UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

MÁSTER UNIVERSITARIO EN ANÁLISIS DE DATOS MASIVOS

TRABAJO FIN DE MÁSTER

COMPARATIVA DE ALGORITMOS DE SISTEMAS DE RECOMENDACIÓN: UN ANÁLISIS BASADO EN RETOS COMUNES.

CARLOS ÁLVAREZ CABALLERO

Dirigido por

Sergio Iglesias Pérez

CURSO 2024 - 2025

TÍTULO: COMPARATIVA DE ALGORITMOS DE SISTEMAS DE RECOMENDACIÓN: UN ANÁLISIS BASADO EN RETOS COMUNES.

AUTOR: CARLOS ÁLVAREZ CABALLERO

TITULACIÓN: MÁSTER UNIVERSITARIO EN ANÁLISIS DE DATOS MASIVOS

DIRECTOR DEL PROYECTO: Sergio Iglesias Pérez

FECHA: [Septiembre] de 2025

RESUMEN

Este Trabajo Fin de Máster aborda el análisis comparativo de diferentes algoritmos de sistemas de recomendación, con el objetivo de evaluar su eficacia frente a retos comunes como *Next Best Item (NBI)* y *Cold Start (CS)*. Para ello, se han utilizado datasets abiertos representativos de cada problema: *Amazon Electronics* para el análisis secuencial relacionado con el problema de *NBI* y *Movielens* para escenarios de objetos nuevos. Se han implementado y evaluado distintos algoritmos, incluyendo enfoques clásicos como *SVD*, modelos secuenciales como *LSTM* y modelos híbridos como *LightFM*.

Los resultados muestran que los modelos secuenciales *(LSTM)* superan a *SVD* en la predicción de interacciones futuras, lo que confirma su idoneidad para capturar dinámicas contextuales en las sesiones de usuario. Sin embargo, *SVD* logró una mayor cobertura de catálogo, haciéndolo útil en contextos donde la exploración es la prioridad. Por su parte, *LightFM* se mostró especialmente eficaz frente al problema de *Cold Start*, al incorporar metadatos que le permiten recomendar ítems sin historial previo de interacciones.

En conclusión, el estudio evidencia que no existe un algoritmo universalmente mejor para todos los retos, sino que la selección depende del contexto, los datos disponibles y los objetivos de la aplicación. Este trabajo aporta una visión práctica y crítica de las ventajas y limitaciones de cada enfoque, sirviendo como guía para la elección de algoritmos en escenarios reales.

Palabras clave: sistemas de recomendación, Next Best Item, Cold Start, SVD, LSTM, LightFM.

ABSTRACT

This Master's Thesis addresses the comparative analysis of different recommendation system algorithms, with the aim of evaluating their effectiveness in the face of common challenges such as Next Best Item (NBI) and Cold Start (CS). To this end, open datasets representative of each problem have been used: Amazon Electronics for sequential analysis related to the NBI problem and Movielens for new object scenarios. Various algorithms have been implemented and evaluated, including classical approaches such as SVD, sequential models such as LSTM, and hybrid models such as LightFM.

The results show that sequential models (LSTM) outperform SVD in predicting future interactions, confirming their suitability for capturing contextual dynamics in user sessions. However, SVD achieved greater catalog coverage, making it useful in contexts where exploration is the priority. For its part, LightFM proved particularly effective in addressing the Cold Start problem by incorporating metadata that allows it to recommend items with no previous interaction history.

In conclusion, the study shows that there is no universally best algorithm for all challenges, but rather that the selection depends on the context, the available data, and the objectives of the application. This work provides a practical and critical view of the advantages and limitations of each approach, serving as a guide for choosing algorithms in real-world scenarios.

Keywords: recommender systems, Next Best Item, Cold Start, SVD, LSTM, LightFM.

Índice general

1. RESUMEN DEL PROYECTO **9**
1.1. Contexto y justificación . . . 9
1.2. Planteamiento del problema . . . 9
1.3. Objetivos del proyecto . . . 9
1.4. Resultados obtenidos . . . 9
1.5. Estructura de la memoria . . . 10

2. OBJETIVOS **11**
2.1. Objetivos generales . . . 11
2.2. Objetivos específicos . . . 11
2.3. Beneficios del proyecto . . . 11

3. ANTECEDENTES / ESTADO DEL ARTE **12**
3.1. Estado del arte . . . 12
3.1.1. Tipos de sistemas de recomendación . . . 12
3.1.2. Retos comunes en los sistemas de recomendación . . . 12
3.1.3. Principales algoritmos . . . 13
3.1.4. Métricas de evaluación . . . 16
3.2. Contexto y justificación . . . 17
3.3. Planteamiento del problema . . . 18

4. DESARROLLO DEL PROYECTO **19**
4.1. Planificación del proyecto . . . 19
4.2. Metodología . . . 20
4.3. Herramientas a utilizar . . . 20
4.3.1. Entorno de desarrollo y ejecución . . . 20
4.3.2. Lenguajes de programación . . . 20
4.3.3. Librerías . . . 20

5. EVALUACIÓN EXPERIMENTAL **22**
5.1. Reto 1: Next Best Item (NBI) . . . 22
5.1.1. Preprocesamiento del Dataset - Amazon Electronics . . . 22
5.1.2. Modelado . . . 25
5.1.3. Evaluación de modelos . . . 29
5.2. Reto 2: Cold Start (CS) . . . 38
5.2.1. Dataset - Movielens Dataset . . . 38
5.2.2. Modelado . . . 41
5.2.3. Evaluación de modelos . . . 42

6. DISCUSIÓN **47**

7. CONCLUSIONES **48**

7.1. Conclusiones del trabajo 48
7.2. Conclusiones personales 48

8. FUTURAS LÍNEAS DE TRABAJO **49**

Bibliografía **50**

Índice de Figuras

4.1. Cronograma del Trabajo de Fin de Máster. 19
4.2. Diagrama de flujo de la metodología. 20

5.1. Formato de los registros de Amazon Products Review: Electronics. 22
5.2. Filtrado de usuarios con menos de 5 valoraciones en Amazon Products Review: Electronics. 23
5.3. Modificación de formato a la variable *timestamp* en Amazon Products Review: Electronics. 24
5.4. Aplicación del filtro al dataset de Amazon Products Review: Electronics. 24
5.5. Separación de los conjuntos de entrenamiento y prueba. 25
5.6. Entrenamiento del modelo SVD. 25
5.7. Función para obtener el *top-k* de cada usuario. 26
5.8. Resultado de las *top-k* predicciones de un usuario. 26
5.9. Preprocesamiento de datos para el modelo LSTM. 27
5.10. Entrenamiento del modelo LSTM. 28
5.11. Generación del top-k productos para cada usuario del modelo LSTM. 29
5.12. Función que calcula *precision@k* y *recall@k* para el modelo *SVD*. 29
5.13. Función que calcula *NDCG@k* para el modelo *SVD*. 30
5.14. Función que calcula *Coverage@k* para el modelo *SVD*. 30
5.15. Resultado métricas SVD. 31
5.16. Función que calcula *Precision@k* y *Recall@k* para el modelo *LSTM*. 32
5.17. Función que calcula *coverage@k* para el modelo *LSTM*. 32
5.18. Función que calcula *NDCG@k* para el modelo *LSTM*. 33
5.19. Resultado métricas LSTM. 33
5.20. Comparativa de métricas de modelo SVD vs modelo LSTM. 34
5.21. Comparativa de métricas normalizadas de modelo SVD vs modelo LSTM. 35
5.22. Diagrama representativo del proceso de Backtesting. 36
5.23. Tasa de acierto del proceso de backtesting. 37
5.24. Distribución de Ratings de backtesting. 38
5.25. DataFrame df_tags incluido en el Movielens dataset. 39
5.26. DataFrame df_movies incluido en el Movielens dataset. 40
5.27. DataFrame df_ratings incluido en el Movielens dataset. 40
5.28. Preprocesamiento del dataset de Movielens. 41
5.29. Vectorización del contenido. 41
5.30. Creación de la matriz de interacciones. 42
5.31. Entrenamiento del modelo LightFM. 42
5.32. Resultados de evaluación del modelo SVD. 42
5.33. Resultados de evaluación del modelo LightFM. 43
5.34. Resultados de evaluación del modelo SVD introduciendo CS ítems. 44
5.35. Resultados de evaluación del modelo LightFM introduciendo CS ítems. 45

5.36. Comparativa de resultados de SVD vs LightFM: Escenario Mixto (10 % de CS ítems). 46
5.37. Comparativa de resultados de SVD vs LightFM 46

Índice de Tablas

5.1. Resultado Métricas modelo SVD vs LSTM. 34
5.2. Tiempo de ejecución de la Generación del top-k. 35
5.3. Resultado Métricas modelo SVD vs LightFM. 43
5.4. Resultado Métricas modelo SVD con ítems CS. 44
5.5. Resultado Métricas modelo LightFM con ítems CS. 45

Capítulo 1. RESUMEN DEL PROYECTO

1.1 Contexto y justificación

Con la evolución de la inteligencia artificial y el aprendizaje automático, los sistemas de recomendación se han consolidado como una de las aplicaciones más extendidas en plataformas digitales. Dichos sistemas permiten personalizar la experiencia del usuario mediante la sugerencia automatizada de productos, servicios o contenidos, basándose en sus preferencias o en su historial de comportamiento.

En la actualidad, los sistemas de recomendación desempeñan un papel crucial en numerosos sectores gracias a su capacidad para personalizar contenidos y mejorar significativamente la experiencia del usuario. En el ámbito del comercio, permiten sugerir nuevos productos basándose en el historial de navegación o compra del usuario. En la industria del entretenimiento, facilitan el descubrimiento de contenidos relacionados con las preferencias del usuario. También se utilizan en servicios de información, publicidad personalizada, redes sociales y plataformas de empleo, entre otros. Esta amplia usabilidad demuestra su valor como herramienta en entornos digitales actuales. Estos sistemas presentan distintos tipos de problemas que necesitan ser mitigados para no afectar negativamente a la experiencia del usuario.

1.2 Planteamiento del problema

A pesar de su éxito y extensión, estos sistemas no carecen de problemas y deben enfrentarse a varios retos comunes que afectan a la eficacia de estos recomendadores, por lo que es de vital importancia plantear soluciones o paliativos para estos problemas. Problemas como la predicción del siguiente ítem que consumirá el usuario (*Next Best Item*), el problema de arranque sin datos o arranque frío (*Cold Start*), o la dispersión de datos en grandes catálogos. Con la intención de abordar estos problemas, se han desarrollado múltiples algoritmos y enfoques, desde métodos más clásicos como el filtrado colaborativo hasta modelos basados en *Deep Learning* como pueden ser las redes neuronales convolucionales (CNN) para extraer patrones en secuencias.

1.3 Objetivos del proyecto

Este proyecto tiene como objetivo analizar y comparar distintos algoritmos usados en los sistemas de recomendación en relación con su capacidad para abordar estos desafíos anteriormente mencionados. Para diferenciar este trabajo de otros ya realizados, se va a adoptar un enfoque más general basado en el tipo de problema técnico que los sistemas deben resolver. Para ello, se utilizarán conjuntos de datos abiertos para entrenar modelos y así poder evaluar, mediante métricas estándar, el rendimiento de cada algoritmo en escenarios representativos.

1.4 Resultados obtenidos

Durante este trabajo se verá cómo, mediante el uso de diferentes técnicas, se consigue mitigar el impacto negativo de los retos que afrontan los sistemas de recomendación. En la mayoría de casos se obtienen resultados positivos, lo cual podía ser esperado, para los retos afrontados.

1.5 Estructura de la memoria

En esta sección se detallará un pequeño resumen de cada uno de los capítulos de este proyecto.

En el Capítulo 2 titulado “Objetivos”, se detallan el objetivo general y los objetivos específicos. Se establece qué se pretende conseguir con el estudio comparativo de algoritmos y qué beneficios puede aportar, tanto en términos de comprensión de los retos como de utilidad práctica.

En el Estado del arte se detalla la base teórica necesaria. Se describen distintos tipos de sistemas de recomendación (colaborativo, basado en contenido e híbrido), los principales retos comunes (*Next Best Item, Cold Start, Data Sparsity*), los algoritmos más representativos (*SVD, LSTM, GRU4Rec, LightFM, k-NN*) y las métricas empleadas en la evaluación. También se justifica el contexto y la justificación del proyecto, además de plantear el problema a resolver.

En el Capítulo 4 titulado “Desarrollo del proyecto”, se incluye la planificación temporal del TFM mediante un cronograma, la metodología seguida para llevar a cabo los experimentos y las herramientas utilizadas (*Google Colab, Python* y librerías específicas utilizadas). Se detallan las decisiones de uso de las tecnologías seleccionadas.

En el siguiente capítulo (“Evaluación Experimental”), se recopila el núcleo del trabajo. Se abordan por separado los dos retos principales. En el reto de *Next Best Item* se trabaja con el dataset de *Amazon Electronics*, describiendo el preprocesamiento, modelado y evaluación de dos modelos para compararlos entre sí y ver si la solución planteada mejora a un modelo base establecido. En cuanto al problema de *Cold Start*, se utiliza el dataset de *Movielens*, y se implementan el modelo base (*SVD*) y el modelo híbrido (*LightFM*) y se comparan sus resultados en escenarios con distintos porcentajes de ítems nuevos.

En el Capítulo 6 titulado “Discusión”, se interpretan los resultados obtenidos, identificando fortalezas y limitaciones de cada enfoque. También se reflexiona sobre el impacto de las restricciones técnicas encontradas y el alcance de los resultados.

En el siguiente capítulo se detallan las conclusiones principales del trabajo, diferenciando entre conclusiones del estudio y conclusiones personales.

En el Capítulo 8 titulado “Futuras líneas de trabajo”, se plantean posibles ampliaciones de la investigación, como la explotación de datasets completos, la incorporación de arquitecturas más avanzadas o el diseño de modelos híbridos que combinen secuencialidad con metadatos.

Capítulo 2. OBJETIVOS

2.1 Objetivos generales

El objetivo principal de este proyecto consiste en ampliar el conocimiento actual sobre los algoritmos usados en los sistemas de recomendación y analizar cuán eficaces son para resolver diferentes problemas.

2.2 Objetivos específicos

El objetivo general del proyecto se puede dividir en los siguientes objetivos específicos:

- Estudiar diferentes algoritmos usados en la creación de sistemas recomendadores.
- Investigar cómo afrontar los diferentes problemas a los que se enfrentan los sistemas de recomendación frecuentemente.
- Recopilar conjuntos de datos que estén diseñados para presentar cada uno de los problemas anteriormente mencionados para su estudio.
- Entrenar modelos para evaluar, mediante métricas estándar, el rendimiento de cada algoritmo.
- Valorar el impacto de los sistemas de recomendación en la personalización, satisfacción y fidelización del usuario.
- Analizar cómo diferentes plataformas reales implementan sus sistemas de recomendación.
- Comparar las estrategias de recomendación entre distintos sectores.

2.3 Beneficios del proyecto

Entre los beneficios de este proyecto se encuentra medir la eficacia de los sistemas recomendadores en diferentes escenarios, además de ver el impacto que tienen en la personalización, satisfacción y fidelización del usuario.

Capítulo 3. ANTECEDENTES / ESTADO DEL ARTE

3.1 Estado del arte

Gracias a su gran aplicabilidad, los sistemas de recomendación pueden clasificarse en tres grandes categorías: filtrado colaborativo, filtrado basado en contenido y sistemas híbridos, cada una de las cuales resulta útil en escenarios específicos según las características del problema y los datos disponibles.

