su uso como su ampliación.

La librería permite trabajar tanto con valoraciones explícitas como con interacciones implícitas, y está orientada a la experimentación sistemática sobre distintos datasets. Asimismo, incorpora métricas tradicionales (precisión, recall, NDCG) y otras más recientes centradas en la diversidad y novedad de las recomendaciones, como el Expected Popularity Complement (EPC), el índice de Gini y la diversidad agregada.

2. Definición del proyecto

El proyecto parte del objetivo de ofrecer una herramienta versátil que permita evaluar modelos de recomendación en condiciones comparables. Se plantean los siguientes objetivos:

- Implementar un módulo de datos que gestione la carga, procesamiento y partición de los conjuntos de interacciones usuario-ítem.

- Desarrollar un módulo de recomendadores con diferentes algoritmos, siguiendo una interfaz común.
- Crear un módulo de evaluación capaz de medir el rendimiento de los modelos desde múltiples perspectivas.
- Incorporar scripts de entrenamiento, recomendación y evaluación que permitan automatizar experimentos de forma reproducible.
- Analizar el comportamiento de los modelos mediante gráficas y tablas, evaluando el efecto de los hiperparámetros en su rendimiento.

La arquitectura modular de la librería facilita su extensibilidad. Nuevos modelos o métricas pueden añadirse sin modificar el resto del sistema, lo que la hace apta para usos educativos, experimentales o productivos.

3. Descripción del modelo/sistema/herramienta

La librería está organizada en varios módulos interdependientes:

- *datamodule/*: Encargado de cargar y estructurar los datos. Convierte las interacciones en matrices dispersas y permite tanto trabajar con datasets ya particionados como aplicar distintas estrategias de división en entrenamiento y test.
- *recommenders/*: Contiene las implementaciones de los distintos algoritmos, agrupados en:
    - Modelos simples: Popularidad y recomendación aleatoria.
    - KNN: Versiones user-based y item-based, con medidas de similitud como coseno y Pearson.
    - Matrix Factorization (MF): Entrenado con PyTorch minimizando el error cuadrático medio.
    - BPRMF: Basado en optimización de ranking sobre tripletas (usuario, ítem positivo, ítem negativo).
    - MLP: Perceptrón multicapa que concatena embeddings y los procesa mediante una red neuronal densa.
    - GNN: Graph Neural Network sobre un grafo bipartito usuario–ítem, implementado como una red convolucional.
- *evaluation/*: Implementa las métricas mencionadas anteriormente. Puede evaluar recomendaciones almacenadas en ficheros CSV y está completamente desacoplado del modelo que las ha generado.
- *pipelines/*: Scripts de automatización para entrenar múltiples configuraciones de modelos (grid search), evaluar sus resultados y generar visualizaciones.

Todos los modelos siguen una interfaz común basada en herencia, lo que permite llamar a cualquier recomendador con una estructura estándar y evaluar su salida de forma uniforme. Las estrategias de filtrado (por ejemplo, excluir ítems ya vistos) también están separadas en clases especializadas.

4. Resultados

Se ha validado la librería sobre dos datasets con características distintas: MovieLens 100K, que contiene valoraciones explícitas y permite evaluar la precisión directa de los modelos, y Foursquare New York, basado en datos implícitos de check-ins, donde los usuarios pueden repetir interacciones, lo que requiere ajustar las estrategias de evaluación. Esta dualidad ha permitido comprobar el comportamiento de los modelos en distintos contextos.

Los resultados muestran que BPRMF fue el modelo más eficaz en ambos datasets en precisión, recall y NDCG, gracias a su entrenamiento orientado a ranking. KNN user-based también ofreció buen rendimiento, especialmente en MovieLens, destacando por su simplicidad y eficiencia. En contraste, los modelos item-based lograron menor precisión, pero sobresalieron en novedad y diversidad, siendo adecuados para tareas de descubrimiento. Matrix Factorization (MF) fue muy sensible a los hiperparámetros, alcanzando buenos valores de EPC y diversidad en MovieLens. MLP no alcanzó buenos resultados en precisión, y, por último, GNN, aunque llegó a ser bastante competitivo, no superó a otros modelos, lo que hace que no se pueda acabar de justificar esta complejidad agregada sin una mejora de los resultados.

5. Conclusiones

El desarrollo de esta librería ha permitido comparar distintos sistemas de recomendación en dos tipos de datos muy diferentes. Gracias a los experimentos realizados, se pueden extraer varias conclusiones importantes:

- No hay un único modelo mejor en todo. BPRMF es el más preciso en general, pero otros modelos, como el KNN basado en ítems, ofrecen más diversidad, lo que puede ser útil según el objetivo del sistema.
- El tipo de datos influye mucho en el resultado. En datasets como Foursquare, donde hay muchos más ítems que usuarios, los modelos que comparan usuarios funcionan mejor porque hay más información compartida entre ellos.
- Los modelos más complejos no siempre dan mejores resultados. Aunque GNN y MLP usan técnicas más avanzadas, no han superado a modelos más simples como BPRMF, sobre todo cuando no se cuenta con información adicional como características de los ítems o contexto.

En conjunto, la librería desarrollada es una herramienta flexible y completa que permite probar y comparar diferentes modelos de forma sencilla. Su diseño modular la hace útil tanto para experimentar como para seguir ampliándola en el futuro, añadiendo nuevos algoritmos o evaluaciones más avanzadas.

6. Referencias

Disponibles al final de la memoria.

DEVELOPMENT OF A PYTHON LIBRARY FOR RECOMMENDER SYSTEMS

Author: Royo-Villanova Seguí, Catalina.
Supervisor: Sánchez Pérez, Pablo.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

Abstract

This work presents the development of a Python library designed to implement, compare, and evaluate recommender systems in a modular and extensible manner. Various models have been implemented, ranging from classical approaches such as popularity, nearest neighbors (KNN), or matrix factorization, to neural network-based methods such as Multi-Layer Perceptron (MLP) and Graph Neural Networks (GNN). The tool has been validated on two well-known real-world datasets (MovieLens 100K [1] and Foursquare New York [2], [3]) using metrics that assess not only the accuracy of recommendations but also their novelty and diversity.

Keywords: Recommender Systems, Python, Library, Evaluation, Collaborative Filtering

1. Introduction

Nowadays, users face an overwhelming volume of information, making decision-making difficult and reducing the efficiency of content consumption. Recommender systems have emerged as a key solution on platforms like Netflix, Amazon, or Spotify, where they deliver personalized content based on user preferences.

This TFG focuses on the development of a Python library for the implementation, training, and evaluation of recommender systems, allowing rigorous comparison of different algorithms. Unlike existing libraries, which may be complex, poorly extensible, or dependent on rigid data formats, the solution developed aims for maximum flexibility, facilitating both usage and extension.

The library supports both explicit ratings and implicit interactions and is oriented towards systematic experimentation on various datasets. It also incorporates traditional metrics (precision, recall, NDCG) as well as more recent ones focused on recommendation diversity and novelty, such as Expected Popularity Complement (EPC), Gini index, and aggregate diversity.

2. Project Definition

The project is based on the objective of providing a versatile tool for evaluating recommender systems under comparable conditions. The following objectives are proposed:

- Implement a data module to manage loading, processing, and partitioning of user–item interaction datasets.
- Develop a recommender module with different algorithms, following a common interface.
- Create an evaluation module capable of measuring model performance from multiple perspectives.

- Include training, recommendation, and evaluation scripts to automate reproducible experiments.
- Analyze model behavior through graphs and tables, evaluating the effect of hyperparameters on performance.

The library's modular architecture facilitates its extensibility. New models or metrics can be added without modifying the rest of the system, making it suitable for educational, experimental, or production use.

3. Model/System/Tool Description

The library is organized into several interdependent modules:

- *datamodule/*: Responsible for loading and structuring the data. It converts interactions into sparse matrices and supports both working with pre-split datasets and applying different train/test splitting strategies.
- *recommenders/*: Contains implementations of various algorithms, grouped into:
    - Simple models: Popularity and random recommendation.
    - KNN: User-based and item-based versions, with similarity measures such as cosine and Pearson.
    - Matrix Factorization (MF): Trained with PyTorch, minimizing mean squared error.
    - BPRMF: Based on ranking optimization over triplets (user, positive item, negative item).
    - MLP: Multi-layer perceptron that concatenates embeddings and processes them through a dense neural network.
    - GNN: Graph Neural Network over a bipartite user–item graph, implemented as a convolutional network.
- *evaluation/*: Implements the aforementioned metrics. It can evaluate recommendations stored in CSV files and is completely independent from the model that generated them.
- *pipelines/*: Automation scripts to train multiple model configurations (grid search), evaluate their results, and generate visualizations.

All models follow a common interface based on inheritance, allowing any recommender to be called with a standard structure and its output to be evaluated uniformly. Filtering strategies (e.g., excluding already seen items) are also separated into specialized classes.

4. Results

The library has been validated on two datasets with different characteristics: MovieLens 100K, which contains explicit ratings and allows direct evaluation of model accuracy, and Foursquare New York, based on implicit check-in data where users can repeat interactions, requiring adjusted evaluation strategies. This duality has permitted observing model behavior in different contexts.

Results show that BPRMF was the most effective model on both datasets in terms of precision, recall, and NDCG, due to its ranking-oriented training. User-based KNN also performed well, especially on MovieLens, standing out for its simplicity and efficiency. In

contrast, item-based models achieved lower precision but excelled in novelty and diversity, making them suitable for discovery tasks. Matrix Factorization (MF) was highly sensitive to hyperparameters, achieving good EPC and diversity scores on MovieLens. MLP did not achieve good accuracy results, and finally, GNN, while relatively competitive, did not outperform other models, making it difficult to justify the added complexity without improved results.

5. Conclusions

The development of this library has enabled the comparison of different recommender systems on two very distinct types of data. From the experiments conducted, several important conclusions can be drawn:

- There is no single best model for all situations. BPRMF is the most accurate overall, but other models, like item-based KNN, offer greater diversity, which may be useful depending on the system's goals.
- The type of data strongly influences results. In datasets like Foursquare, where there are many more items than users, user-based models perform better due to more shared information.
- More complex models do not always yield better results. Although GNN and MLP use more advanced techniques, they did not outperform simpler models like BPRMF, especially when no additional information such as item features or context is available.

Overall, the developed library is a flexible and comprehensive tool that allows for easy testing and comparison of different models. Its modular design makes it suitable for future extensions, including new algorithms or more advanced evaluations.

6. References

Available at the end of the thesis.

Índice de la memoria

1. Introducción 1

2. Estado del Arte 1
2.1. Definición del problema y notación 1
2.2. Tipos de datos 2
2.3. Recomendadores 2
2.3.1. Recomendador de Popularidad 3
2.3.2. Recomendador Aleatorio 3
2.3.3. K-Nearest Neighbors 3
2.3.4. Matrix Factorization 4
2.3.5. Bayesian Personalized Ranking (BPR) 5
2.3.6. Multi Layer Perceptron (MLP) 6
2.3.7. Graph Neural Network (GNN) 6
2.4. Métricas de evaluación 7
2.4.1. Métricas de Relevancia 7
2.4.2. Métricas de Novedad y Diversidad 8
2.5. Métodos de división de datos 9

3. Metodología 10
3.1. Justificación 10
3.2. Herramientas 10
3.3. Diseño 10
3.3.1. Datamodule 12
3.3.2. Recommenders 13
3.3.3. Evaluation 14
3.3.4. Utils 15
3.3.5. Main 15
3.3.6. Pipelines 16
3.3.7. Extensibilidad y configuración 16
3.4. Objetivos 17
3.4.1. Objetivos Principales 17
3.4.2. Objetivos Secundarios 17

4. Experimentos 17
4.1. Conjunto de Datos 17

4.1.1. MovieLens 100K Dataset ... *17*
4.1.2. Foursquare Check-in Dataset ... *18*
4.2. Configuración ... 18
4.3. Análisis de Rendimiento ... 19
4.4. Diseño experimental ... 19
4.4.1. Particiones de entrenamiento y test ... *19*
*4.4.2. Tamaño de la lista de recomendación **n** (cutoff)* ... *19*
4.4.3. Estrategia de selección del conjunto de candidatos ... *19*
4.4.4. Optimización de hiperparámetros ... *20*
5. Resultados ... ***20***
5.1. Introducción ... 20
5.2. Dataset MovieLens 100k ... 21
5.2.1. Comparativa general entre modelos ... *21*
5.2.2. Resultados de KNN ... *21*
5.2.3. Resultados de Matrix Factorization ... *21*
5.2.4. Resultados de BPRMF ... *22*
5.2.5. Resultados de MLP ... *22*
5.2.6. Resultados de GNN ... *22*
5.3. Dataset Foursquare New York ... 23
5.3.1. Comparativa general entre modelos ... *23*
5.3.2. Resultados de KNN ... *23*
5.3.3. Resultados de MF ... *23*
5.3.4. Resultados BPRMF ... *23*
5.3.5. Resultados de MLP ... *24*
5.3.6. Resultados de GNN ... *24*
5.4. Discusión de resultados ... 24
6. Conclusiones y Trabajos Futuros ... ***26***
7. Bibliografía ... ***26***
ANEXO I 29