3.1.1. Tipos de sistemas de recomendación

- **Filtrado colaborativo**: Este enfoque se basa en la similitud entre los usuarios para recomendar elementos. Existen dos variantes principales dentro del filtrado colaborativo: el filtrado colaborativo basado en usuarios, que recomienda elementos a un usuario según las valoraciones de otros usuarios similares, y el filtrado colaborativo basado en elementos, que sugiere elementos similares a los que otro usuario ya ha valorado positivamente [1]. Además, se han desarrollado modelos basados en matrices, que permiten representar las relaciones usuario-elemento (SVD, ALS). Es uno de los más utilizados, pero presenta limitaciones al tener problemas como *cold start* (problema del que se hablará más adelante).

- **Filtrado basado en contenido**: Este método genera recomendaciones a partir de las características del contenido. El sistema anota estas características para crear un perfil basado en el usuario. A partir de este perfil, el sistema será capaz de recomendar contenido con características similares a las que consume el usuario. Es útil si se tiene una buena descripción de los elementos, aunque puede encontrar dificultades para recomendar elementos fuera del historial del usuario (*overspecialization*).

- **Sistemas híbridos**: Para mejorar la precisión y la generación de las recomendaciones es posible combinar las dos técnicas mencionadas anteriormente, al integrar las fortalezas de cada método, estos sistemas pueden mitigar algunas de las limitaciones de las que se han hablado anteriormente.

3.1.2. Retos comunes en los sistemas de recomendación

Uno de los retos de usar sistemas de recomendación es encontrar soluciones a problemas recurrentes en los sistemas. Estos problemas afectan a la funcionalidad de las recomendaciones por lo que es imprescindible plantear soluciones a estos retos. Entre los más relevantes se pueden encontrar los siguientes:

- ***Next Best Item***: el problema se plantea cuando se tienen diferentes posibilidades de recomendaciones y se debe predecir cuál será el siguiente elemento que el usuario consumirá. La falta de información puede ser una de las raíces del problema, ya que

si el sistema no dispone de la suficiente información puede no acertar con su respuesta. Otra dificultad podría ser la ambigüedad en las interacciones del usuario, un mismo usuario puede tener diferentes intereses, lo que complica predecir cuál será el siguiente elemento que consumirá. Se pueden utilizar técnicas más convencionales para paliar este problema como: utilizar filtrado colaborativo, aunque esto puede acarrear otros problemas, o utilizar técnicas *Deep Learning* usando diferentes tipos de redes neuronales (RNN, CNN) para predecir cuál será el siguiente elemento. [2]

- ***Cold Start***: Cuando un usuario se registra en la plataforma, el sistema de recomendación no dispone de información previa asociada a su perfil. Por lo tanto, no es posible determinar con precisión qué elementos recomendar. A este inconveniente se le denomina *Cold Start*. Este problema también puede presentarse en relación con los ítems, es decir, cuando se introduce un nuevo elemento en el sistema y aún no ha sido valorado por ningún usuario, el recomendador tampoco cuenta con información suficiente para gestionarlo adecuadamente. Entre las estrategias comúnmente utilizadas para mitigar este problema se encuentran: la realización de preguntas iniciales para construir un perfil básico del usuario; la solicitud de información explícita sobre sus preferencias, priorizando ítems según sus intereses; o la posibilidad de registrarse mediante redes sociales, lo que permite al sistema acceder a sus interacciones y preferencias previas a partir de sus perfiles sociales.

- ***Data Sparsity***: La escasez de datos es un problema que afecta a los recomendadores cuando el usuario valora solo un pequeño número de los elementos disponibles. Esto genera una matriz de interacción usuario-elemento, en su mayoría vacía, lo que dificulta la identificación de patrones relevantes. Este reto afecta a la precisión y al rendimiento del sistema de recomendaciones. En el filtrado colaborativo, este problema suele ir de la mano con el inconveniente de *Cold Start*, ya que ambos se deben a la falta de datos. Para mitigar los efectos de la escasez de datos se suelen emplear diferentes técnicas, entre ellas algunas relacionadas con el *Deep Learning*, desarrollando redes neuronales que mejoran la predicción en contextos con información limitada. También se suelen explorar métodos que tienen en cuenta las relaciones entre los usuarios, lo que empareja a usuarios similares para recomendarles contenido afín, entre otras estrategias. [3]

3.1.3. Principales algoritmos

Como se ha observado en la sección anterior, los sistemas recomendadores se enfrentan a diversos problemas que deben ser, al menos, mitigados para mejorar la precisión y el rendimiento de las recomendaciones. Para contrarrestar el efecto negativo de estos inconvenientes, pueden emplearse distintas técnicas y algoritmos para cada caso en específico:

- ***GRU4Rec***: Plantea una solución basada en redes neuronales recurrentes para modelar sesiones de usuario como secuencias temporales. La idea central es la siguiente: cada ítem que un usuario consulta en una sesión se transforma en un vector, un embedding, que alimenta a la red. A medida que el usuario interactúa con nuevos elementos, la

GRU, una variante eficiente de las RNN clásicas, actualiza su estado interno para capturar el contexto acumulado hasta ese momento. Este estado oculto actúa como una especie de ”memoria condensada” de la sesión. Una vez procesada la secuencia, el modelo genera una puntuación para cada ítem disponible en el catálogo. Estas puntuaciones permiten ordenar las opciones y seleccionar aquellas que tienen más probabilidades de ser relevantes en el siguiente paso. Es decir, el sistema no solo recomienda lo más visto o lo más popular: su verdadera fortaleza está en anticipar lo que tiene más sentido para ese usuario en ese instante concreto. Además, GRU4Rec no se queda en una aplicación directa de las RNN tradicionales [4]. Introduce adaptaciones específicas para entornos de recomendación, como el uso de funciones de pérdida orientadas al ranking. Un ejemplo destacado es la TOP1 loss, que penaliza aquellas situaciones en las que un ítem irrelevante recibe una puntuación superior a uno verdaderamente relevante. A diferencia de las funciones de pérdida típicas en clasificación, aquí lo que importa no es si acierta o falla una clase, sino el orden de las recomendaciones. Este enfoque encaja especialmente bien con el reto del *Next Best Item*, ya que permite capturar la dinámica de la sesión y generar sugerencias que no solo tienen sentido desde un punto de vista estadístico, sino también desde la coherencia del comportamiento individual del usuario.

- ***LSTM***: La *Long Short-Term Memory (LSTM)* es un tipo de arquitectura de RNN utilizada en *Deep Learning*. Las redes *LSTM* tienen conexiones de retroalimentación, lo que les permite aprovechar las dependencias temporales entre secuencias de datos. Estas redes son capaces de retener información durante secuencias largas. Las celdas LSTM son las que permiten esto y tienen tres componentes principales: una *Input gate*, una *Forget gate* y una *Output gate*. Estas puertas se encargan de ayudar a regular el flujo de información que entra y sale de la celda de memoria. La *Input gate* determina cuánto de la nueva entrada debe almacenarse en la celda de memoria. La *Forget gate* decide qué información descartar de la celda de memoria. La *Output gate* controla qué parte del contenido de la celda de memoria debe utilizarse para calcular el estado oculto. Usando estas tres puertas, las redes LSTM pueden almacenar, actualizar y recuperar información de forma selectiva en secuencias largas [5]. Esto las hace muy valiosas para situaciones donde se requieran recomendaciones contextuales, como puede ser el caso del reto *Next-Best-Item*.

- ***LightFM***: Se trata de un modelo híbrido especialmente diseñado para sacar lo mejor de dos mundos: el filtrado colaborativo y el basado en contenido. Lo que realmente lo hace destacar no es solo esta combinación, sino su capacidad para incorporar metadatos, tanto de los usuarios como de los ítems, directamente en el proceso de factorización [6]. Esta característica le permite funcionar incluso cuando los datos de interacción escasean, algo habitual con usuarios nuevos o productos recién añadidos. A nivel técnico, LightFM representa usuarios e ítems mediante vectores latentes, los conocidos embeddings, y aprende estas representaciones de forma conjunta. ¿El resultado? Un sistema que puede generalizar mucho mejor ante escenarios típicos de *Cold Start*, siempre que se disponga de información auxiliar como género, categoría, edad, o cualquier otro atributo relevante. Y aquí está la clave: no necesita que el usuario haya interactuado

antes para poder ofrecerle una recomendación razonable. En cuanto al entrenamiento, se apoya en funciones de pérdida orientadas al ranking, como BPR (Bayesian Personalized Ranking), aunque también permite utilizar log-loss en contextos de clasificación implícita. Esta flexibilidad lo hace especialmente útil en sistemas reales donde los datos no siempre vienen en forma de calificaciones explícitas.

- ***SVD***: Conocido como *Singular Value Decomposition (SVD)*, este algoritmo descompone la matriz original de interacciones usuario-ítem en tres matrices: $\mathbf{U}$, que representa a los usuarios en un espacio latente; $\mathbf{\Sigma}$, una matriz diagonal con los valores singulares, que indican la importancia relativa de cada dimensión; y $\mathbf{V}^T$, que codifica la representación latente de los ítems. Estos factores latentes actúan como dimensiones ocultas que capturan patrones subyacentes en las preferencias de los usuarios, incluso cuando no son evidentes en los datos observables [7]. Al multiplicar estas tres matrices, se puede reconstruir una versión aproximada de la matriz original, donde los valores faltantes, aquellos ítems que el usuario no ha valorado aún, pueden ser estimados . Este proceso no solo simplifica la representación de los datos, sino que reduce su dimensionalidad, lo cual mejora la eficiencia computacional y facilita la identificación de similitudes entre usuarios e ítems [8]. En escenarios donde la matriz de interacciones está escasamente poblada, como es común en sistemas reales, *SVD* ofrece una solución robusta al problema de *Data Sparsity*.

- ***k-NN***: KNN es un algoritmo de aprendizaje simple, no paramétrico y basado en instancias que puede utilizarse para tareas de clasificación y regresión. En el contexto de los sistemas de recomendación, KNN se utiliza para encontrar los vecinos más cercanos (ya sean usuarios o elementos) basándose en una métrica de similitud [9]. Su funcionamiento se basa en el siguiente razonamiento: si varios usuarios con gustos similares a ti han valorado positivamente ciertos ítems, el sistema pensará en recomendarte ese contenido. Eso es lo que se conoce como enfoque user-based. También puede operar desde el punto de vista de los ítems: si has mostrado interés por ciertos productos o contenidos, el sistema te recomendará otros parecidos. Este sería el enfoque item-based. Desde una perspectiva técnica, *k-NN* calcula una medida de similitud entre usuarios o ítems representados como vectores. Las métricas más comunes para este propósito son la similitud del coseno o la correlación de Pearson. Con esos valores, el algoritmo selecciona los k vecinos más próximos y utiliza esa información para predecir qué podría gustarte a continuación. El parámetro k no es trivial: influye de forma directa en la precisión y en la capacidad del sistema para generalizar. Si bien no está diseñado para resolver problemas complejos como el *Cold Start* o *Next Best Item*, k-NN sigue teniendo un valor claro. Su transparencia, facilidad de implementación y base interpretativa lo convierten en un referente habitual como modelo de referencia. En estudios comparativos, su papel es fundamental: permite establecer un punto de partida y medir de forma concreta cuánto mejoran los resultados al introducir algoritmos más sofisticados.

3.1.4. Métricas de evaluación

Los sistemas de recomendación pueden ser comparados directamente mediante métricas estándar para valorar su rendimiento. Se pueden usar varios tipos de métricas, desde métricas de precisión para valorar en qué medida las recomendaciones coinciden con los intereses reales del usuario, hasta métricas de diversidad para comprobar la variedad de los elementos recomendados. Las métricas que se van a usar para comparar el rendimiento de los modelos serán las siguientes:

- ***Precision@k***: Mide la proporción de ítems relevantes entre las K recomendaciones más importantes [10]. Un ítem se considera relevante cuando el usuario ha mostrado interés por el elemento, ya sea a través de una valoración o de una interacción en el conjunto de prueba. Se centra en la calidad de las recomendaciones y se basa en la siguiente fórmula:

$$\text{Precision@k} = \frac{\text{número de ítems relevantes en las principales } K \text{ recomendaciones}}{K}$$

  Es directa, fácil de interpretar y refleja bien la calidad del ranking. Es clave ante el problema de *Next Best Item* o escenarios en los que se debe hacer un top de recomendaciones. Sin embargo, esta métrica no refleja si el sistema ha omitido otros ítems relevantes fuera del top-k, por lo que se suele complementar con otras métricas para lograr una visión más completa.

- ***Recall@k***: Mide la capacidad del sistema de recomendación para identificar todos los elementos relevantes dentro de las K recomendaciones principales [10]. Un elemento se considera relevante cuando el usuario lo ha consumido o valorado positivamente en el conjunto de evaluación. Se centra en la exhaustividad del sistema y se basa en la siguiente fórmula:

$$\text{Recall@k} = \frac{\text{número de ítems relevantes en las principales } K \text{ recomendaciones}}{\text{número total de ítems relevantes}}$$

  Complementa a la precisión y es útil ante el reto de *Cold Start*, donde cada acierto importa más. No obstante, esta métrica no refleja si el sistema ha omitido otros ítems relevantes dentro del top-k, por lo que puede sobrevalorar sistemas que aciertan tarde.

- ***NDCG@k***: De su nombre en inglés *Normalized Discounted Cumulative Gain at k (NDCG@k)* es una métrica para valorar la calidad de un ranking de elementos. Tiene en cuenta la posición de los ítems relevantes en la lista. Se basa en la idea de que los elementos que ocupan una posición más alta en la clasificación deben recibir más crédito que los que ocupan una posición más baja [11]. Para calcular el valor de esta métrica se usará la siguiente fórmula:

$$\text{NDCG@k} = \frac{\textit{DCG@K}}{\textit{IDCG@K}} = \frac{\sum_{i=1}^{k \text{ (actual order)}} \frac{\text{Gains}}{\log_2(i+1)}}{\sum_{i=1}^{k \text{ (ideal order)}} \frac{\text{Gains}}{\log_2(i+1)}}$$

DCG@k mide la ganancia acumulada ponderada por la posición en la lista, mientras que *IDCG@k* representa la ganancia máxima posible si los ítems relevantes estuvieran perfectamente ordenados, de ahí su nombre, *Ideal Discounted Cumulative Gain*. El rango de valores de *NDCG@k* es de 0 a 1, significando que los valores más altos indican un mejor rendimiento. Es especialmente útil ante el caso de *Next Best Item*, ya que valora no solo si el elemento relevante está presente, sino que también valora su posicionamiento dentro de la lista recomendada.

- ***Diversity@k***: Mide qué tan diferentes son los ítems entre sí dentro de una misma recomendación. Evita que el sistema repita siempre los mismos elementos o recomiende solo lo popular [12]. Se puede medir utilizando la siguiente fórmula:

$$\text{Diversity@}k = \frac{2}{k(k-1)} \sum_{i=1}^{k} \sum_{j=i+1}^{k} (1 - \text{sim}(i,j))$$

  Donde sim(i,j) representa la similitud entre los ítems i y j dentro del conjunto top-k, puede calcularse con la similitud coseno o similitud Jaccard, entre otras. El resultado son valores en el rango de 0 a 1, cuanto más cercanos a 1 más diferentes son los elementos recomendados entre sí, mientras que un valor más cercano a 0 sugiere que las recomendaciones son muy similares. Esta métrica se puede usar frente al problema de *Data Sparsity* ya que fomenta la exploración del catálogo y reduce la frecuencia de la recomendación de los ítems populares.

- ***Coverage@k***: Otra métrica que se usa para medir la variedad en los sistemas de recomendación. En este caso se usará la métrica conocida como *item coverage* que mide si el sistema explora bien todo el catálogo entre distintos usuarios, mientras que *Diversity@k* compara los elementos recomendados a un mismo usuario. Es decir, es el porcentaje de posibles recomendaciones que puede proporcionar el sistema de recomendación [13]. Se calcula de la siguiente forma:

$$\text{Coverage@k} = \frac{\text{número de ítems únicos recomendados}}{\text{número total de ítems}}$$

  Cuanto mayor sea el valor de esta métrica significa que el sistema recomienda más ítems únicos a lo largo de los usuarios. Se podría utilizar para evaluar si el sistema explora el catálogo completo ante el problema de *Data Sparsity*. También sería útil para medir si se está explorando todo el catálogo ante nuevos ítems en un dataset con el problema de *Cold Start* para asegurarse de que los nuevos contenidos tengan más probabilidad de ser recomendados. Sin embargo, esta métrica no evalúa la calidad de las recomendaciones, por lo que se usará de la mano de otras métricas anteriormente propuestas centradas en la precisión o relevancia.

3.2 Contexto y justificación

Los sistemas de recomendación son herramientas usadas ampliamente en los últimos años para personalizar la experiencia del usuario en diferentes ámbitos digitales, por lo que su constante evolución es importante.

Este proyecto parte de la necesidad de analizar cómo responden los recomendadores a los retos más comunes del área, tales como *Cold Start*, *Next Best Item* o *Data Sparsity*. Para ello, se utilizan métricas de evaluación que permiten valorar la eficacia, variedad y cobertura del sistema dentro del catálogo disponible.

Dado que este proyecto se centra en la investigación, su propósito es aportar una visión sobre cómo reaccionan los sistemas de recomendación frente a los problemas mencionados.

3.3 Planteamiento del problema

El uso de sistemas de recomendación en diferentes plataformas es muy común, pero no todas enfrentan los problemas que tienen estas herramientas, o las enfrentan de una manera errónea/incompleta. Estos problemas pueden resultar en pérdida de rendimiento, eficacia o fallos de recomendaciones, entre otros, por parte del sistema, por lo que es importante plantear soluciones correctas y completas.

En la actualidad, existen distintos algoritmos diseñados para mejorar el rendimiento de los recomendadores. Sin embargo, muchos estudios se enfocan en la aplicación o conjunto de datos específicos, por lo que resulta complicado sacar conclusiones generales.

Este proyecto parte de la siguiente cuestión: ¿cómo varía el rendimiento de los algoritmos de recomendación cuando se enfrentan a los principales problemas?

Resolver esta duda permite ofrecer una guía para elegir la solución más adecuada, teniendo en cuenta el contexto en cada caso, además de identificar las fortalezas y debilidades de cada enfoque.

Capítulo 4. DESARROLLO DEL PROYECTO

En este capítulo se detallarán varios aspectos relacionados con el desarrollo del trabajo, incluyendo la planificación, la metodología y las herramientas a utilizar.

4.1 Planificación del proyecto

La planificación se resume en el siguiente cronograma (véase *Figura 4.1*):

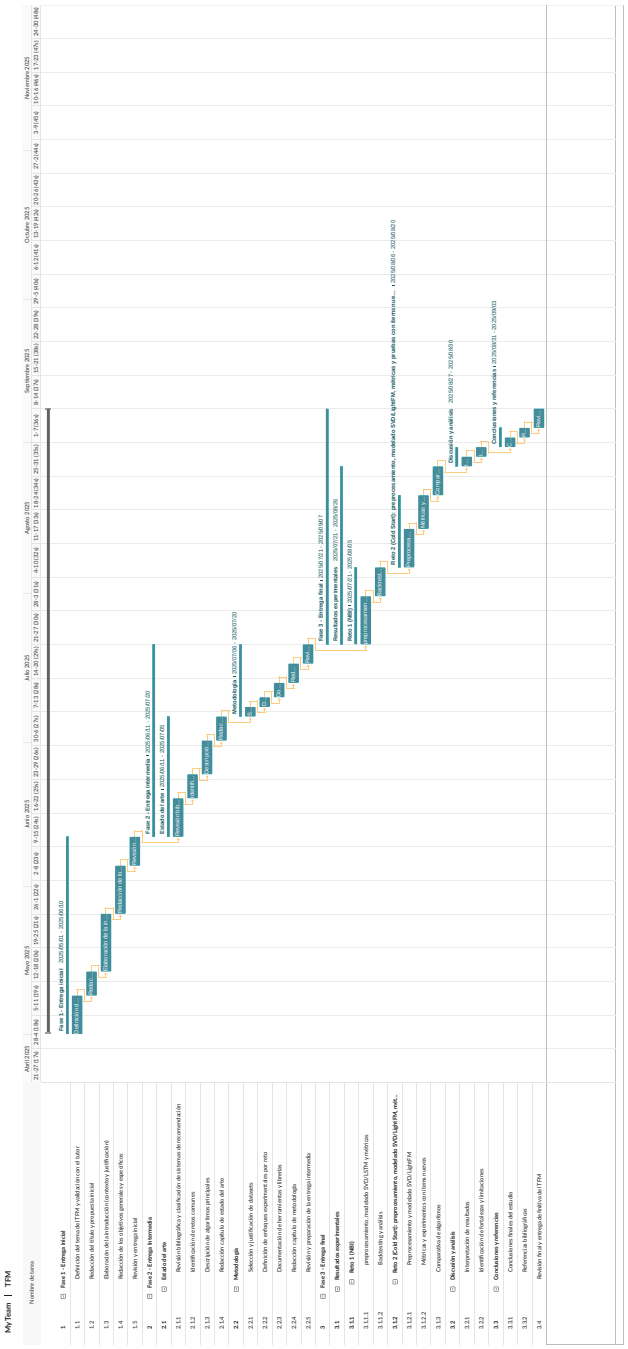

Figura 4.1. Cronograma del Trabajo de Fin de Máster.

4.2 Metodología

La metodología seguida en este trabajo consiste en realizar una comparativa experimental de varios algoritmos de sistemas de recomendación aplicados sobre datasets públicos que presentan cada uno de los retos comunes. Para lograrlo, se han seleccionado varios algoritmos representativos de los principales enfoques existentes (colaborativo, basado en contenido, híbrido y *deep learning*). Los modelos se entrenarán y evaluarán sobre diferentes conjuntos de datos adaptados a los retos analizados (*Next Best Item y Cold Start*). Posteriormente, se aplicarán métricas estándar de recomendación para comparar el rendimiento de cada algoritmo en distintos escenarios. Este proceso se puede resumir en la *Figura 4.2*:

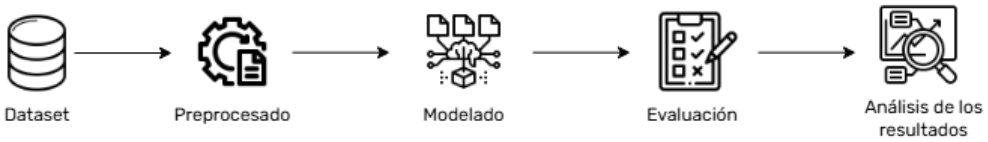

Figura 4.2. Diagrama de flujo de la metodología.

4.3 Herramientas a utilizar

En esta sección, se mencionarán las herramientas utilizadas para el análisis descrito durante el trabajo. Tanto las aplicaciones utilizadas como las librerías y lenguajes de programación.

4.3.1. Entorno de desarrollo y ejecución

Para el entorno de programación, se ha utilizado la plataforma en la nube conocida como ***Google Colab***. Se trata de un entorno que permite utilizar cuadernos *Jupyter Notebook* de forma interactiva sin necesidad de configurar un entorno local. Destaca por ofrecer potentes TPU y GPU, lo que facilita la experimentación con modelos de *machine learning* y *deep learning* [14]. Durante este proyecto se acabó por utilizar esta herramienta como entorno principal de desarrollo y ejecución, debido a que ofrece flexibilidad y accesibilidad, principalmente.

4.3.2. Lenguajes de programación

Como lenguaje de programación se optó por utilizar ***Python***. Es un lenguaje ampliamente utilizado en el ámbito científico y académico debido a su sencilla sintaxis y a la gran cantidad de librerías disponibles para el análisis de datos, *machine learning* y procesamiento estadístico [15]. La elección de ***Python*** para este trabajo se acabó dando debido a su versatilidad, su extenso uso y su capacidad de integrar fácilmente distintas etapas del flujo de trabajo, desde la carga y preprocesamiento de los datos hasta la construcción, entrenamiento y validación de los modelos.

4.3.3. Librerías

Para el desarrollo de este TFM se consideran como imprescindibles algunas librerías utilizadas para diferentes fases del proyecto.

Pandas

Pandas es una de las librerías más utilizadas en ciencia de datos, orientada a la manipulación y análisis de estructuras de datos. Permite trabajar de manera eficiente con grandes volúmenes de datos gracias a diferentes tipos de objetos que simplifican tareas de limpieza, transformación y filtrado, entre otras [16]. En este proyecto se utilizó para procesar los datasets de *Movielens* y *Amazon Electronics*, gestionar los conjuntos de entrenamiento y prueba y preparar los datos antes de entrenar los algoritmos de recomendación.

Scikit-learn

Scikit-learn es una librería fundamental de *machine learning* en *Python*, que ofrece herramientas para modelar, evaluar y validar algoritmos. Aporta diferentes funciones utilizadas durante el proyecto para dividir los datos en conjuntos de entrenamiento y prueba, métricas de evaluación y técnicas de preprocesamiento [17]. En este proyecto se utilizó para gestionar algunos de los modelos.

Surprise

Se trata de una librería especializada en sistemas de recomendación, particularmente en algoritmos basados en factorización matricial y filtrado colaborativo. Proporciona una implementación eficiente del algoritmo *SVD* y facilita la validación mediante evaluaciones automáticas. En este trabajo, fue clave para implementar el modelo baseline, permitiendo compararlo frente a alternativas más avanzadas.

LightFM

Diseñada específicamente para construir sistemas de recomendación híbridos, ***LightFM*** combina información colaborativa e información basada en contenido. Su ventaja principal es que permite integrar características adicionales de ítems y usuarios para mitigar el problema de *Cold Start*. En este proyecto se utilizó como modelo principal ante el problema anteriormente mencionado, comparando su rendimiento frente al algoritmo *SVD*, para demostrar su capacidad para recomendar ítems nuevos a partir de sus atributos.

Capítulo 5. EVALUACIÓN EXPERIMENTAL

Para entrenar cada uno de los modelos se eligieron diferentes conjuntos de datos. Cada uno de ellos presenta uno de los problemas comunes. En este apartado se planteará el dataset elegido en cada caso, cómo se ha procesado y todo lo relacionado con el modelado. Por último, se discutirán los resultados obtenidos.

5.1 Reto 1: Next Best Item (NBI)

La falta de información, o la ambigüedad en las interacciones del usuario, puede producir uno de los retos más frecuentes a los que se enfrentan los sistemas de recomendación. El problema se plantea cuando se tienen diferentes posibles recomendaciones y se debe predecir cuál será el siguiente elemento que el usuario consumirá. Es importante que el sistema recomendador elija bien sus recomendaciones, ya que esto afectará directamente a la experiencia del usuario.

5.1.1. Preprocesamiento del Dataset - Amazon Electronics

El dataset elegido para abordar el problema del *Next Best Item* es el *Amazon Products Review: Electronics* [18]. Se trata de un conjunto de datos de la página web de *Kaggle* que hace referencia a reseñas de productos electrónicos de *Amazon*.

Cada uno de los registros dispone de los siguientes atributos mostrados en la *Figura 5.1*: un id del usuario que va a hacer la reseña (*userId*), un id del producto que va a ser valorado (*productId*), una valoración (*Rating*) y, por último, un timestamp con la fecha de la valoración (*timestamp*).

```
import pandas as pd
import numpy as np

df = pd.read_csv('sample_data/ratings_Electronics.csv', names=['userId', 'productId', 'Rating', 'timestamp'])
print(df)
```

```
                userId   productId  Rating   timestamp
0        AKM1MP6P0OYPR  0132793040     5.0  1365811200
1       A2CX7LUOHB2NDG  0321732944     5.0  1341100800
2       A2NWSAGRHCP8N5  0439886341     1.0  1367193600
3       A2WNBOD3WNDNKT  0439886341     3.0  1374451200
4       A1GI0U4ZRJA8WN  0439886341     1.0  1334707200
...                ...         ...     ...         ...
7824477  A2YZI3C9MOHC0L  BT008UKTMW     5.0  1396569600
7824478  A322MDK0M89RHN  BT008UKTMW     5.0  1313366400
7824479  A1MH90R0ADMIK0  BT008UKTMW     4.0  1404172800
7824480  A10M2KEFPEQDHN  BT008UKTMW     4.0  1297555200
7824481  A2G81TMIOIDEQQ  BT008V9J9U     5.0  1312675200

[7824482 rows x 4 columns]
```

Figura 5.1. Formato de los registros de Amazon Products Review: Electronics.

Este dataset es perfecto para entrenar y evaluar modelos secuenciales como *GRU4Rec*, además este conjunto de datos representa interacciones motivadas por necesidad, interés o hábito del usuario, lo que lo hace ideal para modelar patrones secuenciales y determinar cuál

será el siguiente producto que adquirirá el usuario. También tiene suficientes interacciones, específicamente 7.824.482 valoraciones de usuarios a diferentes productos, lo que garantiza que tiene suficiente volumen para entrenar modelos con sentido. Otra ventaja reside en que hay diversidad en los patrones de consumo. Al haber miles de usuarios y miles de productos, algunos de los usuarios compran de forma recurrente, mientras que otros exploran nuevos productos, esto permite evaluar cómo el sistema responde en distintos contextos.

Durante el preprocesamiento se hicieron una serie de modificaciones al conjunto de datos por distintos motivos:

El primer filtro que se le aplicó fue una reducción. Los registros provenientes de usuarios con menos de 5 valoraciones se descartan, ya que los usuarios con pocas interacciones no ofrecen suficiente información para que un algoritmo aprenda sus preferencias, se considera ruido o señales poco representativas.

```
[4]  # Filtrar usuarios con >= 5 interacciones
     df = df.groupby('userId').filter(lambda x: len(x) >= 5)
     print(df)
```

	userId	productId	Rating	timestamp
6	A3J3BRHTDRFJ2G	0511189877	2.0	1397433600
10	AZYNQZ94U6VDB	0511189877	5.0	1401321600
13	AO94DHGC771SJ	0528881469	5.0	1370131200
14	AMO214LNFCEI4	0528881469	1.0	1290643200
16	A3N7T0DY83Y4IG	0528881469	3.0	1283990400
...	...	...	...	...
7824466	ATK4Q037EM13F	BT008UKTMW	4.0	1356134400
7824469	A2H4BLBXDNSZ2O	BT008UKTMW	4.0	1316563200
7824475	A2R6Q6KJCYSVH7	BT008UKTMW	3.0	1343520000
7824479	A1MH90R0ADMIK0	BT008UKTMW	4.0	1404172800
7824481	A2G81TMIOIDEQQ	BT008V9J9U	5.0	1312675200

[2281128 rows x 4 columns]

Figura 5.2. Filtrado de usuarios con menos de 5 valoraciones en Amazon Products Review: Electronics.

Como se observa en la *Figura 5.2* el número de columnas se ha reducido de 7.824.482 a 2.281.128 valoraciones. Esto demuestra que muchos de los registros eran usuarios que no suelen valorar muchos productos, aunque se haya reducido un 70,85 %. Esto no supone un problema, ya que, como se ha mencionado anteriormente, los registros eliminados se consideran ruido en este caso.