Índice de figuras

Figura 1: Diagrama mostrando la composición y la relación entre módulos y clases. Se muestra en blanco datamodule.data, en naranja datamodule.splits en amarillo recommenders, en verde evaluation, en azul utils.similarities, en rojo utils.strategies y finalmente en gris utils.datasets 12
Figura 2: Evolución de la precisión en función del valor de *k* para KNN y del número de factores latentes para MF y BPRMF para el dataset MovieLens 100k. En los dos últimos, la línea muestra la media y el área azul representa el intervalo de confianza del 95% alrededor de dicha media. 22
Figura 3: Evolución de la precisión en función del valor de *k* para KNN y del número de factores latentes para MF y BPRMF para el dataset Foursquare New York. En los dos últimos, la línea muestra la media y el área azul representa el intervalo de confianza del 95% alrededor de dicha media. 24
Figura 4: Gráfica mostrando la comparación en las diferentes métricas obtenidas por las mejores configuraciones de cada modelo para el dataset MovieLens 100k 29
Figura 5: Gráfica mostrando la comparación en las diferentes métricas obtenidas por las mejores configuraciones de cada modelo para el dataset Foursquare New York 29

Índice de tablas

Tabla 1: Tabla mostrando los diferentes hiperparámetros probados para los diferentes algoritmos y diferentes datasets. 20
Tabla 2: Resultados obtenidos por la mejor configuración de cada modelo para el dataset de MovieLens 100k 21
Tabla 3: Evaluación de los top-5 modelos de KNN User-based y Item-based para el dataset MovieLens 100k 21
Tabla 4: Evaluación de los top-5 modelos MF en precisión para el dataset MovieLens 100k 21
Tabla 5: Evaluación de los top-5 modelos de BPRMF en precisión para el dataset MovieLens 100k 22
Tabla 6: Evaluación de los top-5 modelos de MLP en precisión para el dataset MovieLens 100k 22
Tabla 7: Evaluación de los top-5 modelos de GNN en precisión para el dataset MovieLens 100k 22
Tabla 8: Resultados obtenidos por la mejor configuración de cada modelo para el dataset Foursquare New York 23
Tabla 9: Evaluación de los top-5 modelos de KNN User-based y Item-based para el dataset Foursquare New York 23
Tabla 10: Evaluación de los top-5 modelos de MF en precisión para el dataset Foursquare New York 23
Tabla 11: Evaluación de los top-5 modelos de BPRMF en precisión para el dataset Foursquare New York 23
Tabla 12: Evaluación de los top-5 modelos de MLP en precisión para el dataset Foursquare New York 24
Tabla 13: Evaluación de los top-5 modelos de GNN en precisión para el dataset Foursquare New York 24

1. INTRODUCCIÓN

Los sistemas de recomendación son herramientas de software diseñadas para ofrecer sugerencias personalizadas a los usuarios [4], como películas, productos, canciones o lugares de interés, basadas en sus preferencias, contexto y necesidades. Para ello, estos sistemas utilizan los datos recopilados sobre los usuarios y los artículos existentes. Estos datos pueden ser demográficos, como la edad y el género, datos geográficos para recomendaciones basadas en una ubicación, y datos de popularidad, que reposan sobre las preferencias generales de una determinada comunidad. El uso de estas herramientas se ha popularizado en los últimos años gracias a ciertas aplicaciones como las que han incluido Netflix, Amazon, TikTok o LinkedIn.

En la era en la que vivimos, los usuarios se enfrentan a la denominada sobrecarga informativa [5], conocida en inglés como "information overload". Este problema radica en el exceso de información que la capacidad de un individuo puede manejar de manera eficiente. Varios síntomas que puede presentar una persona al enfrentarse a este problema son estrés, reducción del rendimiento laboral, confusión, o tardanza en la toma de decisiones. En este contexto, los sistemas de recomendación resultan especialmente útiles, ya que no solo ayudan a gestionar de forma más eficiente la gran cantidad de información disponible, sino que también tienen como objetivo principal identificar y sugerir los artículos más relevantes para cada usuario.

No obstante, los sistemas de recomendación no solo benefician a los usuarios de las plataformas. También hay otros actores involucrados que se ven afectados por las decisiones tomadas por estos sistemas, como los proveedores de los artículos o la propia entidad que proporciona el sistema, convirtiéndose así en entornos multistakeholder. Uno de los dominios más representativos donde destaca esta interacción entre múltiples actores es el de la recomendación de puntos de interés (Points of Interest, POI), que tiene como objetivo sugerir lugares relevantes a los usuarios, como restaurantes, museos o monumentos, en función de su ubicación, preferencias u otros factores contextuales. En este caso, los propietarios de los locales recomendados se benefician de una mayor visibilidad, mientras que, en el comercio electrónico, los proveedores de los artículos recomendados obtienen un aumento potencial en las ventas.

En este Trabajo de Fin de Grado (TFG) se plantea el desarrollo de una librería en Python diseñada para implementar y evaluar diferentes sistemas de recomendación. Este proyecto busca aplicar desde modelos más clásicos como el de popularidad hasta modelos más novedosos, con el objetivo de analizar su rendimiento con distintas métricas.

El desarrollo de esta librería en Python tiene como objetivo facilitar la replicación de resultados y la comparación de modelos de recomendación. Esto es relevante ya que la implementación de los recomendadores puede ser sensible al tipo de datos a los que se enfoca, lo que complica la posibilidad de verificar y reproducir resultados de diferentes estudios. En este TFG, se busca implementar una versión lo más abstraída de los datos posibles, que sea fácilmente extrapolable de un conjunto de datos a otro con características completamente diferentes.

2. ESTADO DEL ARTE

2.1. DEFINICIÓN DEL PROBLEMA Y NOTACIÓN

Un sistema de recomendación es una herramienta diseñada para predecir la relevancia de una serie de ítems para un usuario determinado, con el objetivo de generar una lista ordenada de recomendaciones

personalizadas. Estos sistemas se aplican en una amplia variedad de dominios, como el comercio electrónico, las plataformas de streaming, la educación, las redes sociales o la recomendación de puntos de interés.

Formalmente, se parte de un conjunto de usuarios $U = \{u_1, u_2, \dots, u_m\}$ y un conjunto de ítems $I = \{i_1, i_2, \dots, i_d\}$, donde el objetivo principal es estimar la utilidad o relevancia $\hat{r}_{ui}$ de un ítem $i \in I$ para un usuario $u \in U$. Esta utilidad puede estar representada por datos explícitos, como valoraciones numéricas o datos implícitos, que suelen ser binarios.

Las relaciones entre usuarios e ítems se pueden representar mediante una matriz de utilidad $R \in \mathbb{R}^{mxd}$, en la que cada elemento r_{ui} indica el nivel de preferencia del usuario u por el ítem i. El objetivo de un sistema de recomendación es, para cada usuario u, generar una lista ordenada de n ítems $L_n(u) = \{i_1, i_2, \dots, i_n\}$ ordenados de mayor a menor relevancia. Estos ítems se seleccionan de entre un conjunto de ítems candidatos $C(u) \subset I$ que representan los ítems que el sistema considera potencialmente recomendables para ese usuario. Este conjunto de candidatos puede estar formado por todos los ítems del sistema o por un subconjunto filtrado, por ejemplo, excluyendo los que el usuario ya ha consumido.

Para evaluar el rendimiento de estos modelos, se considera el conjunto de test ítems relevantes reales para cada usuario $T(u) \subset I$, obtenido separando una sección de sus interacciones pasadas. A partir de ahí, se comparan las recomendaciones producidas $L_n(u)$ y $T(u)$ con diferentes métricas, que se detallarán más adelante.

2.2. TIPOS DE DATOS

Como se ha mencionado en la subsección anterior, los dos tipos principales de datos con los que trabajan los sistemas de recomendación son:

- Los datos explícitos, es decir, aquellos en los que el usuario proporciona directamente una valoración sobre un ítem. Estas valoraciones suelen tener la forma de puntuación numérica en una escala establecida, como del 1 al 5 o de 0 al 100. Este tipo de información da una visión clara y objetiva del nivel de satisfacción de un usuario con un ítem.
- Los datos implícitos: aquellos recolectados a través de interacciones observadas, como clics, reproducciones, compras, tiempo de visualización o visitas. En estos casos, no se obtiene una valoración explícita por parte del usuario, sino que se infieren sus preferencias a partir del comportamiento. Aunque estos datos pueden tener valores no binarios (por ejemplo, número de veces que se ha hecho clic en un ítem), en muchos casos se binarizan para simplificar el tratamiento: se representa la interacción como $r_{ui} = 1$ si ha habido una interacción y $r_{ui} = 0$ si no ha habido. Estos datos suelen ser más abundantes que los datos explícitos, pero no siempre indican una preferencia positiva y no se tiene información sobre ítems no consumidos.

2.3. RECOMENDADORES

Los métodos más destacados en sistemas de recomendación son los de filtrado colaborativo, que permiten generar recomendaciones basadas en patrones de comportamiento compartidos entre usuarios; los basados en contenido, que analizan las propiedades de los artículos, o ítems, para establecer relaciones entre ellos y mejorar así las recomendaciones; y finalmente los híbridos, que son una combinación de ambos métodos para obtener mejores resultados [4]. En este TFG nos centraremos en los sistemas de filtrado colaborativo.

2.3.1. RECOMENDADOR DE POPULARIDAD

El recomendador basado en popularidad es uno de los enfoques más simples que se utiliza como línea base. Consiste en recomendar a todos los usuarios los ítems más populares del sistema, es decir, aquellos con mayor número de interacciones totales.

La puntuación de popularidad del ítem $i \in I$ se calcula como:

$$pop(i) = |U_i|$$

Donde U_i representa el conjunto de usuarios que han interactuado con el ítem i. Este modelo no tiene en cuenta las preferencias individuales del usuario, por lo que las recomendaciones no están personalizadas, pero puede ser útil con nuevos usuarios o usuarios que no tengan historial.

Aunque esta es la definición básica, este enfoque se puede adaptar incorporando información temporal. Por ejemplo, en lugar de considerar la popularidad global acumulada, se pueden calcular las interacciones más recientes (como en el último mes o año), permitiendo recomendar ítems que sean populares actualmente y reflejen tendencias recientes. Esto mejora la frescura de las recomendaciones y puede ser especialmente útil en dominios donde los intereses cambian rápidamente. Un caso representativo es el de la recomendación de noticias, donde la actualidad de los artículos es clave para la satisfacción del usuario. En este tipo de sistemas, incorporar métricas de frescura resulta fundamental para priorizar contenido reciente sobre noticias ya obsoletas [6].

2.3.2. RECOMENDADOR ALEATORIO

El recomendador aleatorio selecciona ítems de forma completamente arbitraria, asignando un valor aleatorio a cada uno y recomendando aquellos con mayor puntuación generada al azar. Para cada usuario u, se genera una lista de recomendación $L_n(u)$ seleccionando n ítems aleatorios del conjunto I.

Su utilidad principal no se basa en la relevancia de las recomendaciones, sino en servir como línea base inferior en términos de novedad y diversidad.

2.3.3. K-NEAREST NEIGHBORS

Los métodos de recomendación basados en vecinos se basan en la idea de que usuarios o ítems similares tienden a comportarse de forma parecida [7], [8]. Hay dos enfoques principales dentro de este grupo de recomendadores:

- El enfoque user-based, que encuentra usuarios similares al usuario objetivo y recomienda ítems que esos usuarios hayan valorado positivamente.
- El enfoque item-based, que identifica ítems similares a aquellos que el usuario ya ha valorado o consumido, y recomienda los más cercanos.

Para calcular la similitud w_{xy} entre dos usuarios o entre dos ítems, se emplean comúnmente medidas como:

- Similitud coseno [8]:

$$w_{xy} = Coseno(x,y) = \frac{\sum_{i \in I_{xy}} x_i y_i}{\sqrt{\sum_{i \in I_{xy}} x_i^2} \cdot \sqrt{\sum_{i \in I_{xy}} y_i^2}}$$

- Correlación de Pearson [8]:

$$w_{xy} = Pearson(x,y) = \frac{\sum_{i \in I_{xy}} (x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sqrt{\sum_{i \in I_{xy}} (x_i - \bar{x})^2} \cdot \sqrt{\sum_{i \in I_{xy}} (y_i - \bar{y})^2}}$$

Donde I_{xy} es el conjunto de ítems valorados por ambos usuarios x y y (o por ambos ítems si se trata de un modelo item-based). Ambas medidas tienen como objetivo identificar patrones compartidos. Mientras que la similitud coseno se centra en la orientación de los vectores, es decir patrones relativos, la correlación de Pearson, en cambio, mide la relación lineal entre las valoraciones normalizadas por su media.

Una vez calculadas las similitudes, la predicción de la relevancia $\hat{r}_{ui}$ se realiza a través de una media ponderada. En el caso de user-based:

$$\hat{r}_{ui} = \frac{\sum_{v \in N_k(u)} w_{uv} \cdot r_{vi}}{\sum_{v \in N_k(u)} |w_{uv}|}$$

Donde $N_k(u)$ es el conjunto de los k vecinos más similares al usuario u. Esta fórmula estima la preferencia del usuario u por el ítem i en función de cómo lo valoraron usuarios similares, ponderando según su similitud.