La segunda modificación fue a raíz de la variable *timestamp*, por comodidad para trabajar con formatos de fecha y así poder ordenar los registros teniendo en cuenta el formato temporal, tal y como se hace en la *Figura 5.3*.

```
# Convertir timestamp y ordenar
df['timestamp'] = pd.to_datetime(df['timestamp'], unit='s')
df = df.sort_values(['userId', 'timestamp'])
print(df)
```

```
                         userId   productId  Rating  timestamp
1412220  A000715434M800HLCENK9  B000UYYZ0M     1.0 2014-05-19
1961050  A000715434M800HLCENK9  B001EHAI6Y     5.0 2014-05-19
3181681  A000715434M800HLCENK9  B003AFONFU     3.0 2014-05-19
3316426  A000715434M800HLCENK9  B003ES5ZUU     2.0 2014-05-19
7552846  A000715434M800HLCENK9  B00EODJZ8C     2.0 2014-05-19
...                        ...         ...     ...        ...
7145947          AZZYW4YOE1B6E  B00BP5MB56     3.0 2014-01-15
2493240          AZZYW4YOE1B6E  B0029LB5D4     5.0 2014-04-02
2307829          AZZYW4YOE1B6E  B001TH7GUA     5.0 2014-07-01
7513901          AZZYW4YOE1B6E  B00E952W3A     4.0 2014-07-01
7797388          AZZYW4YOE1B6E  B00IP1MQNK     4.0 2014-07-12

[2281128 rows x 4 columns]
```

Figura 5.3. Modificación de formato a la variable *timestamp* en Amazon Products Review: Electronics.

Durante el modelado, se intentó una estrategia de evaluación top-k exhaustiva. No obstante, debido a las dimensiones del conjunto de datos y las limitaciones de la versión gratuita de *Google Colab*, se acabó optando por una versión simplificada. Donde no se explora el catálogo completo, sino que se usa una versión filtrada del dataset donde solo se muestran los productos con más valoraciones y los usuarios con más actividad (véase *Figura 5.4*).

```
# Reducción de dataset
top_users = df['userId'].value_counts().nlargest(20000).index
top_items = df['productId'].value_counts().nlargest(5000).index
df_reduced = df[df['userId'].isin(top_users) & df['productId'].isin(top_items)]
print(df_reduced)
```

```
                        userId   productId  Rating  timestamp
2186425  A0284208PB0CNSHI1OC6  B00104EPHA     4.0 2013-07-25
4280264  A0284208PB0CNSHI1OC6  B004MU8VCS     5.0 2013-07-31
2770684  A0284208PB0CNSHI1OC6  B002PN8B1A     3.0 2014-01-02
2463801  A0655860XQH1M5Q8JH1M  B0027EMHM6     5.0 2013-02-08
2770897  A0655860XQH1M5Q8JH1M  B002PN8B1A     1.0 2013-03-22
...                       ...         ...     ...        ...
3215796          AZZYW4YOE1B6E  B003BYRGJU     5.0 2014-01-01
3336139          AZZYW4YOE1B6E  B003FMU320     5.0 2014-01-01
6039118          AZZYW4YOE1B6E  B0082YEH8M     4.0 2014-01-15
7145947          AZZYW4YOE1B6E  B00BP5MB56     3.0 2014-01-15
2307829          AZZYW4YOE1B6E  B001TH7GUA     5.0 2014-07-01

[225077 rows x 4 columns]
```

Figura 5.4. Aplicación del filtro al dataset de Amazon Products Review: Electronics.

No es realista usar el dataset completo ya que el tiempo de ejecución marcaba unas 154 horas usando la opción gratuita de CPU de *Google Colab*, según la librería de tqdm de Python. Por lo que se acabó limitando el dataset a los 20.000 usuarios más activos (los que más reseñas han publicado) y a los 5.000 productos más populares, es decir, los que más valoraciones han recibido por parte de los usuarios. Esto supuso pasar de 2.281.128 a 225.077 registros dentro del dataset, y por lo tanto, a un menor tiempo de ejecución. Usar esta medida tiene desventajas, ya que se pierde variedad de catálogo, pero usar el conjunto de datos filtrado

sigue siendo representativo del dataset completo ya que se sigue preservando la estructura y los retos fundamentales del problema (predicción secuencial y dispersión de datos).

5.1.2. Modelado

Modelo *baseline*: SVD

Una vez se ha completado el preprocesamiento, se puede comenzar con la siguiente fase: crear el modelo. Antes de crearlo, se deben separar dos conjuntos de datos, el de entrenamiento (train) y el de prueba (test). Como su propio nombre indica, el conjunto de entrenamiento se usará para entrenar el modelo, mientras que el conjunto de prueba se usará para ponerlo a prueba. Además, los usuarios e ítems deben estar en ambos conjuntos. Esto se debe a que se pretende evaluar el sistema en escenarios donde se conocen los usuarios e ítems para intentar predecir cuál será el siguiente contenido que consuma el usuario.

```
[10] from sklearn.model_selection import train_test_split

     train_df, test_df = train_test_split(df_reduced, test_size=0.2, random_state=42)

     # Asegurarse de que todos los usuarios e ítems del test estén también en train
     test_df = test_df[test_df['userId'].isin(train_df['userId'])]
     test_df = test_df[test_df['productId'].isin(train_df['productId'])]
```

Figura 5.5. Separación de los conjuntos de entrenamiento y prueba.

A continuación, se procederá a entrenar el modelo. Para comenzar, se creará un modelo que servirá como *baseline* para usarlo como punto inicial de comparación. Para ello, se hará uso del algoritmo *SVD* importado de la librería *surprise* [19] diseñada para trabajar con sistemas de recomendación. En la *Figura 5.6* se puede ver el procedimiento: se comienza utilizando un objeto de la clase *Reader* para establecer el rango de valores posibles de la variable *Rating*, en este caso el dataset contiene valoraciones con un rango de valores de 1 a 5. Posteriormente, se crea un objeto de tipo Dataset para que sea compatible con la librería. Por último, faltaría convertir el dataset en un *trainset* de *surprise*, que contiene los IDs internos que usa la librería. Un trainset, de forma resumida, es un objeto que representa el conjunto de entrenamiento de forma optimizada y anonimizada para los algoritmos. Una vez conseguido el *trainset*, ya se podría crear el modelo y entrenar.

```
[11] from surprise import Dataset, Reader, SVD

     # Entrenar modelo clásico (SVD)
     reader = Reader(rating_scale=(1, 5))
     data = Dataset.load_from_df(train_df[['userId', 'productId', 'Rating']], reader)
     trainset = data.build_full_trainset()

     algo_svd = SVD()
     algo_svd.fit(trainset)

     <surprise.prediction_algorithms.matrix_factorization.SVD at 0x7fde045e1f10>
```

Figura 5.6. Entrenamiento del modelo SVD.

Ahora, se debería calcular el top-k de recomendaciones para cada usuario del *trainset* (véase la *Figura 5.7*). Para ello se usará una función que recibe por parámetro el algoritmo, en este caso SVD, el trainset y el número de ítems a recomendar por usuario. El proceso sería el siguiente: se obtienen los IDs originales de los usuarios, y se calculan los IDs internos de todos los objetos. Después, se itera por cada usuario, la librería tqdm simplemente es para mostrar el progreso de la ejecución. En cada iteración del bucle se extraen los ítems que ya ha valorado ese usuario y se calculan los que no ha visto.

Las predicciones se obtendrán del contenido que no ha visto el usuario, por lo que se predice la puntuación esperada para cada elemento no visto por el usuario. Por último, se ordenan las predicciones de mayor a menor según su puntuación.

```
[14] from tqdm import tqdm

     def get_top_k(algo, trainset, k=10):

         users = trainset._raw2inner_id_users.keys()
         all_items = set(trainset.all_items())
         topk_dict = {}

         for uid in tqdm(users, desc="Calculando top-k completo"):
             try:
                 seen_items = set(j for (j, _) in trainset.ur[trainset.to_inner_uid(uid)])
                 unseen_items = all_items - seen_items

                 predictions = [algo.predict(uid, trainset.to_raw_iid(i)) for i in unseen_items]
                 predictions_sorted = sorted(predictions, key=lambda x: x.est, reverse=True)
                 topk_dict[uid] = predictions_sorted[:k]

             except ValueError:
                 continue

         return topk_dict

     topk = get_top_k(algo=algo_svd, trainset=trainset, k=10)

Calculando top-k completo: 100%|██████████| 19868/19868 [17:19<00:00, 19.11it/s]
```

Figura 5.7. Función para obtener el *top-k* de cada usuario.

El diccionario que se devuelve como resultado tiene la estructura mostrada en la *Figura 5.8*. Las partes más interesantes son las siguientes: dispone de un id que representa el destinatario de la predicción *(user)*, el objeto a recomendar *(ítem)* y la puntuación de la predicción calculada anteriormente *(est)*. Ese diccionario representa los 10 productos que, según el recomendador, el usuario debería comprar.

```
[24] for user_id, predictions in topk.items():
         for prediction in predictions:
             print(prediction)

user: A19D3N662QS9WD item: B000N7VPRW r_ui = None   est = 5.00   {'was_impossible': False}
user: A19D3N662QS9WD item: B000PCX780 r_ui = None   est = 5.00   {'was_impossible': False}
user: A19D3N662QS9WD item: B001MSU1HG r_ui = None   est = 5.00   {'was_impossible': False}
user: A19D3N662QS9WD item: B00894YP00 r_ui = None   est = 5.00   {'was_impossible': False}
user: A19D3N662QS9WD item: B004SCSV2U r_ui = None   est = 5.00   {'was_impossible': False}
user: A19D3N662QS9WD item: B002JP92K8 r_ui = None   est = 5.00   {'was_impossible': False}
user: A19D3N662QS9WD item: B00IX2VGFA r_ui = None   est = 5.00   {'was_impossible': False}
user: A19D3N662QS9WD item: B003DZJQQI r_ui = None   est = 5.00   {'was_impossible': False}
user: A19D3N662QS9WD item: B000I1X3W8 r_ui = None   est = 5.00   {'was_impossible': False}
user: A19D3N662QS9WD item: B004EBUXHQ r_ui = None   est = 5.00   {'was_impossible': False}
```

Figura 5.8. Resultado de las *top-k* predicciones de un usuario.

Modelo Secuencial: LSTM

Para este modelo se usará la librería *spotlight* que permite crear modelos secuenciales de forma fácil y rápida. Anteriormente se intentó hacer uso de diferentes librerías para implementar un modelo *GRU4Rec*, pero las incompatibilidades y errores hicieron que optase por otro algoritmo, en este caso, *LSTM (Long Short-Term Memory)*.

El primer paso será realizar el preprocesamiento de los datos (véase *Figura 5.9*), en este caso se está usando el dataset ya modificado para el modelo *SVD*, por lo que solo habrá que hacer pequeñas modificaciones sobre esto. El formato de la fecha habrá que convertirlo al formato soportado por *spotlight*. Por lo que habrá que convertirlo nuevamente a UNIX.

Acto seguido, se hará un mapeo mediante la factorización de los usuarios y productos, ya que *spotlight* trabaja mediante los *ids*, y estos deben ser consecutivos y enteros. Haciendo esta factorización se consigue que en la variable `user_ids` se contenga el índice numérico de la lista, mientras que en la variable `user_index` se guardará el mapeo inverso, es decir, el id del usuario que tenía antes de factorizarse. Es importante establecer un *padding* en los *ids*, ya que el 0 se reserva como un índice especial.

```
[108] # Convertir la columna timestamp de strings "YYYY-MM-DD" a enteros UNIX
      df['timestamp'] = pd.to_datetime(df['timestamp'], format='%Y-%m-%d')
      # Convertir a tipo int64 (necesario con spotlight)
      timestamps = df['timestamp'].values.astype(np.int64)

      # Factorización
      user_ids, user_index = pd.factorize(df['userId'])
      item_ids, item_index = pd.factorize(df['productId'])

      # Construir objeto Interactions
      interactions = Interactions(
          user_ids   = user_ids.astype(np.int32) + 1,
          item_ids   = item_ids.astype(np.int32) + 1,
          timestamps = timestamps
      )
```

Figura 5.9. Preprocesamiento de datos para el modelo LSTM.

Una vez preprocesados los datos, se podrá comenzar con el entrenamiento del modelo LSTM. Para ello, se hace la división del conjunto de datos en el conjunto de entrenamiento (80 %) y el conjunto de pruebas (20 %), tal y como se muestra en la *Figura 5.10*. Al trabajar con un modelo secuencial, se deben convertir ambos conjuntos a secuencias para que sean formatos válidos para *spotlight*, y a continuación se crea el modelo y se entrena. Los hiperparámetros que se usan se han ido probando para lograr la mejor evaluación posible. La búsqueda de hiperparámetros como *GridSearch* no se puede usar en este caso, ya que los formatos de *spotlight* como *Interactions* no son soportados en la librería de *sklearn*. En el caso de querer hacer una búsqueda de hiperparámetros, se debería implementar a mano, sin hacer uso de ninguna función, lo que conllevaría un gran gasto de tiempo para una menor recompensa.

```
[109] train, test = user_based_train_test_split(interactions, test_percentage=0.2)

    train_seq = train.to_sequence()
    test_seq  = test.to_sequence()

    model = ImplicitSequenceModel(
        representation      = 'lstm',
        loss                = 'bpr',
        embedding_dim       = 128,
        n_iter              = 20,
        batch_size          = 512,
        learning_rate       = 1e-3,
        use_cuda            = True,
        num_negative_samples= 50
    )
    model.fit(train_seq, verbose=True)

    Epoch 0: loss 0.41814474670254453
    Epoch 1: loss 0.32370902202567275
    Epoch 2: loss 0.3111999545778547
    Epoch 3: loss 0.30576665121681834
    Epoch 4: loss 0.3026231624642197
    Epoch 5: loss 0.29968687891960144
    Epoch 6: loss 0.2986528338218222
    Epoch 7: loss 0.295743002575271
    Epoch 8: loss 0.2954117710493049
    Epoch 9: loss 0.29460571310958084
    Epoch 10: loss 0.29359244570440174
    Epoch 11: loss 0.29291150947006384
    Epoch 12: loss 0.2924882581039351
    Epoch 13: loss 0.2901941318901218
    Epoch 14: loss 0.28981778998764196
    Epoch 15: loss 0.2907107296038647
    Epoch 16: loss 0.28967614684786114
    Epoch 17: loss 0.2884020227558759
    Epoch 18: loss 0.2880769968032837
    Epoch 19: loss 0.2882379998966139
```

Figura 5.10. Entrenamiento del modelo LSTM.

Como se ha mencionado anteriormente, la librería *spotlight* trabaja con secuencias, por lo que no se puede reutilizar la función creada anteriormente para el modelo svd, ya que usa objetos de tipo *trainset* de la librería *surprise*.

Habrá que implementar una nueva función para este caso (véase *Figura 5.11*). La función recibirá por parámetros el modelo *LSTM*, el conjunto de test con las secuencias de interacción por usuario. Lo primero que se hace es obtener tanto los `ids` de los usuarios como las listas de las secuencias de los productos vistos por cada usuario. Para cada usuario se itera sobre su secuencia de ítems y se genera un array de scores que predice qué ítems vendrían a continuación según la secuencia. Para que no se sugieran ítems ya vistos, se les da un peso muy bajo. Para finalizar, se seleccionan los índices de los k ítems con mejores puntuaciones y se ordenan.

La generación del top-k conllevó muchísimo menos tiempo de ejecución debido a que el modelo *SVD* debe recorrer todo el catálogo, ítem por ítem. Mientras que, el modelo *LSTM* asume que las secuencias del usuario resumen su historial de comportamiento y utiliza vectores para aumentar su velocidad.

```python
from collections import defaultdict
from tqdm import tqdm

def get_top_k_lstm(model, sequence_interactions, k=10, exclude_seen=True):

    user_ids = sequence_interactions.user_ids
    sequences = sequence_interactions.sequences
    topk = defaultdict(list)

    for user_id, sequence in tqdm(zip(user_ids, sequences), total=len(user_ids), desc="Generando top-k"):
        scores = model.predict(sequence)

        if exclude_seen:
            scores[sequence] = -np.inf

        top_items = np.argpartition(scores, -k)[-k:]
        top_items = top_items[np.argsort(-scores[top_items])]
        topk[user_id] = top_items.tolist()

    return topk

topk_lstm = get_top_k_lstm(model, test_seq, k=10)
```

```
Generando top-k: 100%|██████████| 6411/6411 [00:07<00:00, 908.48it/s]
```

Figura 5.11. Generación del top-k productos para cada usuario del modelo LSTM.

5.1.3. Evaluación de modelos

Los modelos, una vez construidos, no siempre son precisos, por lo que se deberían evaluar para demostrar cuán eficientes son. Para ello, se usarán diferentes métricas como pueden ser: ***precision@k***, ***recall@k***, ***NDCG@k*** y ***coverage@k***. Así se podrán comparar directamente los diferentes modelos implementados.

Modelo SVD

Para comenzar, se explicará la función que calcula la *precision@k* y *recall@k*, mostrada en la *Figura 5.12*. La función obtiene por parámetro el top-k `preds_topk`, anteriormente generado, y un dataframe con los ítems vistos por usuario `test_df`. Para comenzar, se agruparán las interacciones reales de cada usuario, generando así un diccionario `test_dict`. Para cada usuario se itera sobre su lista de recomendaciones top-k y se extraen los ítems vistos por ese usuario. No debería pasar, pero si ese usuario no tiene interacciones reales en `test_df`, lo omite. Se extraen en la lista `recommended` las `ids` de las recomendaciones. Por último, se calcula la *precision* y *recall* mediante sus respectivas fórmulas, y se calcula la media de esos valores.

```python
def precision_recall_at_k(preds_topk, test_df, k=10):
    precisions, recalls = [], []
    test_dict = test_df.groupby('userId')['productId'].apply(set).to_dict()
    for uid, preds in preds_topk.items():
        true_items = test_dict.get(uid, set())
        if not true_items:
            continue
        recommended = set([p.iid for p in preds])
        n_rel = len(true_items)
        n_rec_k = len(recommended & true_items)
        precisions.append(n_rec_k / k)
        recalls.append(n_rec_k / n_rel)
    return np.mean(precisions), np.mean(recalls)
```

Figura 5.12. Función que calcula *precision@k* y *recall@k* para el modelo *SVD*.

Para evaluar la métrica *NDCG@k* se implementó la función representada en la *Figura 5.13*. El método recibe por parámetro el mismo top-k que genera el modelo *SVD* (`preds_topk`) y un dataframe con los ítems vistos por usuario `test_df`. Primero se crea un diccionario con los ítems relevantes de cada usuario a partir del `test_df`. Para cada usuario y su lista de predicciones: se obtienen los ítems realmente vistos por el usuario. Se inicializa la ganancia acumulada descontada para ese usuario y se recorren los k ítems recomendados. Si el ítem es relevante (`true_items`) se añade a la variable `dcg` el inverso proporcional al algoritmo de la posición. De esta forma, cuanto antes aparezca, más crédito recibe. Una vez obtenido el *DCG*, se calcula el *Ideal DCG (IDCG)*, que es el *DCG* máximo posible si todos los ítems relevantes estuvieran en las primeras posiciones. Por último, se calcula el *NDCG* dividiendo el *DCG* entre el *IDCG* y se calcula la media de todos los valores de *NDCG* de todos los usuarios.

```python
def ndcg_at_k(preds_topk, test_df, k=10):
    test_dict = test_df.groupby('userId')['productId'].apply(set).to_dict()
    ndcgs = []

    for uid, preds in preds_topk.items():
        true_items = test_dict.get(uid, set())
        if not true_items:
            continue

        dcg = 0.0
        for rank, pred in enumerate(preds[:k], start=1):
            if pred.iid in true_items:
                dcg += 1 / np.log2(rank + 1)

        ideal_hits = min(len(true_items), k)
        idcg = sum(1 / np.log2(i + 1) for i in range(1, ideal_hits + 1))

        ndcg = dcg / idcg if idcg > 0 else 0
        ndcgs.append(ndcg)

    return np.mean(ndcgs)
```

Figura 5.13. Función que calcula *NDCG@k* para el modelo *SVD*.

La última métrica que se usará para evaluar el modelo es *Coverage@k* (véase *Figura 5.14*). La función recibe por parámetro el top-k generado por el modelo *SVD* y el número total de ítems del catálogo. Para cada una de las predicciones de cada usuario se guardan todos los ítems distintos recomendados en el top-k de todos los usuarios. Por último, se calcula el *coverage@k* dividiendo el número de ítems únicos recomendados entre el total de ítems del catálogo. Así se podrá ver el porcentaje de ítems del catálogo que se recomiendan.

```python
def coverage_at_k(preds_topk, total_items, k=10):
    recommended_items = set()

    for preds in preds_topk.values():
        recommended_items.update(p.iid for p in preds[:k])

    coverage = len(recommended_items) / total_items
    return coverage
```

Figura 5.14. Función que calcula *Coverage@k* para el modelo *SVD*.

Haciendo uso de las funciones anteriormente vistas, se obtienen todos los valores de las métricas para poder evaluar el modelo SVD, en este caso los resultados están representados en la *Figura 5.15*.

```
[21] # NDCG@10
     ndcg = ndcg_at_k(topk, test_df, k=10)
     print(f"NDCG@10: {ndcg:.4f}")

     # Coverage@10
     all_items = df_reduced['productId'].nunique()
     coverage = coverage_at_k(topk, all_items, k=10)
     print(f"Coverage@10: {coverage:.4f}")

     # Precision@10 y Recall@10
     p, r = precision_recall_at_k(topk, test_df, k=10)
     print(f'Precision@10: {p:.4f}, Recall@10: {r:.4f}')

     NDCG@10: 0.0027
     Coverage@10: 0.2456
     Precision@10: 0.0013, Recall@10: 0.0044
```

Figura 5.15. Resultado métricas SVD.

Modelo LSTM

Para evaluar el siguiente modelo, hay que hacer pequeños cambios en todas las funciones que calculan cada una de las métricas. Esto se debe al formato de los datos, ya que la librería *Surprise* trabaja con objetos de tipo *Trainset*, mientras que la librería *Spotlight* utiliza *SequenceInteractions*.

En la *Figura 5.16* se detalla la función implementada en *Python* para calcular el *Precision@k* y *Recall@k* para el modelo LSTM. La función recibe por parámetros el top-k generado por el modelo *LSTM* y las secuencias de interacciones en el conjunto de prueba. Se recorre cada usuario para evaluar la calidad de las recomendaciones hechas y se obtiene la lista de ítems recomendados (`predicted_items`) para ese usuario. `Mask` es un array booleano que selecciona las secuencias en `test_sequence` pertenecientes al usuario actual. Mientras que en `user_sequences` se almacena el subconjunto de secuencias del usuario en el conjunto de pruebas. En el caso de que el usuario actual no tenga secuencias en el conjunto de prueba, se salta al siguiente usuario. A continuación, se toma el último ítem `(seq[-1])` de cada secuencia del usuario. Después, se recorre cada ítem real (`true_item`) y se comprueba si aparece en la lista de ítems recomendados (hit = True). Por último, se calcula el sumatorio de ambas variables haciendo uso de sus respectivas fórmulas y se devuelve la media de ambas variables.

```python
def precision_recall_at_k_lstm(topk_dict, test_sequence, k=10):

    precisions = []
    recalls = []

    for user_id in topk_dict:
        predicted_items = topk_dict[user_id]

        mask = (test_sequence.user_ids == user_id)
        user_sequences = test_sequence.sequences[mask]

        if len(user_sequences) == 0:
            continue

        true_items = [seq[-1] for seq in user_sequences if len(seq) > 0]

        for true_item in true_items:
            hit = true_item in predicted_items

            precisions.append(1.0 / k if hit else 0.0)
            recalls.append(1.0 if hit else 0.0)

    return {
        f"precision@{k}": np.mean(precisions),
        f"recall@{k}": np.mean(recalls)
    }
```

Figura 5.16. Función que calcula *Precision@k* y *Recall@k* para el modelo *LSTM*.

Para calcular el *Coverage@k* se implementó la función representada en la *Figura 5.17*. Dicha función recibe por parámetro las recomendaciones top-k de cada usuario y el conjunto de todos los ítems disponibles en el dataset. El cálculo conlleva iterar sobre cada uno de los productos recomendados por el modelo, para cada usuario se seleccionan las k primeras recomendaciones y se añaden a `recommended_items` (sin haber duplicados). El método devuelve la división del número de productos recomendados entre todos los del catálogo.

```python
def coverage_at_k_lstm(preds_topk, all_items, k=10):

    recommended_items = set()
    for items in preds_topk.values():
        recommended_items.update(items[:k])

    return len(recommended_items) / len(all_items)
```

Figura 5.17. Función que calcula *coverage@k* para el modelo *LSTM*.

A continuación se procederá con el procedimiento usado para calcular *NDCG@k* (véase *Figura 5.18*). La función recibe por parámetro lo mismo que la versión del modelo *SVD*, un diccionario con las recomendaciones top-k por usuario y un *DataFrame* de prueba con los ítems que realmente consumió cada usuario. A raíz del *DataFrame* anterior, se agrupan los productos consumidos por cada usuario. Después empieza el recorrido de los usuarios junto a sus predicciones y se obtienen los ítems vistos por el usuario. Acto seguido, se itera sobre el top-k de las recomendaciones. Si la predicción se encuentra entre los productos vistos por

el usuario, se calcula el *DCG*. Para calcular el *IDCG* se obtiene el número mínimo entre k y los ítems reales del usuario (`ideal_hits`), en el caso de que el usuario no haya valorado ningún producto, se omite la evaluación. Para acabar, se calcula el *IDCG* y a partir del *DCG* y *IDCG* se calcula el *NDCG* del usuario.

```python
def ndcg_at_k_lstm(preds_topk, test_df, k=10):
    test_dict = test_df.groupby('userId')['productId'].apply(set).to_dict()
    ndcgs = []

    for uid, preds in preds_topk.items():
        true_items = test_dict.get(uid, set())
        if not true_items:
            continue

        dcg = 0.0
        for rank, pred in enumerate(preds[:k], start=1):
            if pred in true_items:
                dcg += 1 / np.log2(rank + 1)

        ideal_hits = min(len(true_items), k)
        if ideal_hits == 0:
            continue

        idcg = sum(1 / np.log2(i + 1) for i in range(1, ideal_hits + 1))
        ndcg = dcg / idcg
        ndcgs.append(ndcg)

    return np.mean(ndcgs) if ndcgs else 0.0
```

Figura 5.18. Función que calcula *NDCG@k* para el modelo *LSTM*.

Usando todas las funciones explicadas anteriormente, se pueden calcular cada uno de los valores de las métricas para evaluar el modelo *LSTM*, obteniendo los siguientes valores (véase *Figura 5.19*):

```python
# Precision@10 y recall@10
metrics = precision_recall_at_k_lstm(topk_lstm, test_seq, k=10)
for metric, value in metrics.items():
    print(f"{metric}: {value:.4f}")

# NDCG@10
ndcg10 = ndcg_at_k_lstm(topk_lstm, test_df, k=10)
print(f"NDCG@10: {ndcg10:.4f}")

all_items = np.unique(item_ids)

# Coverage@10
coverage = coverage_at_k_lstm(topk_lstm, all_items, k=10)
print(f"Coverage@10: {coverage:.4f}")
```

```
precision@10: 0.0015
recall@10: 0.0151
NDCG@10: 0.0193
Coverage@10: 0.0026
```

Figura 5.19. Resultado métricas LSTM.

Comparativa

Ahora que se han entrenado dos modelos y se han evaluado, se pueden comparar y concluir varias cosas. Los resultados de ambos modelos se resumen en la *Tabla 5.1*:

Algoritmo	Precision@10	Recall@10	NDCG@10	Coverage@10
SVD	0.0013	0.0044	0.0027	0.2456
LSTM	0.0015	0.0151	0.0193	0.0026

Tabla 5.1. Resultado Métricas modelo SVD vs LSTM.

La tabla anterior se puede resumir gráficamente con la siguiente imagen (véase *Figura 5.20*):

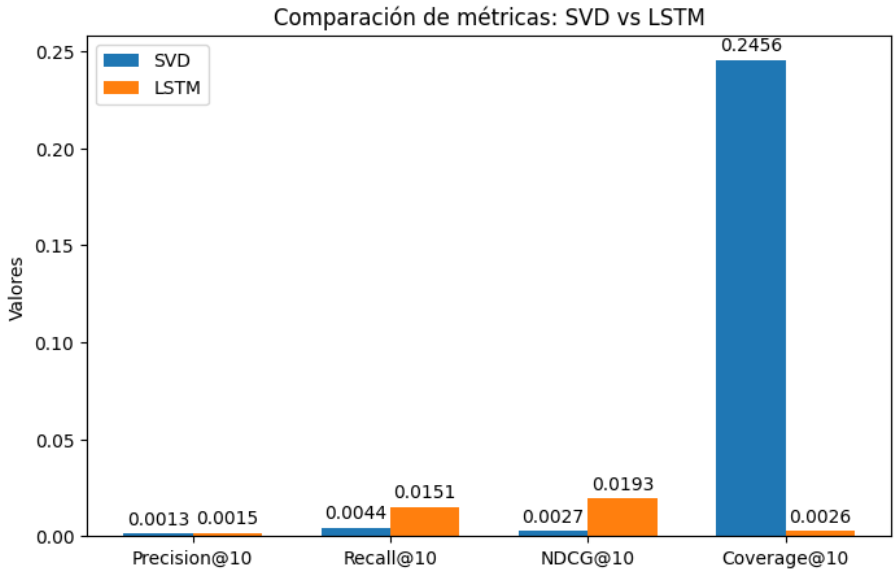

Figura 5.20. Comparativa de métricas de modelo SVD vs modelo LSTM.

- ***Precision@10***: Aunque el modelo *LSTM* tiene un resultado ligeramente superior, ambos algoritmos tienen un resultado similar y muy bajo, lo que indica que las recomendaciones de ambos modelos rara vez aciertan en los intereses del usuario.
- ***Recall@10***: En este caso si se nota una gran diferencia entre los modelos, el modelo *LSTM* obtiene un resultado bastante superior al modelo *SVD*. Esto indica que el algoritmo *LSTM* suele obtener los ítems que realmente interesan al usuario dentro de las diez primeras recomendaciones.
- ***NDCG@10***: El modelo *LSTM* tiene mejores resultados que el modelo *SVD* evaluando la calidad del ranking completo del top-k recomendaciones, donde importa el orden de los productos recomendados.
- ***Coverage@10***: El único caso en el que el modelo *SVD* es mejor que el modelo *LSTM*. Lo que implica que el modelo *SVD* suele recomendar más porcentaje del catálogo completo de productos.