En el enfoque item-based, se utiliza una fórmula análoga:

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u(i)} w_{ij} \cdot r_{ui}}{\sum_{j \in N_u(i)} |w_{ij}|}$$

Donde $N_u(i)$ es el conjunto de los k ítems más similares al artículo i. En este caso, la predicción se basa en las valoraciones previas del usuario sobre ítems parecidos, ponderadas por su similitud con i. En tareas de ranking top-n, es habitual omitir el denominador de estas fórmulas, ya que al tratarse de una constante positiva para cada usuario o ítem, no afecta al orden relativo de las puntuaciones. Esto permite mejorar la eficiencia computacional sin comprometer la calidad del ranking final.

Estos métodos son intuitivos y eficaces en contextos donde los patrones de comportamiento entre usuarios o ítems están bien definidos, aunque pueden presentar dificultades de escalabilidad y escasez de datos en conjuntos muy grandes.

2.3.4. MATRIX FACTORIZATION

La factorización de matrices es una de las técnicas más usadas en sistemas de recomendación, especialmente cuando hay una gran cantidad de interacciones. Su objetivo es encontrar una representación latente [8] de usuarios e ítems a partir de los patrones observados en la matriz de utilidad R.

La idea principal es asociar a cada usuario $u \in U$ un vector latente de tamaño f (o embedding) $p_u \in \mathbb{R}^f$ y a cada ítem $i \in I$ un vector $q_i \in \mathbb{R}^f$, donde f es un hiperparámetro del sistema. La predicción de la relevancia entre un usuario e ítem se calcula como el producto escalar entre sus vectores:

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

Para aprender estos embeddings, se entrena el modelo minimizando el error cuadrático medio (MSE) [9] entre las predicciones y las valoraciones reales observadas, normalmente con una penalización para evitar que el modelo haga overfitting (regularización L2):

$$L = \sum_{(u,i)\in R} (r_{ui} - \hat{r}_{ui})^2 + \lambda(\|p_u\|^2 + \|q_i\|^2)$$

Además, este modelo puede extenderse para incluir factores temporales que capturen cómo evolucionan las preferencias de los usuarios o la popularidad de los ítems a lo largo del tiempo, como se propuso en [8].

Este modelo es especialmente útil cuando se trabaja con datos explícitos (como puntuaciones), ya que busca predecir directamente el valor de las interacciones. Sin embargo, en escenarios con datos implícitos, donde solo se conoce si una interacción ocurrió o no, este enfoque puede no ser adecuado, y se requieren otros métodos de optimización.

2.3.5. BAYESIAN PERSONALIZED RANKING (BPR)

El Bayesian Personalized Ranking (BPR) [10] es un enfoque diseñado específicamente para sistemas de recomendación con datos implícitos, como clics o compras, donde no existen valoraciones numéricas explícitas. En lugar de predecir puntuaciones como en los recomendadores anteriores, BPR busca aprender un orden de preferencia entre ítems para cada usuario.

El punto de partida del modelo es la observación de que, en los sistemas con feedback implícito, solo se dispone de ejemplos positivos (interacciones observadas), mientras que el resto de los ítems son una mezcla desconocida entre ítems irrelevantes y posibles futuras interacciones. En este contexto, BPR propone aprender directamente un ranking para cada usuario u de tal manera que los ítems con los que ha interactuado se clasifiquen por encima de aquellos con los que no lo ha hecho.

Para ello, el modelo genera tripletas de entrenamiento $D_S \subseteq U \times I \times I$, donde cada tripleta (u, i, j) representa que el usuario u prefiere el ítem i con el que ha interactuado al ítem j con el que no lo ha hecho.

De esta manera, BPR propone maximizar la probabilidad a posteriori $p(\Theta| >_u)$, donde Θ son los parámetros del modelo, y $>_u$ representa el orden de preferencias del usuario u. Este enfoque se concreta en un criterio de optimización denominado BPR-Opt, el cual se obtiene gracias al estimador de máxima a posteriori (MAP):

$$BPR - Opt = \sum_{(u,i,j)\in D_S} \ln\left(\sigma\left(\hat{r}_{ui} - \hat{r}_{uj}\right)\right) + \lambda\|\Theta\|^2$$

Donde $\sigma(x) = \frac{1}{1+e^{-x}}$ es la función sigmoide y λ es un parámetro de regularización. Este criterio busca maximizar la probabilidad de que, para cada tripleta, el modelo ordene correctamente i por encima de j.

Este algoritmo se puede utilizar para optimizar el ranking en distintos tipos de modelos [10] que puedan calcular $\hat{r}_{ui}$, por lo que incluye a los modelos de factorización de matrices $\hat{r}_{ui} = p_u^\top q_i$ y los modelos basados en vecinos cercanos (KNN aprendidos) donde $\hat{r}_{ui}$ se calcula como la suma de las similitudes del ítem i con otros ítems que el usuario ha consumido.

Este tipo de entrenamiento permite mejorar la calidad de las recomendaciones, ya que el criterio de optimización está específicamente diseñado para aprender a ordenar ítems según las preferencias del usuario.

2.3.6. MULTI LAYER PERCEPTRON (MLP)

El modelo MLP (Multi-Layer Perceptron) es una extensión del filtrado colaborativo que utiliza una red neuronal completamente conectada para modelar relaciones no lineales entre usuarios e ítems. A diferencia de los métodos lineales como Matrix Factorization, que asumen una relación lineal entre los embeddings, MLP concatena los vectores de usuario e ítem y los pasa por varias capas densas con activaciones no lineales (ReLU), permitiendo capturar interacciones complejas [11].

Cada usuario $u \in U$ e ítem $i \in I$ se representan con un vector de embedding de dimensión f $p_u, q_i \in \mathbb{R}^f$ respectivamente. Estos vectores se concatenan y se introducen en la red, produciendo una predicción escalar de relevancia:

$$\hat{r}_{ui} = MLP([p_u; q_i])$$

El modelo se entrena con descenso por gradiente mediante la minimización del error cuadrático medio (MSE) entre las predicciones $\hat{r}_{ui}$ y los valores reales r_{ui}.

$$L = \sum_{(u,i)\in R} (\hat{r}_{ui} - r_{ui})^2 + \lambda \|\Theta\|^2$$

Donde Θ representa todos los parámetros entrenables del modelo (embeddings y pesos de la red), y λ es un parámetro de regularización que evita el sobreajuste. Este modelo está especialmente indicado para escenarios en los que existen patrones no lineales en los datos. Sin embargo, su entrenamiento es más costoso y sensible a la configuración de hiperparámetros como la arquitectura de la red (número y tamaño de capas), el número de epochs, o la tasa de aprendizaje [12].

2.3.7. GRAPH NEURAL NETWORK (GNN)

Las Graph Neural Networks (GNN) representan un enfoque más avanzado en el ámbito de los sistemas de recomendación, especialmente útiles cuando la estructura del grafo de interacciones (usuario–ítem) puede aprovecharse como fuente de información adicional. En este trabajo, se ha implementado una variante simple basada en Graph Convolutional Networks (GCN) [13], [14], aplicada sobre un grafo bipartito usuario–ítem. En este modelo, la información se propaga entre nodos del grafo para refinar los embeddings latentes de tamaño f de usuarios e ítems $p_u, q_i \in \mathbb{R}^f$. Sea $E^{(0)} = [p_1, \dots, p_m, q_1, \dots, q_d]$ la matriz inicial de embeddings. En cada capa de propagación se actualiza de la siguiente manera:

$$E^{(k+1)} = \hat{A} \cdot E^{(k)}$$

Donde Â es la matriz de adyacencia normalizada. Para obtener los embeddings finales E^* que se utilizarán en la predicción de los artículos, se promedian las matrices de todas las capas. Es decir:

$$E^* = \frac{1}{L+1}\sum_{k=1}^{L} E^{(k)} = [p_1^*, \dots, p_m^*, q_1^*, \dots, q_d^*]$$

Siendo L un hiperparámetro del modelo indicando el número de capas que tiene. La predicción se realiza mediante producto interno entre los vectores propagados de usuario e ítem:

$$\hat{r}_{ui} = (p_u^*)^\top \cdot q_i^*$$

Entrenado con MSE, el modelo permite aprender representaciones enriquecidas por los contextos locales del grafo [12].

2.4. MÉTRICAS DE EVALUACIÓN

La evaluación de los sistemas de recomendación es un aspecto clave para entender su rendimiento global y su impacto en la experiencia del usuario. En el pasado, el enfoque principal de la evaluación estaba puesto en las métricas de relevancia, es decir, en qué medida el sistema es capaz de recomendar ítems que de verdad interesan al usuario. Sin embargo, en los últimos años han ganado importancia otras dimensiones como la novedad y la diversidad, que buscan mejorar la experiencia del usuario más allá de la precisión.

A continuación, se describen las principales métricas [9] utilizadas en la evaluación de sistemas de recomendación.

2.4.1. MÉTRICAS DE RELEVANCIA

Estas métricas miden la capacidad del sistema para sugerir ítems relevantes para el usuario, comparando las recomendaciones generadas con un conjunto de ítems presentes en el conjunto de test $T(u)$.

Precisión

La precisión evalúa qué proporción de los elementos recomendados al usuario está entre los elementos valorados en el conjunto de test. Se calcula como:

$$precision@n(u) = \frac{|L_n(u) \cap T(u)|}{n}$$

Donde n representa el tamaño de la lista de recomendación generada para cada usuario. La precisión toma valores entre 0 y 1, donde 1 indica que todos los ítems recomendados han sido relevantes según el conjunto de test [9].

Recall

Esta métrica evalúa que proporción de los ítems del conjunto de test valorados por el usuario han sido recomendaciones. Se define como:

$$recall@n(u) = \frac{|L_n(u) \cap T(u)|}{|T(u)|}$$

El recall varía entre 0 y 1, donde valores cercanos a 1 implican que el sistema ha logrado recuperar la mayor parte de los ítems relevantes existentes [9].

Normalized Discounted Cumulative Gain (NDCG)

Evalúa la calidad del orden de las recomendaciones teniendo en cuenta la relevancia y la posición de los ítems. Penaliza los aciertos que aparecen en posiciones más bajas. Se calcula con:

$$NDCG = \frac{1}{IDCG@n(u)} \sum_{x=1}^{n} \frac{rel_x}{\log_2(x+1)}$$

Donde rel_x es 1 si el ítem en la posición x es relevante, es decir, si está en el conjunto de test $T(u)$ y 0 en caso contrario.

NDCG también está acotado entre 0 y 1. Valores cercanos a 1 reflejan rankings donde los ítems relevantes aparecen en posiciones altas, lo que indica un ordenamiento más efectivo [9].

2.4.2. MÉTRICAS DE NOVEDAD Y DIVERSIDAD

Además de evaluar la relevancia de las recomendaciones, es importante analizar hasta qué punto el sistema ofrece novedad (ítems poco conocidos o explorados) y diversidad (variedad de contenido). Estas métricas contribuyen a mejorar la experiencia del usuario, evitando recomendaciones redundantes o predecibles.

Expected Popularity Complement (EPC)

Esta métrica mide la novedad de las recomendaciones, valorando más la recomendación de ítems menos populares. Se calcula de la siguiente manera:

$$EPC = \frac{1}{n} \sum_{i \in L_n(u)} \left(1 - \frac{pop(i)}{|U|}\right)$$

Observando la fórmula podemos ver que valores más cercanos a uno indican recomendaciones menos populares y, por tanto, más novedosas, y valores cercanos a cero implican una fuerte dependencia de ítems populares.

La fórmula original de EPC incluye además un factor de descuento posicional y un modelo de relevancia para combinar novedad y precisión [15]. Sin embargo, en este trabajo se ha optado por utilizar esta versión simplificada de la métrica [16], centrada exclusivamente en la popularidad, ya que la relevancia se mide por separado con métricas específicas como la precisión, recall o NDCG.

Aggregate Diversity

Mide cuántos ítems diferentes se han recomendado en total al conjunto completo de usuarios. Se calcula como:

$$AggregateDiversity = \frac{|\cup_{u \in U} L_n(u)|}{\sum_{u \in U} |L_n(u)|}$$

El numerador representa el número de ítems únicos recomendados, y el denominador el total de recomendaciones generadas. Cuanto más cercano a 1, mayor es la diversidad global del sistema. Cabe destacar que esta métrica, tal como se propuso originalmente, no está normalizada, pero en este trabajo se ha normalizado dividiendo entre el total de recomendaciones posibles, con el objetivo de facilitar la comparación entre distintos sistemas de recomendación.

Gini Index

Evalúa la desigualdad en la distribución de los ítems recomendados. Cuanto más bajo es el índice de Gini, más parecidas serán las probabilidades de recomendar cada ítem. Se calcula de la siguiente manera:

$$G = \frac{1}{n - 1} \sum_{j=1}^{n} (2j - n - 1) p(i_j)$$

Donde:

- $i_1, i_2, \ldots, i_n$ es la lista de los ítems ordenados en orden creciente en función de $p(i_j)$
- $p(i_j)$ es la proporción de veces que el ítem i ha sido recomendado entre todos los usuarios, dado por:

$$p(i_j) = \frac{\sum_{u \in U} 1_{\{i_j \in L_n(u)\}}}{\sum_{u \in U} |L_n(u)|}$$

Un Gini cercano a 0 indica una distribución uniforme (todos los ítems tienen similares probabilidades de ser recomendados), mientras que un Gini cercano a 1 implica que solo unos pocos ítems concentran la mayoría de las recomendaciones.

2.5. MÉTODOS DE DIVISIÓN DE DATOS

Existen diversas estrategias de partición comúnmente utilizadas para dividir los datos en conjuntos de entrenamiento y test en sistemas de recomendación, cada una con sus ventajas e inconvenientes [17].

- Random Split: Realiza una partición aleatoria de las interacciones en entrenamiento y test. Es sencilla de implementar y reproducible (si se fija una semilla), pero no respeta la dimensión temporal, lo que puede distorsionar la evaluación en contextos donde el orden cronológico es relevante.
- Leave-One-Out: Reserva la última interacción de cada usuario como conjunto de test. Esta estrategia es ampliamente utilizada en tareas top-n, ya que maximiza el uso de datos para entrenamiento. Sin embargo, puede introducir *temporal leakage* si los datos no están correctamente ordenados por tiempo.
- Temporal User Split: Para cada usuario, ordena sus interacciones cronológicamente y reserva un porcentaje fijo de las últimas como test. Permite controlar el tamaño del conjunto de prueba individualmente, pero, al igual que la anterior, no establece un límite temporal común para todos los usuarios, lo que puede restar realismo a la evaluación.

- Temporal Global Split: Ordena todas las interacciones del conjunto de datos según su marca temporal y utiliza las primeras para entrenamiento y las más recientes para test. Esta técnica, considerada la más rigurosa desde el punto de vista temporal, evita la fuga de información, aunque puede reducir significativamente el número de usuarios e ítems presentes en ambos conjuntos, afectando a la cobertura y dificultando la evaluación.

3. METODOLOGÍA

3.1. JUSTIFICACIÓN

La motivación principal del proyecto se basa en la necesidad técnica y práctica de contar con una librería en Python que esté específicamente diseñada para implementar, evaluar y comparar sistemas de recomendación de una manera eficiente y sencilla. Aunque ya hay varias librerías de recomendación disponibles, muchas de ellas tienen limitaciones en cuanto a su capacidad de generalización y abstracción, especialmente cuando se trata de manejar diferentes tipos de datos, modelos y métricas de evaluación.

El objetivo de esta nueva librería es superar esas limitaciones al ofrecer una interfaz común y abstracta que facilite la gestión de diversos conjuntos de datos, desde valoraciones de películas hasta recomendaciones de productos o lugares de interés. Esta abstracción permite replicar resultados de manera sencilla y facilita la comparación directa entre distintos algoritmos de recomendación. Además, al incluir métricas como relevancia y diversidad, la librería podrá analizar el rendimiento de cada recomendador, lo que resulta realmente útil en tareas de investigación y comparación de sistemas de recomendación.

3.2. HERRAMIENTAS

El desarrollo de este proyecto se realizará en Python, ya que este lenguaje de programación es muy popular en las áreas del Machine Learning y Data Science, lo que facilita el futuro uso de la librería. Para el control de versiones se ha empleado Git junto con GitHub, lo que ha permitido mantener un seguimiento ordenado del código fuente y facilitar su distribución. Se puede encontrar el código del proyecto en el siguiente enlace: github.com/CataRVS/RecommenderSystems.

Durante el desarrollo se han utilizado entornos virtuales para facilitar la reproducibilidad tanto en ordenadores personales como en equipos de la universidad.

3.3. DISEÑO

El proyecto sigue una estructura modular y ordenada, diseñada para separar claramente las distintas responsabilidades del sistema. La raíz del repositorio está organizada en carpetas dedicadas a los datos, los resultados y el código fuente. En primer lugar, se encuentran los directorios *data/* y *results/*, que almacenan respectivamente los conjuntos de datos utilizados y las salidas generadas por los modelos (como métricas, recomendaciones y gráficos) separados por datasets. El núcleo funcional del sistema se encuentra en *src/*, que agrupa los distintos módulos encargados de la carga de datos, implementación de algoritmos de recomendación, evaluación de resultados y utilidades generales. Además, se ha incluido una carpeta de *pipelines/* pensada para automatizar procesos como el preprocesamiento, el entrenamiento y la evaluación, facilitando la ejecución de experimentos completos. Por último, la carpeta *commands/* contiene ficheros de texto con una gran cantidad de ejemplos de comandos empleados para entrenar o evaluar los modelos seleccionados. La estructura en detalle se puede encontrar a continuación:

```
RecommenderSystems
├──commands/
│   ├──evaluation/
│   └──train/
├──data/
│   ├──ml-100k/
│   └──NewYork/
├──results/
│   ├──graphs/
│   │   ├──ml-100k/
│   │   └──NewYork_10/
│   ├──metrics/
│   │   ├──ml-100k/
│   │   └──NewYork_10/
│   └──recommendations/
│       ├──ml-100k/
│       └──NewYork_10/
└──src/
    ├──datamodule/
    │       __init__.py
    │       data.py
    │       splits.py
    ├──evaluation/
    │       __init__.py
    │       evaluation.py
    ├──main/
    │       main_evaluate.py
    │       main_recommend.py
    ├──pipelines/
    │   ├──data_processing/
    │   │       kcore.py
    │   ├──evaluate/
    │   │       evaluate_lot.py
    │   │       plot_lot.py
    │   └──train/
    │           train_lot.py
    ├──recommenders/
    │       __init__.py
    │       basic_recommenders.py
    │       knn.py
    │       matrix_factorisation.py
    │       neural_networks.py
    └──utils/
            __init__.py
            datasets.py
            similarities.py
            strategies.py
            utils.py
```

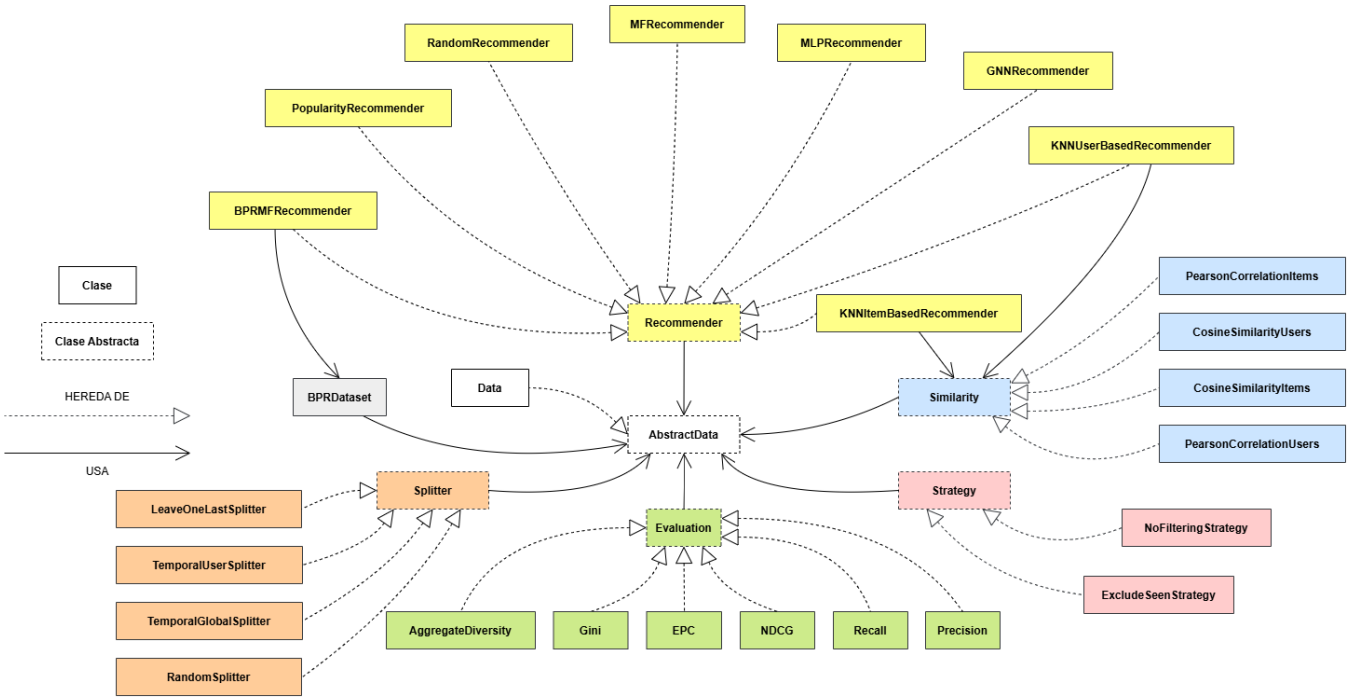

Figura 1: Diagrama mostrando la composición y la relación entre módulos y clases. Se muestra en blanco datamodule.data, en naranja datamodule.splits en amarillo recommenders, en verde evaluation, en azul utils.similarities, en rojo utils.strategies y finalmente en gris utils.datasets

3.3.1. DATAMODULE

En primer lugar, tenemos el módulo de *datamodule/*. Este contiene la clase central *Data*, que se encarga de la carga, procesamiento, representación y acceso estructurado a los datos de interacción usuario-ítem. Esta clase hereda de una clase abstracta *AbstractData*, lo que permite establecer una interfaz clara y reutilizable para futuras implementaciones sobre otros tipos de datos.

Este módulo implementa lógica avanzada como:

- Mapeo interno de IDs: convierte los identificadores originales de usuarios e ítems a índices internos para un procesamiento más eficiente, y mantiene los mapeos inversos.
- Preprocesamiento controlado: permite cargar datasets ya particionados (train/test) o dividir uno automáticamente.
- Representación en formato disperso: transforma las interacciones en una matriz dispersa útil para algoritmos matriciales y de optimización.
- Acceso eficiente a interacciones: ofrece funciones para consultar ítems valorados por un usuario, usuarios que han interactuado con un ítem, así como sus respectivos índices y valores.
- Compatibilidad con test y recomendaciones: permite recuperar interacciones de test de un usuario y cargar archivos externos de recomendaciones para evaluación.

Además, abstrae detalles como el delimitador, el formato del fichero o si se debe ignorar la primera línea del archivo, facilitando así la flexibilidad en distintos entornos y datasets.

Una parte adicional importante dentro del módulo es la gestión de estrategias de partición de datos, implementada en el archivo *splits.py*. Este componente define varias clases de partición que permiten dividir

los datos en entrenamiento y test de forma configurable. Se incluyen estrategias como *Random Split*, *Leave-One-Out*, *Temporal User Split* y *Temporal Global Split*, que cubren desde enfoques aleatorios hasta divisiones informadas por el orden temporal. Todas estas estrategias heredan de una clase abstracta común (*Splitter*), lo que facilita su reutilización, extensión y selección según las necesidades del experimento.

Este módulo es un elemento clave del proyecto, ya que concentra y resume toda la lógica relacionada con la estructura de los datos, lo que permite que los modelos y evaluadores trabajen sobre una base homogénea, robusta y eficiente.

3.3.2. RECOMMENDERS

El módulo *recommenders/* contiene todas las implementaciones de los algoritmos de recomendación utilizados en el proyecto. La estructura del código sigue un enfoque orientado a objetos, donde cada recomendador hereda de una clase abstracta base llamada *Recommender*, definida en *basic_recommenders.py*. Esta clase base define una interfaz común a través del método *recommend()*, lo que garantiza consistencia en la forma en que los distintos algoritmos generan recomendaciones.

Cada recomendador implementa su propia lógica interna, pero todos reciben como entrada un *user_id* y una instancia de *Strategy* (que filtra los ítems candidatos) y devuelven una instancia *Recommendation* con los ítems recomendados y sus puntuaciones. Se han programado los siguientes recomendadores:

Recomendadores simples (PopularityRecommender, RandomRecommender)

Calculan scores directamente sobre los ítems candidatos (por recuento o aleatoriedad). No requieren entrenamiento previo ni almacenamiento de estado.

KNN (KNNUserBasedRecommender, KNNItemBasedRecommender)

Precalculan matrices de similitud para ponderar valoraciones vecinas. La similitud se consulta directamente sobre las matrices precomputadas, lo que permite eficiencia en la recomendación sin recalcular en cada llamada.

Matrix Factorization (MFRecommender)

Utiliza PyTorch para entrenar embeddings de usuarios e ítems con *MSELoss*. El entrenamiento se ejecuta en el constructor, y los vectores resultantes (user_factors, item_factors) se almacenan como arrays NumPy. La recomendación se basa en el *dot product* entre el vector del usuario y los vectores de los ítems candidatos, usando solo operaciones NumPy.

BPRMFRecommender

Entrena sobre tripletas usuario–ítem positivo–ítem negativo, usando la función de pérdida BPR. Para ello usa un BPRDataset creado en otro módulo que se encarga de generar estas tripletas a partir de la matriz de entrenamiento. Al igual que el modelo MF, guarda los embeddings entrenados como matrices NumPy y utiliza *dot products* para generar las valoraciones.

Multi-Layer Perceptron (MLPRecommender)

Este recomendador utiliza un perceptrón multicapa que combina los embeddings de usuario e ítem como entrada a una red neuronal entrenada con pérdida MSE. Los vectores de embedding se concatenan y se introducen en una red configurable compuesta por capas lineales con activaciones ReLU, definida por la lista configurable *hidden_dims*.

El modelo se entrena con PyTorch usando el optimizador AdamW y guarda los embeddings finales como arrays de NumPy para acelerar la inferencia. Durante la recomendación, se calculan las puntuaciones de los ítems candidatos mediante inferencia por lotes, seleccionando los de mayor puntuación. Esta arquitectura permite modelar relaciones no lineales entre usuarios e ítems, aunque requiere mayor coste computacional y ajuste de hiperparámetros frente a modelos más simples como MF o KNN.