Normalizando las variables, ya que el *Coverage@10* del modelo *SVD* tiene un valor ampliamente mayor que las demás variables. Se obtiene la siguiente gráfica radar representada en la *Figura 5.21*:

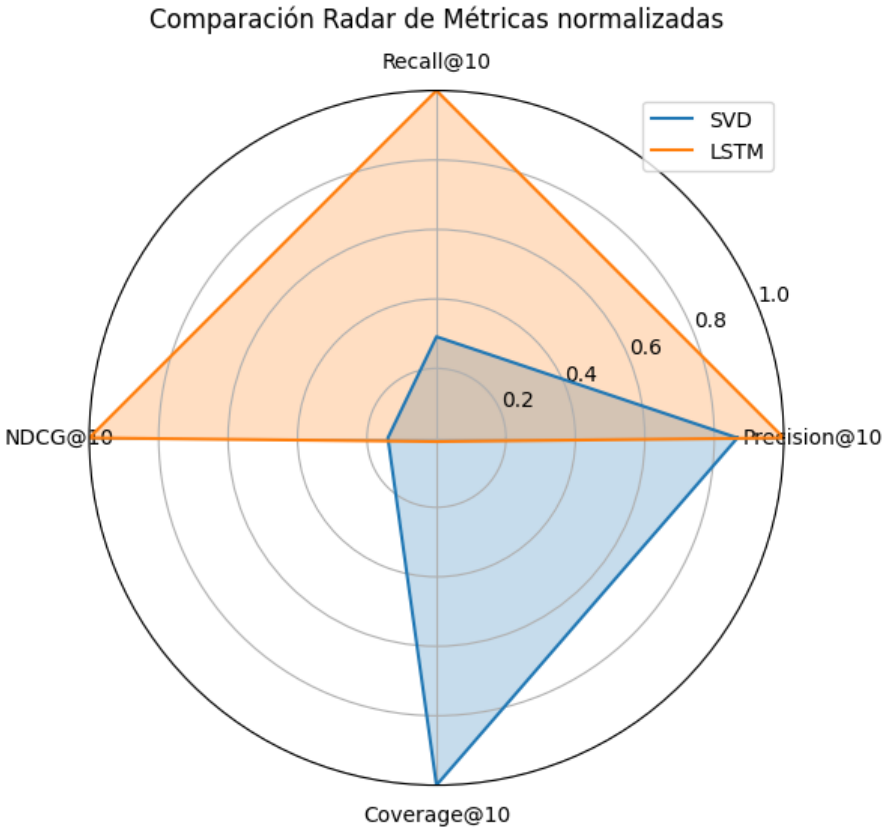

Figura 5.21. Comparativa de métricas normalizadas de modelo SVD vs modelo LSTM.

Para concluir, se podría decir que el modelo *LSTM* es mejor prediciendo los ítems que le interesan al usuario, sus recomendaciones top-k suelen estar en los intereses del usuario y las recomendaciones top-k suelen ser mejores. Mientras que el modelo *SVD* aprovecha mejor el catálogo, ya que suele recomendar más productos diferentes. Esto no necesariamente es un inconveniente para el modelo *LSTM*, ya que lo que importa en este caso es mejorar las recomendaciones contextuales. El modelo *SVD* es un sistema colaborativo clásico, por lo que es eficaz para detectar patrones globales. Sin embargo, el modelo *LSTM* aprovecha la secuencialidad para detectar lo que el usuario consumirá a continuación.

En cuanto a tiempos de ejecución, la generación del top-k de cada modelo conllevó un tiempo distinto en cada caso:

Algoritmo	Tiempo de ejecución (min)	Usuarios (iteraciones)	Iteraciones/s
SVD	17:19	19 868	19.11
LSTM	00:06	6 326	916.00

Tabla 5.2. Tiempo de ejecución de la Generación del top-k.

Como se puede comprobar en la *Tabla 5.2*, el modelo *LSTM* no solo obtiene mejores resultados para este problema que el modelo *SVD*. Además, tiene un tiempo de ejecución para generar el top-k de cada usuario muchísimo menor. Esto se debe a que el modelo *SVD* debe recorrer todo el catálogo, ítem por ítem. Mientras que el modelo *LSTM* asume que las secuen-

cias del usuario resumen su historial de comportamiento y utiliza vectores para aumentar su velocidad. Además, la librería *Spotlight* ignora secuencias que no tengan suficientes interacciones. Por ello, el número de usuarios se reduce tanto de un modelo a otro.

La evaluación de los modelos se podría dar por concluida aquí, pero se realizará un último experimento para comprobar la eficacia del modelo. Para ello, se realizará un proceso de *Backtesting*.

Backtesting

La idea de hacer este proceso de *Backtesting* surge para comprobar la verdadera eficacia del modelo [20]. Para ello, la idea sería dividir el dataset filtrado, usado en el entrenamiento de los modelos, en 2 nuevos datasets, uno donde se encuentren las últimas 3 valoraciones de cada usuario, que se usará para comprobar si el modelo logra predecir correctamente. Y otro donde se encuentren el resto de valoraciones, usado para entrenar el modelo. Con el dataset reducido se entrenaría el modelo y se obtendría el top-k del modelo para cada usuario. Por último, se buscarían las 3 valoraciones eliminadas al principio del proceso en el top-k generado por el modelo. Así se podría ver, sin comparar con ningún otro modelo, si realmente el sistema de recomendación hace buenas o malas predicciones en una comparativa general.

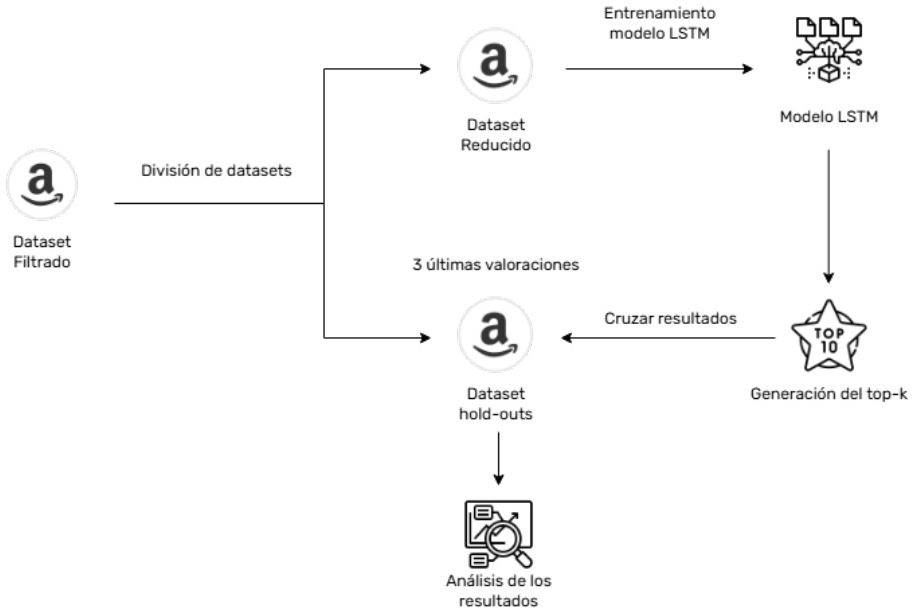

Figura 5.22. Diagrama representativo del proceso de Backtesting.

El resultado que se obtiene es el que viene representado en la *Figura 5.23*. La sumatoria de las predicciones de todos los usuarios es de 32 289 predicciones, de las cuales solo se encuentran 1 321 predicciones en el dataset de *hold-outs*. Obteniendo una tasa de acierto de:

$$\textit{Tasa de acierto} = \frac{\text{Número de predicciones acertadas}}{\text{Número de predicciones totales}} = \frac{1321}{32289} = 4{,}09\,\%$$

Figura 5.23. Tasa de acierto del proceso de backtesting.

Es una tasa bastante baja, de cada 100 predicciones el modelo solo acertaría 4. Se podría decir que es algo normal debido a que el número medio de valoraciones por usuario es de 11 valoraciones, si se usan como *hold-outs* las últimas 3 valoraciones solo se usaría una secuencia de 8 valoraciones para el entrenamiento del modelo. Cuanto más larga es la secuencia, mejor se entrenaría el modelo teóricamente. Además, se limitan los usuarios con menos de 8 valoraciones para que el modelo tenga una secuencia lo suficientemente larga.

Con lo visto en los anteriores resultados se podría concluir que el modelo no consigue predecir muchos de los contenidos que el usuario va a consumir a continuación. Pero, de las recomendaciones que sí acabó consumiendo el usuario, ¿qué valoración le dio al contenido? Observando esto se podría ver si dentro de las recomendaciones que sí consume el usuario, al menos, le gustaron o si el contenido recomendado tampoco era del gusto del usuario. En la *Figura 5.24* se muestra la distribución de cada uno de los valores de los 1321 acertados en el dataset de *hold-outs*.

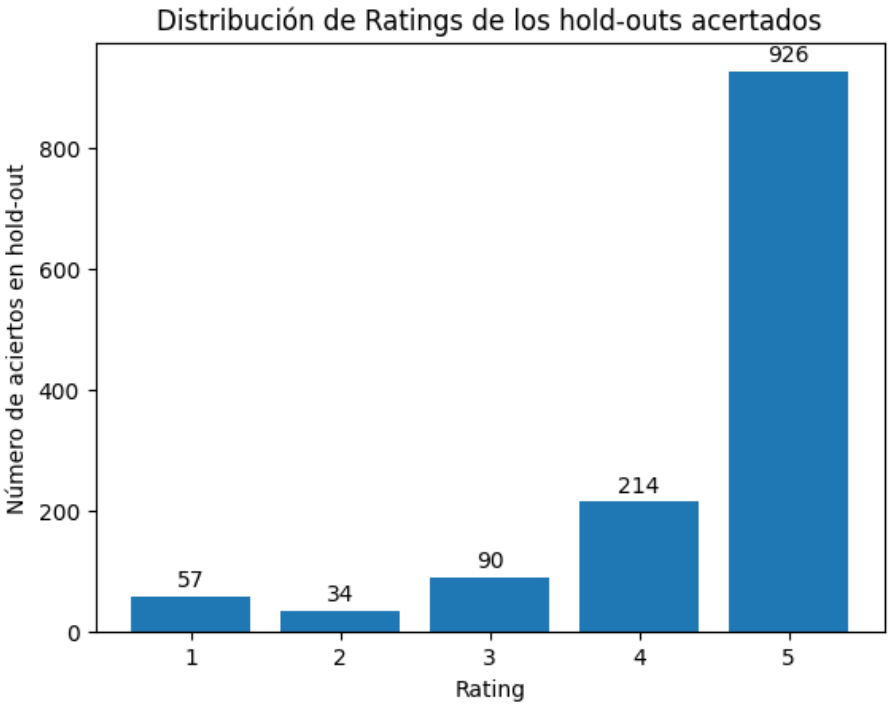

Figura 5.24. Distribución de Ratings de backtesting.

Observando las valoraciones del usuario, la mayoría de los productos recomendados por el modelo sí eran del gusto del usuario. Exactamente el 86 % de las valoraciones tiene al menos 4 estrellas. Al menos, cuando el modelo consigue acertar, se demuestra que suele captar los gustos más claros del usuario.

Si el modelo tuviera más ítems para entrenarse, podría obtener un mejor resultado, en vez de usar el conjunto de datos filtrado durante el preprocesamiento de la *Sección 5.1.1*. El conjunto de datos era demasiado grande y por ello no se utilizó el dataset completo.

5.2 Reto 2: Cold Start (CS)

Cold Start supone una limitación muy recurrente en los sistemas recomendadores que dependen únicamente de interacciones, ya que impide recomendar ítems nuevos o recomendar a usuarios nuevos. Es de vital importancia intentar aminorar este problema o intentar tratarlo, ya que influirá directamente en la fidelidad del usuario nuevo, o en el caso de ser un contenido nuevo, influirá negativamente en el producto.

5.2.1. Dataset - Movielens Dataset

Para abordar el problema de *Cold Start* se usará un dataset que presenta este problema en muchos casos, en el caso de esta fuente de datos el problema será de *Cold Start* de ítems ya que no se tiene información sobre los usuarios. El conjunto de datos contiene información de películas, valoraciones de las reseñas de los usuarios y tags de películas, entre otra información adicional dependiendo del modelo elegido. *GroupLens* ofrece la posibilidad de descargar

diferentes versiones del dataset de *Movielens* dependiendo de la cantidad de datos que se quieran manejar. En este caso se usará el dataset *ml-latest-small* que incluye datos actuales. Se hace uso de su versión *small* debido a que su versión normal incluye demasiados datos, lo que afecta directamente a los tiempos de ejecución. Estos datasets no incluyen directamente ítems nuevos (ítems *Cold Start*), por lo que se incluirán posteriormente al experimentar.

En el caso de este dataset no hay mucho preprocesamiento necesario, por lo que se mostrará la estructura de los diferentes *DataFrames* que contiene esta fuente de datos. A pesar de contener 4 *DataFrames*, se usarán 3, ya que el cuarto es usado para unir con otros datasets externos. En la *Figura 5.25* se puede observar la estructura de *df_tags*:

```
print(df_tags)
```

	userId	movieId	tag	timestamp
0	2	60756	funny	1445714994
1	2	60756	Highly quotable	1445714996
2	2	60756	will ferrell	1445714992
3	2	89774	Boxing story	1445715207
4	2	89774	MMA	1445715200
...	...	...	...	...
3678	606	7382	for katie	1171234019
3679	606	7936	austere	1173392334
3680	610	3265	gun fu	1493843984
3681	610	3265	heroic bloodshed	1493843978
3682	610	168248	Heroic Bloodshed	1493844270

[3683 rows x 4 columns]

Figura 5.25. DataFrame df_tags incluido en el Movielens dataset.

En esta estructura se puede observar que se contienen etiquetas añadidas por los usuarios, es decir, información útil para clasificar el contenido. Cada registro tiene un *userId*, un *movieId*, un *tag* y un *timestamp*.

En la siguiente figura se mostrará la estructura del siguiente *DataFrame*, el *df_movies* (véase *Figura 5.26*). La estructura mostrada contiene información sobre las películas, también será útil para clasificar el contenido. Cada registro contiene un *movieId*, un *title* y sus respectivos *genres* asociados a cada película.

Estos dos primeros *DataFrames* sirven principalmente para clasificar el contenido, es decir, poder construir unos *item_features* para el modelo *LightFM*, pero para el modelo baseline *SVD* no se usarán.

```
print(df_movies)

      movieId                                       title  \
0           1                            Toy Story (1995)
1           2                              Jumanji (1995)
2           3                     Grumpier Old Men (1995)
3           4                    Waiting to Exhale (1995)
4           5          Father of the Bride Part II (1995)
...       ...                                         ...
9737   193581  Black Butler: Book of the Atlantic (2017)
9738   193583               No Game No Life: Zero (2017)
9739   193585                                 Flint (2017)
9740   193587         Bungo Stray Dogs: Dead Apple (2018)
9741   193609         Andrew Dice Clay: Dice Rules (1991)

                                        genres
0     Adventure|Animation|Children|Comedy|Fantasy
1                      Adventure|Children|Fantasy
2                                  Comedy|Romance
3                            Comedy|Drama|Romance
4                                          Comedy
...                                           ...
9737              Action|Animation|Comedy|Fantasy
9738                     Animation|Comedy|Fantasy
9739                                       Drama
9740                           Action|Animation
9741                                       Comedy

[9742 rows x 3 columns]
```

Figura 5.26. DataFrame df_movies incluido en el Movielens dataset.

Como el esqueleto del dataset se encuentra el *DataFrame df_ratings*, que contiene las valoraciones de los usuarios para poder hacer el modelado (véase *Figura 5.27*).

```
print(df_ratings)

        userId  movieId  rating   timestamp
0            1        1     4.0   964982703
1            1        3     4.0   964981247
2            1        6     4.0   964982224
3            1       47     5.0   964983815
4            1       50     5.0   964982931
...        ...      ...     ...         ...
100831     610   166534     4.0  1493848402
100832     610   168248     5.0  1493850091
100833     610   168250     5.0  1494273047
100834     610   168252     5.0  1493846352
100835     610   170875     3.0  1493846415

[100836 rows x 4 columns]
```

Figura 5.27. DataFrame df_ratings incluido en el Movielens dataset.

Como se menciona anteriormente, este *DataFrame* contiene la información necesaria para el modelado de cada uno de los sistemas de recomendación. Cada registro contiene un *userId*, un *movieId*, un *rating* y un *timestamp*.