Graph Neural Network (GNNRecommender)

El *GNNRecommender* utiliza una red de convolución sobre grafos (GCN) para refinar los embeddings de usuario e ítem mediante propagación de mensajes en un grafo bipartito. Primero, construye una matriz de adyacencia dispersa y normalizada que representa las interacciones entre usuarios e ítems. En cada capa de la red, los embeddings de todos los nodos (usuarios e ítems) se actualizan mediante multiplicación dispersa con la matriz de adyacencia.

Durante el entrenamiento, los embeddings finales se utilizan para predecir las interacciones observadas a través del producto escalar, y se optimiza el error MSE. La propagación se repite varias veces (número de capas), y al final se calcula la media de los embeddings intermedios para obtener una representación más estable.

Finalmente, en la fase de recomendación, se usan los embeddings precomputados para generar predicciones por producto escalar, seleccionando los ítems con mayor puntuación para cada usuario.

3.3.3. EVALUATION

El módulo *evaluation/* se encarga de implementar las métricas que permiten cuantificar el rendimiento de los distintos sistemas de recomendación desarrollados. Al igual que en los módulos anteriores, se sigue una estructura orientada a objetos basada en herencia, donde todas las métricas derivan de una clase abstracta *Evaluation*. Esta clase define la interfaz común *evaluate()*, que recibe como entrada un fichero CSV con recomendaciones generadas por algún modelo y devuelve un valor escalar con el resultado de la métrica.

Este enfoque permite evaluar cualquier recomendador sin necesidad de modificar su lógica interna, siempre que las recomendaciones se ajusten al formato esperado. La estructura modular de este componente permite extender fácilmente la librería con nuevas métricas en el futuro, manteniendo una interfaz uniforme.

El archivo principal *evaluation.py* contiene todas las métricas implementadas. Estas cubren tanto métricas de relevancia (precisión, recall, NDCG), como métricas de novedad y diversidad (EPC, Aggregate Diversity y Gini Index). Todas ellas utilizan funciones del módulo Data para acceder tanto a los datos de test reales (ground truth) como a las recomendaciones generadas desde ficheros externos. La función *_load_recs()* permite centralizar el proceso de lectura y formateo del archivo de recomendaciones, convirtiéndolo en un diccionario *{usuario: [ítems recomendados]}*.

Cada métrica itera usuario por usuario, comparando los ítems recomendados con los ítems relevantes reales. En el caso de NDCG, se penaliza el orden incorrecto de los ítems relevantes. Para EPC, se calcula la popularidad de cada ítem sobre el conjunto de entrenamiento, y se valora más la aparición de ítems menos frecuentes. La diversidad agregada mide cuántos ítems distintos han sido recomendados, y Gini evalúa si el sistema concentra sus recomendaciones en pocos ítems o las distribuye equitativamente.

Este módulo permite, por tanto, cuantificar múltiples aspectos del comportamiento de los sistemas de recomendación implementados, no solo en cuanto a precisión sino también desde una perspectiva de personalización y cobertura del catálogo. Gracias a su diseño desacoplado y su integración con la clase Data, la evaluación es sencilla, extensible y coherente a lo largo de todo el flujo experimental.

3.3.4. UTILS

El módulo *utils/* proporciona funciones y clases auxiliares necesarias para el funcionamiento del sistema. Entre ellas, destaca *Recommendation*, una clase que encapsula las recomendaciones generadas por los modelos y permite almacenarlas, mostrarlas o exportarlas fácilmente. También se incluye *set_seed()*, una función para fijar la semilla aleatoria y garantizar la reproducibilidad de resultados.

El archivo *datasets.py* contiene la clase *BPRDataset*, un generador de tripletas (usuario, ítem positivo, ítem negativo) para el entrenamiento del modelo BPRMF. Esta clase gestiona las interacciones positivas y realiza el muestreo negativo en cada batch.

En *similarities.py* se implementan las medidas de similitud necesarias para los recomendadores KNN. Se incluyen versiones coseno y Pearson, tanto para usuarios como para ítems. Todas las clases precalculan matrices de similitud utilizando estructuras dispersas, lo que mejora significativamente el rendimiento.

Por último, *strategies.py* define estrategias para filtrar los ítems candidatos antes de la recomendación. Se incluyen estrategias que devuelven todos los ítems (*NoFilteringStrategy*) o que excluyen los ya vistos por el usuario (*ExcludeSeenStrategy*), aportando flexibilidad al sistema sin modificar los modelos.

3.3.5. MAIN

La carpeta *main/* contiene los scripts de entrada para ejecutar el sistema desde la línea de comandos. Estos ficheros permiten automatizar el proceso completo de recomendación y evaluación, facilitando la interacción con los modelos y la ejecución de experimentos.

El archivo *main_recommend.py* permite generar recomendaciones usando cualquiera de los modelos implementados. Mediante argumentos por línea de comandos, el usuario puede seleccionar el tipo de recomendador (popularidad, aleatorio, KNN, MF, BPRMF), la estrategia de filtrado (como excluir ítems ya vistos), y ajustar los hiperparámetros específicos (número de vecinos, tipo de similitud, número de factores latentes, tasa de aprendizaje, regularización, etc.). Internamente, este script se encarga de construir la instancia del modelo, aplicar la estrategia seleccionada, recorrer todos los usuarios de test y guardar los resultados en un archivo .csv.

Por su parte, *main_evaluate.py* está diseñado para calcular métricas de evaluación sobre un archivo de recomendaciones previamente generado. El script admite múltiples métricas (precisión, recall, NDCG, EPC, Gini, diversidad agregada), permitiendo así comparar distintos modelos desde distintas perspectivas. Ambas

interfaces están orientadas a facilitar la experimentación rápida, con un enfoque reproducible y controlado a través de argumentos como la semilla aleatoria, separadores de datos o nombres de columnas.

3.3.6. PIPELINES

La carpeta *pipelines/* incluye scripts diseñados para automatizar tareas frecuentes durante el flujo de trabajo, como el preprocesamiento, la generación masiva de recomendaciones y la evaluación de resultados. Esta organización permite lanzar experimentos de forma más eficiente y replicable.

El script *data_processing/kcore.py* aplica un filtrado de k-core sobre un conjunto de datos, asegurando que todos los usuarios e ítems tengan al menos k interacciones. Este tipo de preprocesamiento es común en sistemas de recomendación para reducir la dispersión de los datos y garantizar un mínimo de información por usuario y por ítem.

En cuanto a entrenamiento, *train/train_lot.py* permite realizar una búsqueda sistemática de hiperparámetros (*grid search*) sobre distintas configuraciones de los modelos implementados, incluyendo MF, BPRMF, KNN user-based, KNN item-based, MLP y GNN. Internamente, construye los comandos necesarios e invoca automáticamente el script *main_recommend.py* con los parámetros adecuados. De este modo, permite evaluar de forma sistemática múltiples configuraciones sin intervención manual.

Por su parte, para evaluar los sistemas procesados, *evaluate/evaluate_lot.py* permite evaluar automáticamente un lote de ficheros de recomendaciones pertenecientes a un tipo específico de modelo (MF, BPRMF, KNN user-based, KNN item-based, MLP o GNN). Analiza los nombres de archivo para extraer los parámetros utilizados en cada experimento, calcula múltiples métricas sobre cada fichero (precision, recall, ndcg, epc, gini, aggregate_diversity) y exporta los resultados en un CSV resumido. También muestra una previsualización de los modelos con mejor precisión.

Finalmente, *evaluate/plot_lot.py* se encarga de generar gráficos a partir de los resultados de evaluación obtenidos por el fichero anterior para facilitar su análisis.

3.3.7. EXTENSIBILIDAD Y CONFIGURACIÓN

El sistema está diseñado para ser fácilmente extensible. Para añadir un nuevo recomendador, basta con crear una nueva clase que herede de *Recommender* y sobrescriba el método *recommend()*, siguiendo la interfaz común del proyecto. De igual forma, incorporar nuevas métricas solo requiere definir una subclase de *Evaluation* e implementar el método *evaluate()*.

Del mismo modo, si en el día de mañana las librerías de pandas o de scipy se volviesen obsoletas, o simplemente se prefiriera trabajar los datos con otra librería, bastaría con crear una nueva clase *Data* que herede de *AbstractData* e implementase todas las funciones que tiene descritas.

En cuanto a la configuración de hiperparámetros, como el número de factores o de vecinos, la tasa de aprendizaje o la regularización, se realiza mediante argumentos desde línea de comandos en los scripts del módulo de *main*, lo que facilita la ejecución sistemática de experimentos.

Finalmente, si se quisiese probar con un dataset distinto, simplemente se debería preprocesar de tal manera que al consumirlo con la librería tenga formato de tabla en la que las cuatro primeras columnas del fichero sean, en este orden: *usuario, ítem, rating, timestamp*.

3.4. OBJETIVOS

3.4.1. OBJETIVOS PRINCIPALES

Objetivo base 1: Desarrollar una librería funcional en Python encargada de implementar distintos sistemas de recomendación y proporcionar métricas que permitan la evaluación de la calidad de estos.

> **Subobjetivo 1**: Desarrollar un módulo para el tratado de datos, con el que trabajen el resto de las secciones para poder abstraerse del uso de librerías en concreto. Tratará con los usuarios, los artículos y otras estructuras de datos necesarias para el correcto funcionamiento de la librería.
>
> **Subobjetivo 2**: Desarrollar otro módulo para la creación de recomendadores, es decir, los modelos con los que se harán las recomendaciones. El objetivo es programar desde recomendadores más básicos como el de popularidad, hasta algunos más novedosos y actuales.
>
> **Subobjetivo 3**: Desarrollar un tercer módulo para la evaluación de las recomendaciones hechas por los recomendadores. Se analizarán los datos tanto en términos de relevancia como en términos de novedad y diversidad.

Objetivo 2: Realizar un análisis profundo y específico sobre el rendimiento de cada recomendador en función de cada métrica, del entrenamiento necesario para cada uno de ellos y de los resultados obtenidos.

3.4.2. OBJETIVOS SECUNDARIOS

Objetivo secundario 1: Desarrollar un recomendador basado en redes neuronales, entrenarlo y analizar los resultados.

4. EXPERIMENTOS

4.1. CONJUNTO DE DATOS

Para evaluar y comparar el rendimiento de los distintos sistemas de recomendación implementados en este proyecto, se han utilizado dos conjuntos de datos conocidos en el ámbito de investigación en sistemas de recomendación:

- MovieLens 100K: dataset con valoraciones explícitas de películas.
- Foursquare Global-scale Check-in Dataset: dataset con interacciones implícitas de check-ins de puntos de interés.

4.1.1. MOVIELENS 100K DATASET

Este conjunto, proporcionado por GroupLens [1], contiene 100.000 valoraciones realizadas por 943 usuarios sobre 1.682 películas, con puntuaciones de 1 a 5. Cada usuario ha puntuado al menos 20 ítems, lo que garantiza una densidad mínima adecuada para entrenamiento y evaluación.

Aunque incluye información demográfica y de contenido, en este trabajo se ha utilizado únicamente la matriz de interacciones, enfocando el análisis en filtrado colaborativo puro. Para la evaluación, se ha usado la primera de las cinco particiones disponibles (u1).

4.1.2. FOURSQUARE CHECK-IN DATASET

El segundo dataset es el Global-scale Check-in Dataset de Foursquare, creado por Dingqi Yang y otros colaboradores [2], [3]. Este conjunto contiene información sobre check-ins realizados por usuarios en distintos puntos de interés (POI) alrededor del mundo entre abril de 2012 y septiembre de 2013. En total, contiene 33.278.683 check-ins de 266.909 usuarios sobre 3.680.126 POIs de 415 ciudades diferentes alrededor del mundo.

Para este proyecto, se ha utilizado un subconjunto preprocesado de este dataset con únicamente las entradas de la ciudad de Nueva York. Los datos están organizados en dos ficheros: uno con las coordenadas geográficas de los lugares (POIs) y otro con los check-ins realizados. Este último incluye columnas con identificador del usuario, identificador del POI, un valor indicativo de la visita (1.0), y dos marcas temporales: una en formato UTC y otra en la hora local de Nueva York. Aunque contábamos con ambos ficheros, como en el caso del dataset anterior, solo se ha utilizado el que contiene las interacciones entre los diferentes usuarios y los puntos de interés.

Este fichero contiene las 380.247 interacciones de 15.785 usuarios que han visitado 41.386 POIs diferentes. A diferencia de MovieLens, este conjunto representa datos implícitos, ya que solo indica si un usuario ha visitado un lugar, sin tener información explícita sobre su opinión. Debido a esto, en este dataset puede haber múltiples visitas del mismo usuario al mismo POI. Se evaluó el impacto de agregar o no estas interacciones repetidas durante el entrenamiento y se observó que en varios modelos la relevancia de las recomendaciones era significativamente mejor si no se agregaban los resultados y simplemente se mantenían los ficheros con las interacciones originales.

Incluso con esta versión reducida, el dataset cuenta con un número especialmente grande de interacciones, usuarios e ítems, por lo que, en un segundo preprocesamiento de datos, se ha optado por realizar un filtrado por k-core con $k = 10$. Esto significa que en nuestro conjunto de datos solo hemos mantenido a los usuarios y a los ítems que cada uno haya tenido al menos k (en este caso 10) interacciones. Este filtrado elimina usuarios e ítems con escasa actividad, lo que reduce la dispersión de los datos y mejora la eficiencia computacional. Así, se preservan interacciones representativas que permiten entrenar y evaluar los modelos de forma más eficiente y fiable.

4.2. CONFIGURACIÓN

La implementación del sistema se ha llevado a cabo utilizando Python, concretamente en las versiones Python 3.11.9 y 3.12.3.

En cuanto a las librerías utilizadas, destacan NumPy (versión 2.1.3) y Pandas (2.2.3) para el procesamiento y análisis de datos, y SciPy (1.15.2) para cálculos numéricos avanzados, especialmente útil para trabajar con estructuras de datos dispersas que permiten acelerar el entrenamiento y reducir el consumo de memoria. Para la construcción de modelos más complejos basados en redes neuronales, se emplea PyTorch (2.5.1), complementado con TensorBoard (2.19.0) para la visualización del entrenamiento del modelo. Además, se utiliza tqdm (4.67.1) para mostrar el progreso de los procesos de forma clara y dinámica, y matplotlib (3.10.3) para la generación de los gráficos de los resultados obtenidos. Todas estas dependencias necesarias para el proyecto se recogerán en un archivo requirements.txt, para facilitar la instalación y para poder replicar el entorno de ejecución.

Para la gestión y control de versiones del código fuente, se ha utilizado Git junto con GitHub, lo que ayuda a manejar el desarrollo de la librería y su descarga y distribución futura una vez finalizado el proyecto.

Finalmente, en cuanto al hardware utilizado, se ha hecho uso de un ordenador personal con procesador 11th Gen Intel® Core™ i7 y con 16GB de RAM, y un ordenador proporcionado por la universidad con procesador 13th Gen Intel® Core™ i7, 64GB de RAM y una GPU GeForce RTX 4070 Ti.

4.3. ANÁLISIS DE RENDIMIENTO

El rendimiento de los modelos se ha evaluado considerando tres dimensiones: relevancia, novedad y diversidad, utilizando los datasets MovieLens 100K (valoraciones explícitas) y Foursquare New York (interacciones implícitas).

Se han empleado las siguientes métricas. De relevancia: precisión, recall y NDCG, de novedad, el Expected Popularity Complement (EPC) y de diversidad: aggregate diversity y el índice de Gini.

4.4. DISEÑO EXPERIMENTAL

Para evaluar el rendimiento de los distintos modelos de recomendación, se han realizado experimentos independientes sobre los dos conjuntos de datos seleccionados: MovieLens 100K y Foursquare New York. Ambos han sido tratados y evaluados siguiendo una estrategia coherente, aunque ligeramente diferente para adaptarse a la naturaleza de los datos.

4.4.1. PARTICIONES DE ENTRENAMIENTO Y TEST

En el caso de MovieLens 100K, se ha utilizado la partición predefinida u1, que incluye dos ficheros separados: u1.base (conjunto de entrenamiento) y u1.test (conjunto de test), garantizando que todos los usuarios estén representados en ambos subconjuntos.

En el caso de Foursquare New York, dado que no existe una partición oficial, se ha realizado una división aleatoria global: se ha reservado el 10% de las interacciones de todo el conjunto como conjunto de test.

4.4.2. TAMAÑO DE LA LISTA DE RECOMENDACIÓN n (CUTOFF)

En todos los experimentos, se ha fijado un cutoff de $n = 5$, es decir, se generan listas de los 5 ítems con mayor puntuación para cada usuario. Las métricas se calculan comparando esta lista con el conjunto de ítems relevantes en test.

4.4.3. ESTRATEGIA DE SELECCIÓN DEL CONJUNTO DE CANDIDATOS

Para definir el conjunto de ítems candidatos a recomendar, se ha seguido una estrategia distinta para cada dataset:

En MovieLens, se ha utilizado la estrategia de excluir los artículos vistos, es decir, a un usuario no se le puede recomendar un ítem que ya haya consumido en el conjunto de entrenamiento. Esto se debe a que, en este dataset, los usuarios no consumen un mismo ítem más de una vez.

En cambio, en el dataset de Foursquare, se ha optado por no filtrar los ítems, es decir, que todos los ítems del catálogo sean candidatos. Esto se debe a que, al ser un dataset de datos implícitos, que solo indica cuando

un usuario visita un sitio, las interacciones pueden estar repetidas, es decir, un usuario puede visitar un sitio más de una vez. Debido a esto, en el archivo de test puede estar presente que un usuario visita un POI que ya ha visitado en el fichero de train.

4.4.4. OPTIMIZACIÓN DE HIPERPARÁMETROS

La selección de modelos se ha realizado mediante una exploración sistemática de combinaciones de hiperparámetros. Cada configuración genera un fichero de recomendaciones, que se evalúa utilizando todas las métricas definidas en el sistema. A partir de esos resultados, se seleccionan las 5 configuraciones con mayor precisión como criterio principal, y se presentan en las tablas de resultados del apartado correspondiente. En la Tabla 1 se pueden observar los diferentes hiperparámetros probados:

Recomendador	Hiperparámetro	Valores probados en MovieLens 100k	Valores probados en Foursquare New York
KNN User-based	k	1, 5, 10, 25, 50, 100	1, 5, 10, 25, 50, 100
	Similitud	Cosine, Pearson	Cosine, Pearson
KNN Item-based	k	1, 5, 10, 25, 50, 100	1, 5, 10, 25, 50, 100
	Similitud	Cosine, Pearson	Cosine, Pearson
MF	Nº de factores	16, 64, 128, 264	8, 16, 512, 1024, 2048
	Learning rate	0.005, 0.01	0.005, 0.075, 0.01
	Regularización (λ)	0.01, 0.1	0.01, 0.1, 0.15, 0.2, 0.25, 0.3
	Epochs	50	25, 50, 100, 150, 200, 250
	Batch size	4096	512, 1024, 2048, 4096
BPRMF	Nº de factores	16, 64, 128, 264	32, 64, 128
	Learning rate	0.005, 0.01	0.005, 0.01
	Regularización (λ)	0.01, 0.1	0.002, 0.01, 0.02, 0.1
	Epochs	50	50, 100, 500
	Batch size	4096	2048
MLP	Nº de factores	32, 64, 256, 512	256, 512
	Learning rate	0.01, 0.005	0.005, 0.01
	Regularización (λ)	0.01, 0.1	0.01, 0.1
	Hidden layers	(512, 256), (256, 128), (256, 128, 64)	(128, 64), (256, 128), (256, 128, 64)
	Epochs	20, 30, 150, 200	100, 250
	Batch size	2048, 4096	2048
GNN	Nº de factores	128, 256, 264	256, 502
	Learning rate	0.001, 0.005, 0.01	0.001, 0.005, 0.01, 0.1
	Regularización (λ)	0.01, 0.1	0.01, 0.1
	Nº de capas	2, 3	2, 3
	Epochs	50, 100	50, 100
	Batch size	2048	2048, 4096

Tabla 1: Tabla mostrando los diferentes hiperparámetros probados para los diferentes algoritmos y diferentes datasets.

5. RESULTADOS

5.1. INTRODUCCIÓN

En esta sección se presentan los resultados obtenidos al evaluar los distintos modelos de recomendación implementados. Los resultados se organizan por dataset. Para cada uno, primero se comparan los mejores resultados de cada modelo, y después se muestra cómo afectan los hiperparámetros al rendimiento de cada modelo. Para ello, se disponen los resultados en forma de tablas y gráficos.

5.2. *DATASET MOVIELENS 100K*

5.2.1. COMPARATIVA GENERAL ENTRE MODELOS

Recommender	Precision	Recall	NDCG	EPC	Gini	Aggregate Diversity
BPRMF	**0.517**	**0.102**	**0.536**	0.707	0.683	0.110
GNN	0.397	0.078	0.414	0.681	0.675	0.027
KNN Item-based	0.125	0.012	0.128	0.902	0.464	0.328
KNN User-based	0.385	0.068	0.408	0.761	0.604	0.123
MF	0.266	0.047	0.273	0.796	0.618	0.133
MLP	0.141	0.016	0.148	0.840	0.637	0.137
Popularity	0.321	0.055	0.336	0.574	0.739	0.014
Random	0.022	0.002	0.022	**0.943**	**0.290**	**0.491**

Tabla 2: Resultados obtenidos por la mejor configuración de cada modelo para el dataset de MovieLens 100k

5.2.2. RESULTADOS DE KNN

Type	k	Similarity	Precision	Recall	NDCG	EPC	Gini	Aggregate Diversity
User-based	**5**	**Cosine**	0.385	0.068	0.408	0.761	0.604	0.123
User-based	10	Cosine	0.355	0.066	0.384	0.762	0.634	0.098
User-based	10	Pearson	0.345	0.063	0.357	0.726	0.674	0.090
User-based	5	Pearson	0.329	0.057	0.344	0.743	0.620	0.116
User-based	1	Cosine	0.307	0.061	0.313	0.780	0.646	0.121
Item-based	**10**	**Pearson**	0.125	0.012	0.128	0.902	0.464	0.328
Item-based	5	Pearson	0.113	0.012	0.118	0.896	0.419	0.365
Item-based	1	Pearson	0.084	0.007	0.082	0.890	0.527	0.280
Item-based	5	Cosine	0.080	0.007	0.076	0.945	0.408	0.410
Item-based	10	Cosine	0.073	0.005	0.071	0.963	0.528	0.321

Tabla 3: Evaluación de los top-5 modelos de KNN User-based y Item-based para el dataset MovieLens 100k

5.2.3. RESULTADOS DE MATRIX FACTORIZATION

Nº factors	Reg	Lr	Epochs	Batch Size	Precision	Recall	NDCG	EPC	Gini	Aggregate Diversity
264	0.1	0.01	50	4096	0.266	0.047	0.273	0.796	0.618	0.133
264	0.1	0.005	50	4096	0.249	0.044	0.266	0.789	0.623	0.130
264	0.01	0.01	50	4096	0.239	0.037	0.249	0.803	0.579	0.159
264	0.01	0.005	50	4096	0.238	0.042	0.248	0.802	0.587	0.159
128	0.1	0.005	50	4096	0.233	0.036	0.240	0.808	0.574	0.164

Tabla 4: Evaluación de los top-5 modelos MF en precisión para el dataset MovieLens 100k

5.2.4. RESULTADOS DE BPRMF

Nº factors	Reg	Lr	Epochs	Batch size	Precision	Recall	NDCG	EPC	Gini	Aggregate Diversity
16	0.1	0.005	50	4096	0.517	0.102	0.536	0.707	0.683	0.110
16	0.1	0.01	50	4096	0.516	0.102	0.539	0.734	0.652	0.134
16	0.01	0.005	50	4096	0.498	0.096	0.523	0.723	0.668	0.129
64	0.1	0.005	50	4096	0.485	0.095	0.499	0.750	0.623	0.164
128	0.1	0.005	50	4096	0.481	0.103	0.501	0.758	0.595	0.174

Tabla 5: Evaluación de los top-5 modelos de BPRMF en precisión para el dataset MovieLens 100k

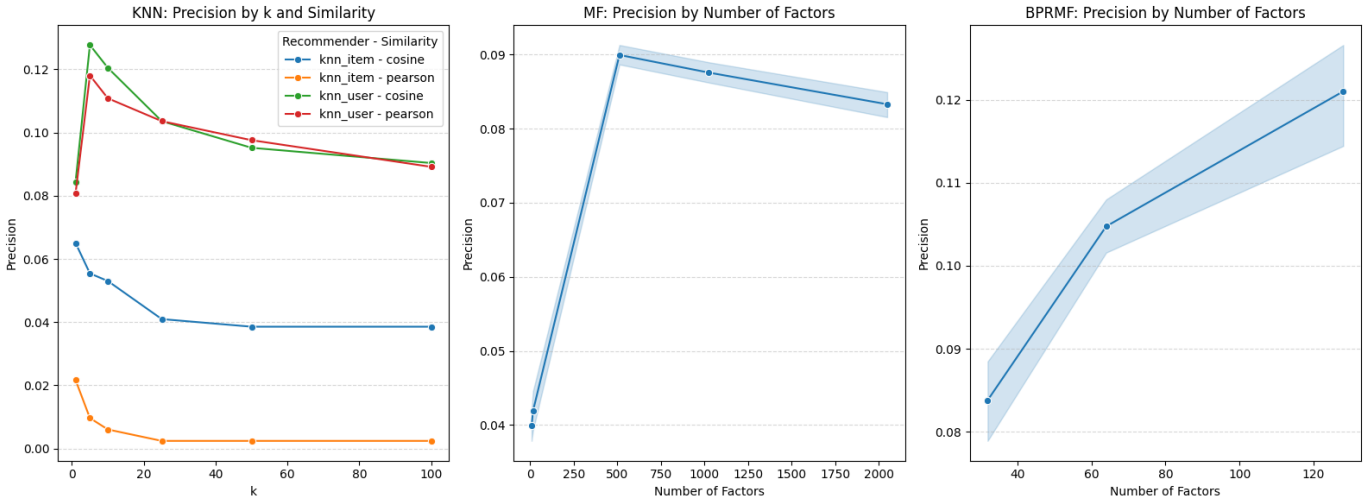

Figura 2: Evolución de la precisión en función del valor de k para KNN y del número de factores latentes para MF y BPRMF para el dataset MovieLens 100k. En los dos últimos, la línea muestra la media y el área azul representa el intervalo de confianza del 95% alrededor de dicha media.