5.2.2. Modelado

En el caso del modelo *SVD* se repite el código utilizado en la Subsección *5.1.2*, tanto para crear y entrenar el modelo como para obtener su top-k ítems.

Para intentar aminorar el problema de *Cold Start* se hará uso de un modelo híbrido *LightFM* que, a través de los metadatos, permite recomendar incluso cuando el usuario no tiene ningún tipo de registros. Para ello, se usará la librería creada explícitamente para poder crear, entrenar e incluso evaluar estos modelos [21].

Se comenzará haciendo un pequeño preprocesamiento del conjunto de datos para poder trabajar cómodamente con *LightFM* (véase *Figura 5.28*). Se comenzará por concatenar todos los tags en una sola cadena separada por espacios. Estos tags serán usados más adelante para crear los *items_features*. Después, se unen la columna de tags y el dataset de películas. Por último, se elimina el carácter de la barra vertical de la columna de *genres*, ya que cada género venía separado por un '|' y se necesita texto limpio para poder vectorizar la matriz de *item_features*, después nos deshacemos de los valores *NaN* de las columnas de *genres* y *tag*.

```
movie_tags = df_tags.groupby('movieId')['tag'].apply(lambda x: ' '.join(x)).reset_index()

df_movies = df_movies.merge(movie_tags, on='movieId', how='left')

df_movies['content'] = df_movies['genres'].fillna('').str.replace('|', ' ') + ' ' + df_movies['tag'].fillna('')
```

Figura 5.28. Preprocesamiento del dataset de Movielens.

El siguiente paso será crear la matriz de pesos que representará los *item_features* para el modelo *LightFM*. Este proceso se representa en la *Figura 5.29* y permite recomendar ítems nuevos basándose en su contenido. Para crear esta matriz, se hace uso de la librería *Sklearn*, específicamente la función de *TfidfVectorizer* y se le asignan los pesos *TF-IDF* [22] del texto de la columna *content*. Es crucial que el orden de filas coincida con el índice interno de los ítems, esto se garantiza haciendo un mapeo con los respectivos IDs.

```
tfidf = TfidfVectorizer(max_features=1000)
item_features = tfidf.fit_transform(df_movies['content'])

movie_id_map = {id_: idx for idx, id_ in enumerate(df_movies['movieId'])}
```

Figura 5.29. Vectorización del contenido.

Por último, faltaría crear la matriz de interacciones. Para comenzar, como feedback ímplicito para la función de pérdida de *LightFM*, se usarán las valoraciones que tengan más de 4 estrellas. Después, se transforman los IDs reales a los índices internos para ser usados en la matriz. Además, el mapeo de ítems debe ser el mismo que el usado antes, al crear los *items_features*. Hecho esto, ya se podrá crear la matriz de interacciones de los usuarios y las películas.

```python
positive_ratings = df_ratings[df_ratings['rating'] >= 4.0].copy()

user_id_map = {id_: idx for idx, id_ in enumerate(positive_ratings['userId'].unique())}
positive_ratings['user_idx'] = positive_ratings['userId'].map(user_id_map)
positive_ratings['item_idx'] = positive_ratings['movieId'].map(movie_id_map)

interactions = csr_matrix((
    [1] * len(positive_ratings),
    (positive_ratings['user_idx'], positive_ratings['item_idx'])
))
```

Figura 5.30. Creación de la matriz de interacciones.

Una vez se tienen correctamente preparados los *items_features* y la matriz de interacciones, se podrá entrenar el modelo (véase *Figura 5.31*).

```python
model = LightFM(loss='warp')
model.fit(interactions, item_features=item_features, epochs=10, num_threads=2)
```

Figura 5.31. Entrenamiento del modelo LightFM.

Para la generación del top-k se hacen pequeños ajustes a la función del modelo *SVD* ya que los pasos son los mismos, solo varían ligeramente los tipos de objetos que se manejan.

Una vez preparados los modelos y generados sus respectivos top-k, se procede con la evaluación de ambos.

5.2.3. Evaluación de modelos

Para evaluar los modelos se observarán las mismas métricas que en la *Subsección 5.1.3*, las funciones utilizadas para el modelo *SVD* se pueden reutilizar, así que se verán directamente los resultados mostrados en la *Figura 5.32*:

```python
test_df = pd.DataFrame(test_df, columns=["userId", "movieId", "rating"])

# Precision y Recall@10
p, r = precision_recall_at_k(topk, test_df, k=10)
print(f"Precision@10: {p:.4f}, Recall@10: {r:.4f}")

# NDCG@10
ndcg = ndcg_at_k(topk, test_df, k=10)
print(f"NDCG@10: {ndcg:.4f}")

# Coverage@10
all_items = df_ratings['movieId'].nunique()
coverage = coverage_at_k(topk, all_items, k=10)
print(f"Coverage@10: {coverage:.4f}")
```

```
Precision@10: 0.0519, Recall@10: 0.0438
NDCG@10: 0.0653
Coverage@10: 0.0342
```

Figura 5.32. Resultados de evaluación del modelo SVD.

Los resultados se compararán con los del modelo *LightFM*, para evaluarlo la propia librería incluye una función para calcular su *precision@10* y su *recall@10*. Además, se han hecho pequeños ajustes a la función NDCG por cuestión de tipos de objetos tratados con este modelo (véase *Figura 5.33*).

```python
from lightfm.evaluation import precision_at_k, recall_at_k

# Precision@k
precision = precision_at_k(model, interactions, item_features=item_features, k=10).mean()
print(f"Precision@10: {precision:.4f}")

# Recall@k
recall = recall_at_k(model, interactions, item_features=item_features, k=10).mean()
print(f"Recall@10: {recall:.4f}")

# Calcular NDCG@10
item_id_map = {movie_id: idx for idx, movie_id in enumerate(df_movies['movieId'])}
ndcg = ndcg_at_k_lightfm(topk_lightfm, test_df, item_id_map, k=10)
print(f"NDCG@10: {ndcg:.4f}")

# Coverage@k
coverage = coverage_at_k(topk_lightfm, all_items, k=10)
print(f"Coverage@10: {coverage:.4f}")
```

```
Precision@10: 0.3811
Recall@10: 0.0810
NDCG@10: 0.0173
Coverage@10: 0.0246
```

Figura 5.33. Resultados de evaluación del modelo LightFM.

Los resultados se resumen en la *Tabla comparativa 5.3*:

Algoritmo	Precision@10	Recall@10	NDCG@10	Coverage@10
SVD	0.0519	0.0438	0.0653	0.0342
LightFM	0.3811	0.0810	0.0173	0.0246

Tabla 5.3. Resultado Métricas modelo SVD vs LightFM.

Analicemos métrica por métrica:

- ***Precision@10:*** el modelo *LightFM* es ampliamente superior en este caso, cabe recordar que esta métrica mide la proporción de ítems realmente relevantes en el top 10.
- ***Recall@10:*** el modelo *LightFM* también logra recuperar una mayor parte de los ítems relevantes disponibles para cada usuario.
- ***NDCG@10:*** en este caso el modelo *SVD* consigue superar al modelo *LightFM*. Se valora la calidad del ranking premiando que los ítems más relevantes aparezcan en las primeras posiciones.
- ***Coverage@10:*** en cuanto a recomendar mayor cantidad de ítems del catálogo también es superior el modelo *SVD*.

A priori podría parecer que el modelo *LightFM* no es excesivamente mejor que el modelo *SVD* ya que no lo logra superar en todas las métricas. Sin embargo, si se observa con detenimiento el significado de cada métrica y se le da contexto, se podría concluir que el modelo *LightFM* es bastante superior para lo que se busca: la calidad del ranking. Este sistema de

recomendación tiene mejor *precision@10* y *recall@10* lo que quiere decir que suele recomendar ítems con alta probabilidad de que le gusten al usuario. Mientras que, el modelo *SVD* muestra mejor capacidad para mostrar más catálogo y muestra una mejor capacidad para ordenar los ítems relevantes en las primeras posiciones.

Ahora, se debería enfocar esta evaluación en el reto al que se enfrentan en este caso: *Cold Start* de ítems, es decir, ¿cómo actúan ambos modelos ante ítems nuevos que no disponen de un historial de valoraciones?

Experimento Cold Start

Este experimento consistirá en introducir en el conjunto de test ítems nuevos, sin historial de valoraciones, para ver cómo reaccionan ambos modelos y poder evaluarlos enfocando el problema de *Cold Start* de ítems. En este caso, el 10 % del total serán ítems nuevos y el otro 90 % serán ítems con historial.

Los resultados obtenidos para el modelo *SVD* son los representados en la *Figura 5.34*:

```
# Mixto
p_m, r_m = precision_recall_at_k(topk, test_mixto, user_col='userId', id_col='movieId', k=k)
ndcg_m   = ndcg_at_k(topk, test_mixto, user_col='userId', id_col='movieId', k=k)
cov_m    = coverage_at_k(topk, total_items=total_items_warm, k=k)

# Solo CS ítems
p_cs, r_cs = precision_recall_at_k(topk, test_cs_only, user_col='userId', id_col='movieId', k=k)
ndcg_cs    = ndcg_at_k(topk, test_cs_only, user_col='userId', id_col='movieId', k=k)

# Sin CS ítems
p_w, r_w = precision_recall_at_k(topk, test_warm_only, user_col='userId', id_col='movieId', k=k)
ndcg_w   = ndcg_at_k(topk, test_warm_only, user_col='userId', id_col='movieId', k=k)

print(f"Mixto: Precision@{k}={p_m:.4f}  Recall@{k}={r_m:.4f}  NDCG@{k}={ndcg_m:.4f}  Cov@{k}={cov_m:.4f}")
print(f"Solo CS: Precision@{k}={p_cs:.4f} Recall@{k}={r_cs:.4f} NDCG@{k}={ndcg_cs:.4f}")
print(f"No CS: Precision@{k}={p_w:.4f}  Recall@{k}={r_w:.4f}  NDCG@{k}={ndcg_w:.4f}")
```

```
Mixto: Precision@10=0.0549  Recall@10=0.0265  NDCG@10=0.0638  Cov@10=0.0385
Solo CS: Precision@10=0.0000 Recall@10=0.0000 NDCG@10=0.0000
No CS: Precision@10=0.0557  Recall@10=0.0423  NDCG@10=0.0697
```

Figura 5.34. Resultados de evaluación del modelo SVD introduciendo CS ítems.

Los anteriores resultados se resumen en la *Tabla comparativa 5.4*:

Algoritmo	Precision@10	Recall@10	NDCG@10	Coverage@10
Mixto	0.0549	0.0265	0.0638	0.0385
Solo CS	0.0000	0.0000	0.0000	0.0385
No CS	0.0557	0.0423	0.0697	0.0385

Tabla 5.4. Resultado Métricas modelo SVD con ítems CS.

Como se puede comprobar, el modelo *SVD* no es capaz de acertar ninguna predicción usando solo ítems nuevos, ya que este modelo se basa en interacciones previas del ítem. Al tener solo un 10 % de *CS ítems* no se ven muchas bajadas en los resultados ya que principalmente el modelo ignora los ítems nuevos, ya que no es capaz de predecir nada, aunque en el *Recall@10* sí, mostrando una bajada representativa por lo comentado anteriormente. En el caso de la *Precision@10* no se muestra un mayor cambio debido a que esta métrica depende más de los pocos aciertos obtenidos que del número de ítems relevantes en el test.

En cuanto al modelo *LightFM*, los respectivos resultados son los siguientes (véase *Figura 5.35*):

```
# Mixto
prec_m = precision_at_k(model, test_global, train_interactions=train_interactions,
                        item_features=item_features_final, k=k).mean()
rec_m = recall_at_k(model,    test_global, train_interactions=train_interactions,
                    item_features=item_features_final, k=k).mean()
ndcg_m = ndcg_at_k_lightfm(topk_global, test_global, k=k)
cov_m = coverage_at_k(topk_global, total_items=n_items, k=k)

# Solo CS ítems

prec_cs = precision_at_k(model, test_cs_only, train_interactions=train_interactions,
                         item_features=item_features_final, k=k).mean()
rec_cs = recall_at_k(model,    test_cs_only, train_interactions=train_interactions,
                     item_features=item_features_final, k=k).mean()
ndcg_cs = ndcg_at_k_lightfm(topk_global, test_cs_only, k=k)

# Sin CS ítems

prec_w = precision_at_k(model, test_warm_only, train_interactions=train_interactions,
                        item_features=item_features_final, k=k).mean()
rec_w = recall_at_k(model,    test_warm_only, train_interactions=train_interactions,
                    item_features=item_features_final, k=k).mean()
ndcg_w = ndcg_at_k_lightfm(topk_global, test_warm_only, k=k)

print(f"Mixto: Precision@{k}={prec_m:.4f}  Recall@{k}={rec_m:.4f}  NDCG@{k}={ndcg_m:.4f}  Cov@{k}={cov_m:.4f}")
print(f"Solo CS: Precision@{k}={prec_cs:.4f} Recall@{k}={rec_cs:.4f} NDCG@{k}={ndcg_cs:.4f}")
print(f"No CS: Precision@{k}={prec_w:.4f}  Recall@{k}={rec_w:.4f} NDCG@{k}={ndcg_w:.4f}")

Mixto: Precision@10=0.1111  Recall@10=0.0710  NDCG@10=0.1356  Cov@10=0.0518
Solo CS: Precision@10=0.0022 Recall@10=0.0012 NDCG@10=0.0019
No CS: Precision@10=0.1109  Recall@10=0.1036 NDCG@10=0.1456
```

Figura 5.35. Resultados de evaluación del modelo LightFM introduciendo CS ítems.

Los anteriores resultados se resumen en la *Tabla comparativa 5.5*:

Algoritmo	Precision@10	Recall@10	NDCG@10	Coverage@10
Mixto	0.1111	0.0710	0.1356	0.0518
Solo CS	0.0022	0.0012	0.0019	0.0518
No CS	0.1109	0.1036	0.1456	0.0518

Tabla 5.5. Resultado Métricas modelo LightFM con ítems CS.

En el caso del modelo *LightFM* sí que se obtiene un valor en el caso de predecir solo ítems nuevos, se trata de un valor muy bajo, pero no es nulo, por lo que el modelo es capaz de predecir nuevos ítems. Comparando los demás valores, el recall baja cuando se usa un test mixto, ya que al añadir ítems el número de ítems relevantes crece, mientras que no se logran recuperar muchos de estos ítems nuevos, pero siguen siendo valores aceptables.

En el caso del escenario donde se usa un test mixto, que es el verdaderamente interesante, ya que es el que añade los ítems nuevos al catálogo, se obtienen los valores representados en la *Figura 5.36*. Como se puede comprobar, el modelo *LightFM* supera en todas las métricas al modelo baseline, ya no solo es mejor ante la situación de introducir ítems nuevos y poder recomendarlos, sino que, a la hora de predecir qué ítem recomendar de forma general, también lo es.

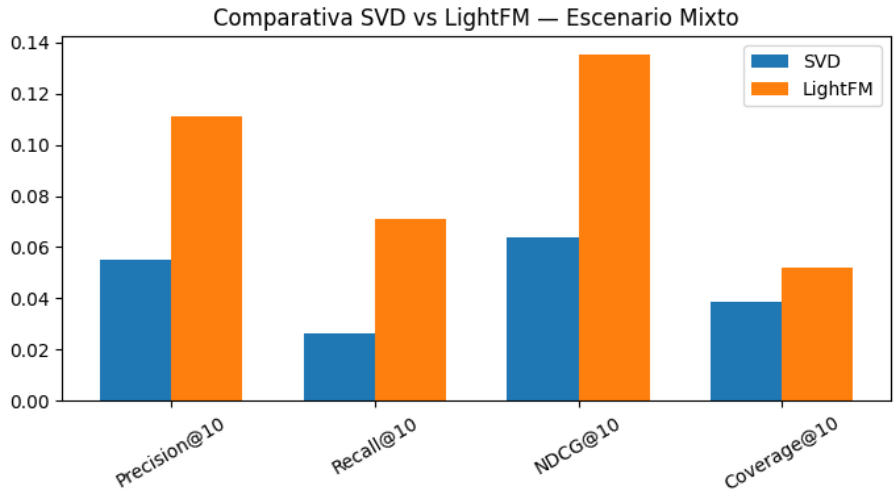

Figura 5.36. Comparativa de resultados de SVD vs LightFM: Escenario Mixto (10 % de CS ítems).