5.2.5. RESULTADOS DE MLP

Nº factors	Reg	Lr	Hidden dims	Epochs	Batch size	Precision	Recall	NDCG	EPC	Gini	Aggregate Diversity
256	0.1	0.01	(512, 256)	200	4096	0.141	0.016	0.148	0.840	0.637	0.137
256	0.01	0.01	(512, 256)	200	2048	0.132	0.015	0.134	0.854	0.607	0.143
256	0.1	0.01	(512, 256)	150	4096	0.129	0.014	0.132	0.853	0.621	0.141
512	0.01	0.01	(256, 128, 64)	150	2048	0.126	0.015	0.131	0.852	0.639	0.142
256	0.01	0.01	(512, 256)	200	4096	0.121	0.013	0.123	0.858	0.616	0.150

Tabla 6: Evaluación de los top-5 modelos de MLP en precisión para el dataset MovieLens 100k

5.2.6. RESULTADOS DE GNN

Nº factors	Reg	Lr	Nº layers	Epochs	Batch Size	Precision	Recall	NDCG	EPC	Gini	Aggregate Diversity
264	0.1	0.005	2	100	2048	0.397	0.078	0.414	0.681	0.675	0.027
128	0.1	0.005	2	100	2048	0.370	0.071	0.381	0.732	0.650	0.034
128	0.1	0.005	3	100	2048	0.356	0.070	0.374	0.677	0.644	0.020
264	0.01	0.005	3	100	2048	0.344	0.070	0.351	0.597	0.666	0.013
256	0.1	0.005	3	50	2048	0.344	0.070	0.365	0.655	0.637	0.015

Tabla 7: Evaluación de los top-5 modelos de GNN en precisión para el dataset MovieLens 100k

5.3. *DATASET FOURSQUARE NEW YORK*

5.3.1. COMPARATIVA GENERAL ENTRE MODELOS

Recommender	Precision	Recall	NDCG	EPC	Gini	Aggregate Diversity
BPRMF	**0.135**	**0.162**	**0.229**	0.975	0.519	0.393
GNN	0.115	0.024	0.114	0.570	0.506	0.003
KNN Item-based	0.065	0.065	0.100	0.994	0.349	0.622
KNN User-based	0.128	0.160	0.206	0.907	0.544	0.407
MF	0.114	0.157	0.152	0.956	0.261	0.723
MLP	0.005	0.003	0.005	**1.000**	0.680	0.127
Popularity	0.049	0.066	0.074	0.774	**0.000**	0.006
Random	0.000	0.000	0.000	0.997	0.043	**0.955**

Tabla 8: Resultados obtenidos por la mejor configuración de cada modelo para el dataset Foursquare New York

5.3.2. RESULTADOS DE KNN

Recommender	k	Similarity	Precision	Recall	NDCG	EPC	Gini	Aggregate Diversity
User-based	5	Cosine	0.128	0.160	0.206	0.907	0.544	0.407
User-based	10	Cosine	0.120	0.151	0.189	0.894	0.602	0.341
User-based	5	Pearson	0.118	0.147	0.191	0.910	0.597	0.346
User-based	10	Pearson	0.111	0.142	0.183	0.898	0.640	0.298
User-based	1	Cosine	0.084	0.121	0.119	0.921	0.523	0.410
Item-based	1	Cosine	0.065	0.065	0.100	0.994	0.349	0.622
Item-based	5	Cosine	0.055	0.053	0.081	0.999	0.282	0.694
Item-based	10	Cosine	0.053	0.053	0.077	0.996	0.189	0.783
Item-based	1	Pearson	0.022	0.022	0.030	0.990	0.332	0.629
Item-based	5	Pearson	0.010	0.011	0.011	0.998	0.252	0.714

Tabla 9: Evaluación de los top-5 modelos de KNN User-based y Item-based para el dataset Foursquare New York

5.3.3. RESULTADOS DE MF

Nº factors	Reg	Lr	Epochs	Batch size	Precision	Recall	NDCG	EPC	Gini	Aggregate Diversity
512	0.15	0.005	250	1024	0.114	0.157	0.152	0.956	0.261	0.723
1024	0.1	0.0075	150	2048	0.113	0.155	0.155	0.949	0.310	0.667
1024	0.15	0.0075	250	2048	0.111	0.152	0.161	0.948	0.309	0.669
512	0.2	0.005	250	2048	0.111	0.154	0.139	0.962	0.242	0.741
2048	0.15	0.005	200	2048	0.111	0.146	0.145	0.958	0.256	0.726

Tabla 10: Evaluación de los top-5 modelos de MF en precisión para el dataset Foursquare New York

5.3.4. RESULTADOS BPRMF

Nº factors	Reg	Lr	Epochs	Batch Size	Precision	Recall	NDCG	EPC	Gini	Aggregate Diversity
128	0.1	0.005	50	2048	0.135	0.162	0.229	0.975	0.519	0.393
128	0.1	0.005	100	2048	0.133	0.164	0.223	0.975	0.585	0.341
128	0.1	0.005	50	4096	0.133	0.160	0.227	0.972	0.505	0.389
128	0.01	0.005	100	4096	0.131	0.162	0.223	0.986	0.485	0.400
128	0.1	0.005	100	4096	0.131	0.158	0.228	0.980	0.531	0.375

Tabla 11: Evaluación de los top-5 modelos de BPRMF en precisión para el dataset Foursquare New York

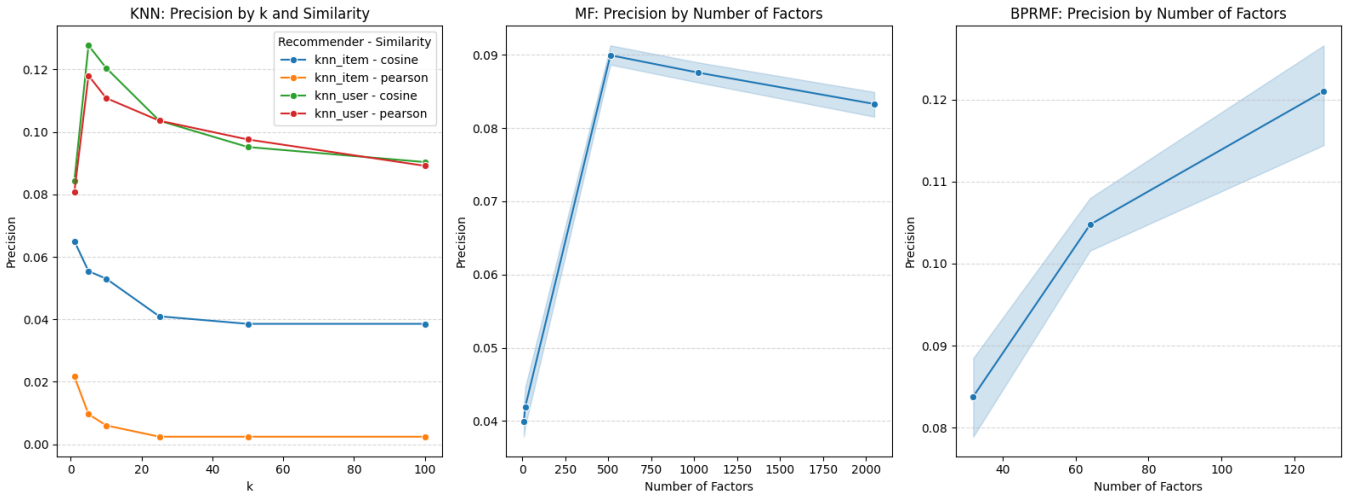

Figura 3: Evolución de la precisión en función del valor de k para KNN y del número de factores latentes para MF y BPRMF para el dataset Foursquare New York. En los dos últimos, la línea muestra la media y el área azul representa el intervalo de confianza del 95% alrededor de dicha media.

5.3.5. RESULTADOS DE MLP

Nº factors	Reg	Lr	Hidden dims	Epochs	Batch size	Precision	Recall	NDCG	EPC	Gini	Aggregate Diversity
512	0.1	0.005	(256, 128, 64)	250	2048	0.005	0.003	0.005	0.999	0.680	0.127
512	0.1	0.010	(256, 128)	100	2048	0.005	0.002	0.004	1.000	0.666	0.098
256	0.1	0.010	(256, 128)	250	2048	0.004	0.008	0.008	1.000	0.755	0.049
256	0.1	0.005	(256, 128)	250	2048	0.004	0.007	0.006	1.000	0.750	0.041
512	0.1	0.010	(256, 128)	250	2048	0.004	0.002	0.003	1.000	0.745	0.058

Tabla 12: Evaluación de los top-5 modelos de MLP en precisión para el dataset Foursquare New York

5.3.6. RESULTADOS DE GNN

Nº factors	Reg	Lr	Nº layers	Epochs	Batch size	Precision	Recall	NDCG	EPC	Gini	Aggregate Diversity
502	0.01	0.100	3	100	2048	0.115	0.024	0.114	0.570	0.506	0.003
502	0.10	0.005	3	50	4096	0.111	0.024	0.112	0.620	0.523	0.005
256	0.01	0.100	3	50	2048	0.110	0.022	0.112	0.583	0.610	0.006
502	0.01	0.100	3	50	2048	0.110	0.023	0.110	0.593	0.674	0.008
502	0.10	0.005	3	50	2048	0.108	0.023	0.111	0.598	0.380	0.003

Tabla 13: Evaluación de los top-5 modelos de GNN en precisión para el dataset Foursquare New York

5.4. DISCUSIÓN DE RESULTADOS

Los experimentos realizados muestran una comparación profunda entre distintos enfoques de recomendación en dos dominios de datos muy distintos: MovieLens 100K, con valoraciones explícitas, y Foursquare New York, con check-ins implícitos. Los resultados muestran cómo la elección del modelo debe adaptarse no solo al tipo de datos, sino también a los objetivos del sistema de recomendación: precisión vs. exploración del catálogo.

En términos generales, BPRMF se posiciona como el modelo más robusto y generalista en ambos conjuntos. En MovieLens (Tabla 2 y Tabla 5), alcanza la mayor precisión (0.517), recall (0.102) y NDCG (0.536), manteniéndose competitivo en novedad (EPC = 0.707) y diversidad (AD = 0.110). En Foursquare (Tabla 8

y Tabla 11), mantiene esta superioridad con una precisión de 0.135 y un EPC aún más elevado (0.975), lo que demuestra su adaptabilidad también en datos implícitos. Este rendimiento se explica por su función de pérdida basada en el aprendizaje de ranking, que optimiza directamente el orden relativo entre ítems, resultando especialmente eficaz en tareas de recomendación top-n. Además, logra un buen equilibrio con respecto a novedad y diversidad, aunque no alcanza los niveles más altos en estas métricas.

Los modelos KNN user-based ofrecen un excelente rendimiento, especialmente en MovieLens donde obtienen 0.385 en precisión (Tabla 2 y Tabla 3). La comparación directa con su contraparte item-based muestra una brecha clara, visible tanto en las tablas como en la evolución mostrada en la Figura 2. Esta diferencia se debe en gran parte a la estructura de los datos, ya que en ambos datasets hay muchos más ítems que usuarios, lo que hace que las similitudes entre ítems sean menos informativas y más ruidosas, especialmente en Foursquare, mientras que las relaciones entre usuarios son más densas y fiables. La configuración óptima (k=5, similitud coseno) ofrece un rendimiento notable con baja complejidad y sin necesidad de entrenamiento en ambos datasets.

Los modelos item-based, aunque significativamente inferiores en precisión, destacan en métricas de novedad y diversidad, lo que los convierte en buenas opciones para escenarios donde el descubrimiento de nuevos ítems es prioritario. Por ejemplo, alcanzan EPC cercanos a 0.99 y alta cobertura de catálogo, aunque a costa de menor relevancia.

Los modelos de factorización de matrices (MF) muestran una notable sensibilidad a los hiperparámetros. En MovieLens (Tabla 4), su rendimiento mejora al aumentar la dimensionalidad (f = 264) y aplicar una regularización fuerte, alcanzando una buena combinación de precisión y novedad (EPC = 0.796). En el dataset Foursquare (Tabla 8), el modelo mejora su rendimiento relativo respecto a otros algoritmos, probablemente debido a la mayor dispersión de los datos y a la capacidad de los embeddings para capturar relaciones implícitas latentes en entornos con menor densidad.

El modelo MLP ha mostrado un rendimiento claramente insuficiente en términos de relevancia en ambos datasets. En MovieLens 100K, su mejor configuración (Tabla 6) alcanza una precisión de apenas 0.141, un recall de 0.016 y un NDCG de 0.148, quedando muy por detrás de modelos como BPRMF (0.517, 0.102, 0.536 respectivamente; Tabla 2) o KNN user-based (0.385, 0.068, 0.408). En el dataset Foursquare New York (Tabla 8), su rendimiento es aún más bajo, con una precisión de 0.005, siendo superado incluso por el modelo de popularidad (0.049). Estos resultados reflejan que, en ausencia de información de contenido o estructuras más complejas, el modelo MLP no es capaz de capturar patrones útiles de preferencia.