Para acabar, se hará un último experimento en el que se aumentará el porcentaje de *CS ítems* para ver cómo responden los modelos en cuanto a valores de las métricas (véase *Figura 5.37*):

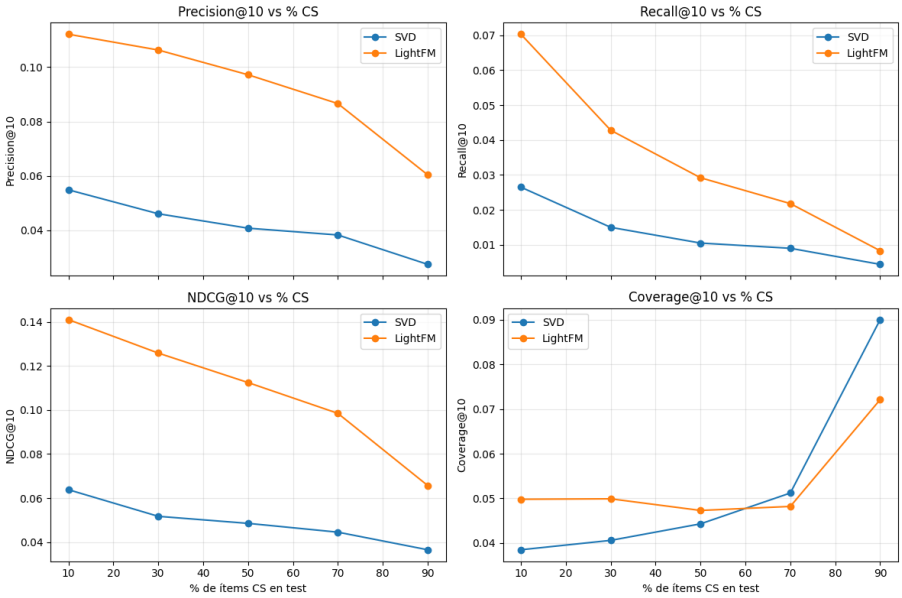

Figura 5.37. Comparativa de resultados de SVD vs LightFM

Las tendencias en todos los casos son similares, las situaciones interesantes son los del *recall@10* y el del *coverage@10*. El recall cae en ambos casos tanto debido a que se aumenta el número de ítems relevantes que se encuentran en el test, por lo que la proporción de ítems baja. En el caso del coverage, al meter ítems nuevos, aumenta el catálogo y los modelos tienen más posibilidades de recomendar ítems nuevos, por lo que este valor aumenta. Teniendo en cuenta los valores, no cabe duda de que el modelo *LightFM* es claramente mejor en casi todos los casos, como se esperaba desde un principio.

Capítulo 6. DISCUSIÓN

El presente trabajo se ha basado en comparar diferentes algoritmos de sistemas de recomendación y cómo pueden abordar los retos comunes como el *Next Best Item* y *Cold Start*. Los resultados muestran que los algoritmos basados en secuencias, como las redes *LSTM*, ofrecen un rendimiento superior en la predicción de interacciones, mientras que los modelos clásicos como *SVD* logran una mayor cobertura del catálogo, a costa de un menor rendimiento en el resto de métricas. Del mismo modo, *LightFM* se ha mostrado especialmente superior ante escenarios de arranque en frío, en particular cuando se incorporan características de ítems y metadatos, permitiendo generar recomendaciones razonables incluso cuando los ítems son nuevos y carecen de interacciones previas.

No obstante, los resultados también muestran varias limitaciones. Para empezar, la reducción de los datasets por motivos computacionales supuso una pérdida de diversidad y volumen de datos, lo que probablemente afectó a la tasa global de aciertos. Esto se observó con claridad en la subsección relacionada con el *Backtesting (5.1.3)*, donde la tasa de acierto se situó en torno al 4 %. Aun así, el análisis de las valoraciones de los ítems recomendados muestra que, cuando el modelo acierta, suele capturar con precisión las preferencias del usuario.

Otra limitación relevante deriva del problema anterior. El uso de Google Colab en su versión gratuita restringió tanto el tiempo de ejecución como la memoria disponible, obligando a realizar filtrados adicionales. Además, la implementación de modelos más avanzados como GRU4Rec se vio condicionada por dificultades relacionadas con incompatibilidades de versiones de librerías, lo que llevó a usar el algoritmo *LSTM* como alternativa.

En relación con los objetivos iniciales, puede afirmarse que la metodología adoptada ha sido útil para evaluar comparativamente distintos algoritmos bajo escenarios representativos. Sin embargo, como se ha mencionado anteriormente, ha sido necesario introducir diferentes cambios en la experimentación para ajustarse a las limitaciones, especialmente en lo relacionado con el tamaño de los datasets. Las adaptaciones tomadas son referencias aproximadas de los conjuntos de datos, lo que permite que se pueda evaluar sin perder la coherencia del trabajo, por lo que los resultados deben tomarse como aproximaciones y no como valores absolutos.

Las aportaciones de este trabajo confirman que no existe un algoritmo universalmente mejor para todos los retos, sino que cada reto obliga a tomar un enfoque dependiendo del contexto. Los modelos secuenciales (*LSTM*) resultan adecuados para capturar dinámicas de interacción a corto plazo, los modelos clásicos como *SVD* aportan mayor exploración del catálogo y los híbridos como *LightFM* destacan cuando hay presentes ítems nuevos sin un historial previo de interacciones. Este análisis refuerza la importancia de seleccionar el algoritmo de recomendación en función del problema concreto y de los recursos disponibles, más que el uso de un modelo único para todos los casos.

Capítulo 7. CONCLUSIONES

7.1 Conclusiones del trabajo

Durante este Trabajo Fin de Máster se ha podido ver una comparativa entre diferentes algoritmos de recomendación aplicados a los retos comunes que suelen presentar, específicamente el *Next Best Item* y el *Cold Start*. De manera resumida, los resultados muestran que:

- En el reto ***Next Best Item***, los modelos secuenciales basados en el algoritmo *LSTM* superan al enfoque clásico de SVD en la mayoría de métricas observadas, lo que confirma su capacidad para capturar patrones contextuales en las interacciones de los usuarios. Sin embargo, SVD resultó mejor en una métrica: la cobertura de catálogo. Dado que en este estudio la importancia reside en la calidad del sistema de recomendación, esta métrica tendría menos valor.
- En cuanto al reto ***Cold Start***, el modelo híbrido *LightFM* demostró ser mejor que el algoritmo *SVD*. Integrar metadatos de ítems le permitió al modelo híbrido ofrecer recomendaciones incluso para contenidos sin historial de interacciones. Aunque *SVD* mostró mejores resultados en la ordenación del ranking (*NDCG*) y la cobertura del catálogo, *LightFM* destacó en las métricas más importantes: precision y recall.
- Otra pequeña conclusión a tener en cuenta es la limitación computacional. La necesidad de reducir los datasets y ajustar la metodología condicionó la profundidad del análisis, aunque permitió obtener resultados representativos y comparables.

Por último, también se podría concluir que no existe un algoritmo universalmente mejor para todos los retos, sino que la idoneidad depende del contexto y las características del problema a resolver. Este trabajo aporta una visión sobre las fortalezas y limitaciones de cada enfoque, permitiendo ver si los algoritmos seleccionados son útiles ante los retos.

7.2 Conclusiones personales

El desarrollo de este proyecto ha supuesto un aprendizaje en diferentes ámbitos. En primer lugar, me ha permitido conocer más a fondo el campo de los sistemas de recomendación, un campo del que desconocía la mayor parte, y no solo desde un punto de vista teórico, sino también mediante la implementación práctica de modelos sobre datasets reales. Asimismo, he aprendido a gestionar las limitaciones que se han ido presentando a lo largo de este proyecto, lo que me ha obligado a adaptar el diseño experimental a los recursos disponibles, intentando obtener resultados representativos.

A nivel personal, considero que este trabajo ha reforzado mi capacidad para evaluar la selección de un algoritmo en función del problema al que se enfrente en cada caso, así como mi criterio para valorar los resultados más allá de las métricas. También me ha ayudado a comprender la relevancia de los sistemas de recomendación en el contexto actual y su potencial impacto en la experiencia de usuario, la fidelización y la personalización de servicios.

En definitiva, este proyecto ha sido una experiencia enriquecedora que me ha permitido adquirir conocimientos técnicos al mismo tiempo que ha reforzado mi interés por continuar investigando y profundizando en el ámbito de *machine learning*.

Capítulo 8. FUTURAS LÍNEAS DE TRABAJO

Viendo los resultados obtenidos y de las limitaciones encontradas, se identifican algunas líneas de investigación que se podrían dar para ahondar en el análisis:

- ***Ampliación y explotación de los datasets***: debido a las limitaciones computacionales el análisis se vio afectado, siendo necesario reducir el volumen de datos de los conjuntos utilizados. Una línea de trabajo continuista de este proyecto consistiría en entrenar los modelos con los datasets completos. Lo que permitiría evaluar el rendimiento de los algoritmos en nuevas casuísticas. Asimismo, se podría comprobar la escalabilidad de los algoritmos y analizar si la mayor diversidad de datos incrementa la tasa de aciertos observada durante el backtesting.
- ***Uso de modelos secuenciales más avanzados***: aunque el modelo *LSTM* mostró un rendimiento superior al algoritmo *SVD* en la predicción del *Next Best Item*, el uso de otros algoritmos o arquitecturas más novedosas como *Transformers* o la implementación de *GRU4Rec* permitiría explorar la capacidad de capturar secuencias a largo plazo y patrones más complejos en las interacciones. Implementar estos modelos conllevaría un mayor esfuerzo en el preprocesamiento y en recursos computacionales, pero podría mejorar significativamente la calidad de las recomendaciones. Además de dar otra visión más exhaustiva al comparar el rendimiento directo con el modelo *LSTM* a través de las métricas exploradas en este trabajo.
- ***Combinación de enfoques híbridos para Cold Start***: *LightFM* ha demostrado buenos resultados al incorporar metadatos de ítems, pero su rendimiento podría verse beneficiado al integrarlo con modelos secuenciales. Esta combinación supondría diseñar una arquitectura capaz de aprovechar la información contextual de las secuencias y los atributos de ítems y usuarios. Implementando este enfoque híbrido implicaría adaptar las métricas de evaluación específicas para este escenario además de diseñar una nueva arquitectura.

Estas líneas de trabajo permitirían superar algunas de las mayores limitaciones detectadas en este estudio y profundizar en el análisis de cómo los sistemas de recomendación se adaptan a entornos más complejos.

Bibliografía

[1] J. M. Ph.D. y E. Kavlakoglu, *¿Qué es el filtrado colaborativo?* Ibm.com, 2024. dirección: https://www.ibm.com/es-es/think/topics/collaborative-filtering (visitado 14-05-2025).

[2] Q. M. Ilyas, A. Mehmood, A. Ahmad y M. Ahmad, *A Systematic Study on a Customer's Next-Items Recommendation Techniques*, 2022. dirección: https://www.mdpi.com/2071-1050/14/12/7175 (visitado 14-05-2025).

[3] Q. M. Ilyas, A. Mehmood, A. Ahmad y M. Ahmad, *Improving Data Sparsity in Recommender Systems Using Matrix Regeneration with Item Features*, 2023. dirección: https://www.mdpi.com/2227-7390/11/2/292 (visitado 16-05-2025).

[4] Q. M. Ilyas, A. Mehmood, A. Ahmad y M. Ahmad, *Session-based Recommendations with Recurrent Neural Networks*, 2015. dirección: https://arxiv.org/abs/1511.06939 (visitado 23-06-2025).

[5] R. Hamad, *What is LSTM? Introduction to Long Short-Term Memory*, 2023. dirección: https://medium.com/@rebeen.jaff/what-is-lstm-introduction-to-long-short-term-memory-66bd3855b9ce (visitado 25-07-2025).

[6] M. Kula, *Metadata Embeddings for User and Item Cold-start Recommendations*, 2015. dirección: https://arxiv.org/abs/1507.08439 (visitado 23-06-2025).

[7] A. Tam, *Using Singular Value Decomposition to Build a Recommender System*, 2021. dirección: https://machinelearningmastery.com/using-singular-value-decomposition-to-build-a-recommender-system/ (visitado 23-06-2025).

[8] R. Gupta, *How Singular Value Decomposition (SVD) is used in Recommendation Systems, "Clearly Explained".* 2023. dirección: https://medium.com/@ritik_gupta/how-singular-value-decomposition-svd-is-used-in-recommendation-systems-clearly-explained-201b24e175db (visitado 23-06-2025).

[9] *Recommender Systems using KNN".* geeksforgeeks.org, 2024. dirección: https://www.geeksforgeeks.org/machine-learning/recommender-systems-using-knn/ (visitado 23-06-2025).

[10] K. Pullakandam, *Understanding Precision, Recall, and F-Score at K in Recommender Systems*, 2024. dirección: https://krishnapullak.medium.com/understanding-precision-recall-and-f-score-at-k-in-recommender-systems-7146a0dce68e (visitado 25-06-2025).

[11] A. Dhinakaran, *Demystifying NDCG*, 2023. dirección: https://medium.com/data-science/demystifying-ndcg-bee3be58cfe0 (visitado 25-06-2025).

[12] A. Dhinakaran, *10 metrics to evaluate recommender and ranking systems*, 2025. dirección: https://www.evidentlyai.com/ranking-metrics/evaluating-recommender-systems#diversity (visitado 26-06-2025).

[13] CodeSignal, *Introduction to Coverage in Recommendation Systems*, 2025. dirección: https://codesignal.com/learn/courses/recommendation-systems-quality-evaluation/lessons/understanding-and-calculating-coverage-in-recommendation-systems (visitado 26-06-2025).

[14] M. Tech, *¿Qué es Google Colab?* 2025. dirección: https://www.mentorestech.com/resource-blog-content/que-es-google-colab (visitado 28-08-2025).

[15] AWS, *¿Qué es Python?* 2025. dirección: https://aws.amazon.com/es/what-is/python/ (visitado 28-08-2025).

[16] Aurora, *Pandas: La Herramienta Esencial para Data Science en Python*, 2024. dirección: https://iddigitalschool.com/bootcamps/que-es-pandas/ (visitado 28-08-2025).

[17] D. D. Lab, *What is Scikit-learn?* 2025. dirección: https://domino.ai/data-science-dictionary/sklearn (visitado 28-08-2025).

[18] S. Anand, *Amazon Product Reviews: Electronics*, 2019. dirección: https://www.kaggle.com/datasets/saurav9786/amazon-product-reviews/data (visitado 16-07-2025).

[19] N. Hug, *Librería surprise*, 2019. dirección: https://surpriselib.com/ (visitado 19-07-2025).

[20] J. Chen, *Backtesting: Definition, How It Works, and Downsides*, 2024. dirección: https://www.investopedia.com/terms/b/backtesting.asp.

[21] M. Kula, *LightFM's Documentation*, 2016. dirección: https://making.lyst.com/lightfm/docs/home.html (visitado 10-08-2025).

[22] Seobility.net, *Pesos TF-IDF*, 2025. dirección: https://www.seobility.net/es/wiki/TF_IDF#content-que-es-tf-idf (visitado 12-08-2025).