El modelo GNN, aunque más competitivo que el MLP, tampoco logra superar al modelo BPRMF en precisión en ninguno de los datasets. En MovieLens 100K (Tabla 2), el GNN alcanza una precisión de 0.397, inferior a los 0.517 de BPRMF, y también obtiene valores menores en recall (0.078 vs. 0.102) y NDCG (0.414 vs. 0.536). En Foursquare (Tabla 8), aunque mejora en precisión respecto a KNN user-based (0.115 vs. 0.128), se mantiene por debajo de BPRMF (0.135) y también muestra peores resultados en métricas de diversidad y novedad como EPC (0.570 vs. 0.975), Gini (0.506 vs. 0.519) o Aggregate Diversity (0.003 vs. 0.393; Tabla 8 y Tabla 13). Esto sugiere que, sin contar con información estructural más rica del grafo o atributos adicionales, el GNN no explota completamente su potencial.

Finalmente, los modelos simples como Popularity y Random actúan como líneas base claras. El primero resulta sorprendentemente competitivo en precisión, pero extremadamente limitado en diversidad (Gini cercano a 1), mientras que el segundo, como es esperable, ofrece máxima cobertura y máxima novedad y diversidad con mínima relevancia.

En resumen, BPRMF se posiciona como el mejor modelo generalista, KNN user-based destaca por su sencillez y eficacia, y los modelos item-based y MF son especialmente útiles cuando se busca fomentar la diversidad o la novedad en las recomendaciones. Estos hallazgos refuerzan la idea de que no existe un modelo universalmente óptimo: la elección debe ajustarse a los objetivos del sistema y a la estructura de los datos disponibles.

6. CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo se ha desarrollado una librería en Python modular y extensible para la implementación, evaluación y comparación de sistemas de recomendación. La herramienta permite integrar distintos modelos de manera homogénea, facilitando la experimentación con diferentes algoritmos, estrategias de recomendación y métricas de evaluación.

Se han evaluado modelos clásicos y modernos sobre dos datasets reales y heterogéneos (MovieLens 100K y Foursquare New York), observando su comportamiento en métricas de relevancia (precisión, recall, NDCG), novedad (EPC) y diversidad (Gini, Aggregate Diversity). Los resultados muestran que no siempre los modelos más complejos, como redes neuronales, ofrecen mejor rendimiento, y que enfoques como BPRMF o KNN user-based logran una excelente relación entre precisión y simplicidad. La librería resultante ha demostrado ser una base eficaz para análisis rigurosos y comparables entre algoritmos.

No obstante, este trabajo abre múltiples líneas de desarrollo futuro. En primer lugar, sería interesante ampliar la librería con modelos más avanzados, como sistemas context-aware o híbridos [4], que combinen filtrado colaborativo con información de contenido o conocimiento experto. También se podrían incorporar nuevos conjuntos de datos, especialmente en dominios distintos como música, noticias o comercio electrónico, para estudiar la generalización de los resultados.

Otra dirección relevante es mejorar la robustez de los experimentos mediante validación cruzada y la evaluación del impacto de distintas estrategias de división de los datos, lo que permitiría analizar la estabilidad de los modelos ante variaciones en los conjuntos de entrenamiento y test. Finalmente, un avance prometedor sería la incorporación de herramientas de optimización automática de hiperparámetros, como la búsqueda bayesiana, que permitirían mejorar la eficiencia experimental y afinar el rendimiento de cada modelo de forma sistemática.

7. BIBLIOGRAFÍA

[1] F. M. Harper and J. A. Konstan, “The MovieLens Datasets: History and Context,” *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, Dec. 2015, doi: 10.1145/2827872.

[2] D. Yang, D. Zhang, L. Chen, and B. Qu, “NationTelescope: Monitoring and Visualizing Large-Scale Collective Behavior in LBSNs,” *Journal of Network and Computer Applications (JNCA)*, vol. 55, 2015.

[3] D. Yang, D. Zhang, and B. Qu, “Participatory Cultural Mapping Based on Collective Behavior Data in Location-Based Social Networks,” *ACM Trans. Intell. Syst. Technol.*, vol. 7, no. 3, Jan. 2016, doi: 10.1145/2814575.

[4] F. Ricci, L. Rokach, and B. Shapira, "Recommender Systems: Techniques, Applications, and Challenges," in *Recommender Systems Handbook*, Third Edition., F. Ricci, L. Rokach, and B. Shapira, Eds., Springer US, 2022, pp. 1–35. doi: 10.1007/978-1-0716-2197-4_1.

[5] I. Wigmore, "Information Overload," TechTarget. Accessed: Jun. 08, 2025. [Online]. Available: https://www.techtarget.com/whatis/definition/information-overload

[6] P. Sánchez and A. Bellogín, "Time-Aware Novelty Metrics for Recommender Systems," in *Advances in Information Retrieval*, G. Pasi, B. Piwowarski, L. Azzopardi, and A. Hanbury, Eds., Cham: Springer International Publishing, 2018, pp. 357–370.

[7] A. N. Nikolakopoulos, X. Ning, C. Desrosiers, and G. Karypis, "Trust Your Neighbors: A Comprehensive Survey of Neighborhood-Based Methods for Recommender Systems," in *Recommender Systems Handbook*, Third Edition., F. Ricci, L. Rokach, and B. Shapira, Eds., Springer, 2022, pp. 39–89. doi: 10.1007/978-1-0716-2197-4_2.

[8] Y. Koren, S. Rendle, and R. Bell, "Advances in Collaborative Filtering," in *Recommender Systems Handbook*, Third Edition., F. Ricci, L. Rokach, and B. Shapira, Eds., Springer, 2022, pp. 92–142. doi: 10.1007/978-1-0716-2197-4_3.

[9] A. Gunawardana, G. Shani, and S. Yogev, "Evaluating Recommender Systems," in *Recommender Systems Handbook*, Third Edition., F. Ricci, L. Rokach, and B. Shapira, Eds., Springer, 2022, pp. 547–601. doi: 10.1007/978-1-0716-2197-4_15.

[10] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: Bayesian Personalized Ranking from Implicit Feedback," Hildesheim, 2012.

[11] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. S. Chua, "Neural collaborative filtering," in *26th International World Wide Web Conference, WWW 2017*, International World Wide Web Conferences Steering Committee, 2017, pp. 173–182. doi: 10.1145/3038912.3052569.

[12] P. Ki, S. A. M. Noah, and H. M. Sarim, "A Survey on Deep Neural Networks in Collaborative Filtering Recommendation Systems," 2024.

[13] X. Wang, X. He, M. Wang, F. Feng, and T. S. Chua, "Neural graph collaborative filtering," in *SIGIR 2019 - Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, Association for Computing Machinery, Inc, Jul. 2019, pp. 165–174. doi: 10.1145/3331184.3331267.

[14] Y. Zhang *et al.*, "BI-GCN: Bilateral Interactive Graph Convolutional Network for Recommendation," in *International Conference on Information and Knowledge Management, Proceedings*, Association for Computing Machinery, Oct. 2023, pp. 4410–4414. doi: 10.1145/3583780.3615232.

[15] S. Vargas and P. Castells, "Rank and relevance in novelty and diversity metrics for recommender systems," in *Proceedings of the Fifth ACM Conference on Recommender Systems*, New York, NY, USA: Association for Computing Machinery, 2011, pp. 109–116. doi: 10.1145/2043932.2043955.

[16] P. Sánchez, A. Bellogín, and L. Boratto, "Bias characterization, assessment, and mitigation in location-based recommender systems," *Data Min Knowl Discov*, vol. 37, no. 5, pp. 1885–1929, 2023, doi: 10.1007/s10618-022-00913-5.

[17] Z. Meng, R. McCreadie, C. MacDonald, and I. Ounis, "Exploring Data Splitting Strategies for the Evaluation of Recommendation Models," in *RecSys 2020 - 14th ACM Conference on Recommender Systems*, Association for Computing Machinery, Inc, Sep. 2020, pp. 681–686. doi: 10.1145/3383313.3418479.

ANEXO I

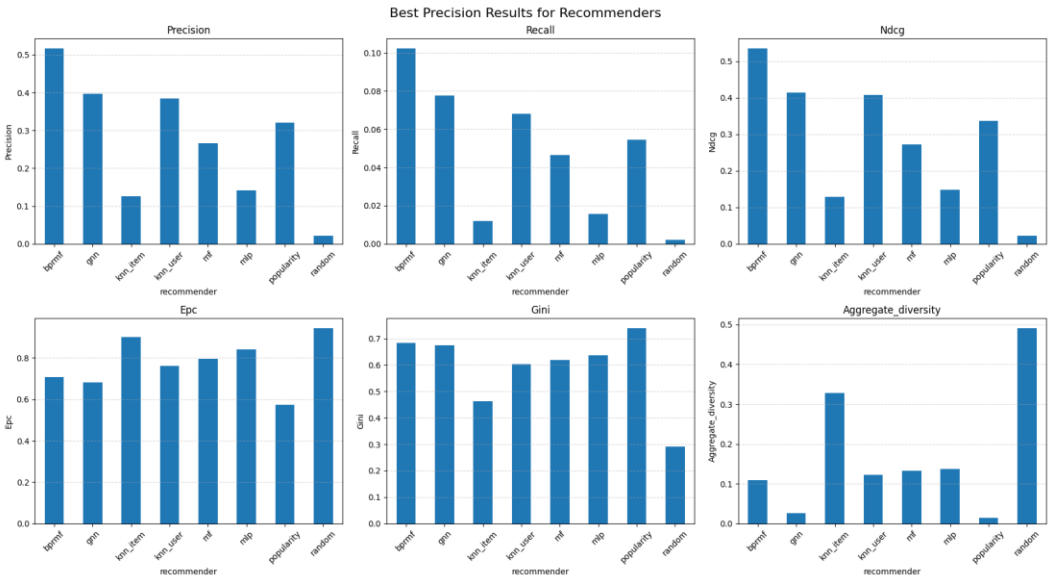

Figura 4: Gráfica mostrando la comparación en las diferentes métricas obtenidas por las mejores configuraciones de cada modelo para el dataset MovieLens 100k

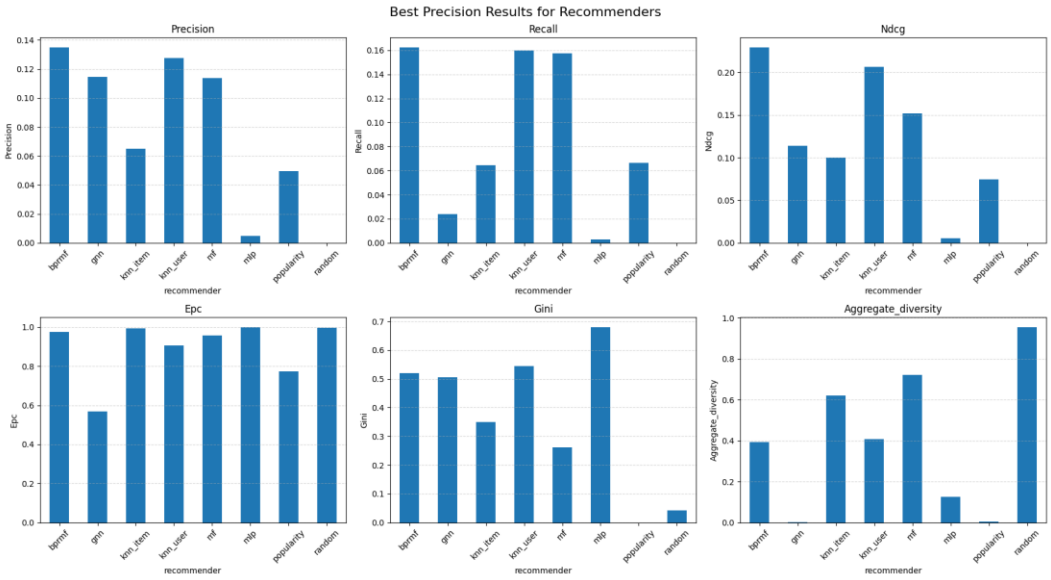

Figura 5: Gráfica mostrando la comparación en las diferentes métricas obtenidas por las mejores configuraciones de cada modelo para el dataset Foursquare New York

La Figura 4 y la Figura 5 muestran, de forma visual, los valores correspondientes a la Tabla 2 y a la Tabla 8 del apartado 5. En concreto, se representan las métricas obtenidas por las mejores configuraciones de cada modelo para ambos conjuntos de datos: MovieLens 100k (Figura 4) y Foursquare New York (Figura 5). Cada gráfico refleja el rendimiento alcanzado por los modelos en términos de precisión, recall, NDCG, EPC, Gini y Aggregate Diversity, facilitando así la comparación directa entre algoritmos en cada una de las métricas evaluadas.

En cuanto a los resultados visualizados, se observa que BPRMF es el modelo más destacado en precisión, recall y NDCG en ambos datasets, lo que confirma su eficacia en términos de relevancia. En cambio, modelos como KNN item-based y Random sobresalen en EPC y Aggregate Diversity, reflejando una mayor capacidad para recomendar ítems menos populares o más diversos, aunque sacrifican notablemente la relevancia. El modelo MLP muestra un rendimiento bajo en todas las métricas de relevancia, especialmente en Foursquare, mientras que GNN mantiene un equilibrio aceptable en MovieLens, pero tiene un comportamiento menos competitivo en Foursquare, particularmente en diversidad, donde su Aggregate Diversity es prácticamente nula. También destaca que el modelo Popularity, aunque ofrece una precisión razonable, concentra sus recomendaciones en pocos ítems, como indica su diversidad agregada cercana a 0.