

Trabajo fin de grado

Análisis Comparativo de Sistemas de Recomendación Secuenciales
en la Plataforma Steam



Lucas Rengifo Garcia

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR



Grado en Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

**Análisis Comparativo de Sistemas de
Recomendación Secuenciales en la Plataforma
Steam**

Autor: Lucas Rengifo Garcia

Tutor: Alejandro Bellogin Kouki

julio 2024

Todos los derechos reservados.

Queda prohibida, salvo excepción prevista en la Ley, cualquier forma de reproducción, distribución comunicación pública y transformación de esta obra sin contar con la autorización de los titulares de la propiedad intelectual.

La infracción de los derechos mencionados puede ser constitutiva de delito contra la propiedad intelectual (*arts. 270 y sgts. del Código Penal*).

DERECHOS RESERVADOS

© 20 de julio de 2024 por UNIVERSIDAD AUTÓNOMA DE MADRID
Francisco Tomás y Valiente, nº 1
Madrid, 28049
Spain

Lucas Rengifo Garcia

Análisis Comparativo de Sistemas de Recomendación Secuenciales en la Plataforma Steam

Lucas Rengifo Garcia

Para quienes me han inspirado y apoyado, este éxito es de todos.

AGRADECIMIENTOS

Quiero expresar mi más profundo agradecimiento a Alejandro Bellogín, mi tutor de TFG. Su inmensa ayuda, interés constante y dedicación han sido invaluables para la realización de este proyecto. Su guía y conocimiento han sido fundamentales para alcanzar los objetivos planteados y superar los desafíos encontrados en el camino.

Agradezco también a mi familia, por su amor incondicional y apoyo constante, a mis amigos, por su ánimo y compañerismo, y a Angela, por su paciencia, cariño y amor. Sin el apoyo de todos ellos, este logro no habría sido posible. Este trabajo es tanto mío como de vosotros, y os estaré eternamente agradecido.

Finalmente, quiero agradecer a la Universidad Autónoma de Madrid por brindarme la oportunidad de desarrollar mis estudios en un entorno académico de excelencia. Los recursos, la comunidad académica y las experiencias vividas en esta institución han sido fundamentales para mi formación y crecimiento personal y profesional.

RESUMEN

En el contexto de la era digital, donde los sistemas de recomendación desempeñan un papel crucial en la personalización de la experiencia del usuario, este Trabajo de Fin de Grado explora el uso de la librería RecBole para el estudio y la evaluación de modelos de recomendación secuenciales. Se realizó una comparación exhaustiva de diversos modelos sobre una base de datos de Steam, una de las plataformas líderes en distribución de juegos. El objetivo fue identificar qué modelos proporcionan las recomendaciones más precisas y relevantes para los usuarios.

Se estudiaron y analizaron varios algoritmos de recomendación secuenciales, que consideran las interacciones pasadas de los usuarios para predecir sus acciones futuras. Los resultados obtenidos de estos modelos se compararon para determinar su efectividad en el contexto real de recomendaciones de juegos.

Para facilitar la interpretación y visualización de los resultados de estos modelos, se desarrolló una aplicación utilizando el framework Django. Esta aplicación permite a los usuarios visualizar de manera interactiva las recomendaciones generadas por los diferentes algoritmos, ofreciendo una herramienta valiosa tanto para usuarios finales interesados en descubrir nuevos juegos como para investigadores y desarrolladores que buscan mejorar la eficacia de los sistemas de recomendación.

PALABRAS CLAVE

Sistemas de recomendación, Modelos secuenciales, Sistemas de evaluación, RecBole

ABSTRACT

In the context of the digital era, where recommender systems play a crucial role in the personalization of the user experience, this Final Degree Thesis explores the use of the RecBole library for the study and evaluation of sequential recommender models. A comprehensive comparison of several models was performed on a database of Steam, one of the leading game distribution platforms. The objective was to identify which models provide the most accurate and relevant recommendations for users.

Several sequential recommendation algorithms, which consider users' past interactions to predict their future actions, were studied and analyzed. The results obtained from these models were compared to determine their effectiveness in the real context of game recommendations.

To facilitate the interpretation and visualization of the results of these models, an application was developed using the Django framework. This application allows users to interactively visualize the recommendations generated by the different algorithms, providing a valuable tool both for end users interested in discovering new games and for researchers and developers seeking to improve the effectiveness of recommendation systems.

KEYWORDS

Recommender systems, Sequential models, Evaluation systems, RecBole

ÍNDICE

1	Introducción	1
1.1	Motivación	1
1.2	Objetivos	2
2	Estado del arte	3
2.1	Sistemas de recomendacion	3
2.1.1	Recomendadores secuenciales	4
2.1.2	Aplicaciones para Steam	4
2.1.3	Evaluación de los recomendadores	6
2.1.4	Librería para el módulo de recomendadores	6
2.2	Modelos seleccionados	8
2.2.1	GRU4REC	8
2.2.2	FOSSIL	9
2.2.3	HRM	10
2.2.4	SRGNN	12
2.2.5	BERT4Rec	12
2.2.6	HGN	14
2.2.7	SHAN	15
2.2.8	NPE	16
2.3	Desarrollo web	17
2.3.1	La elección de Django	17
2.3.2	El ORM de Django	18
3	Diseño, análisis e implementación	21
3.1	Diseño del problema	21
3.1.1	Estructura	21
3.1.2	Análisis	22
3.2	Obtención y preprocesamiento de los datos	23
3.3	Módulo de recomendación	24
3.4	Aplicación web	25
3.4.1	Arquitectura	25
3.4.2	Funcionalidad	26
4	Pruebas y resultados	27

4.1 Entorno de pruebas	27
4.2 Primeros resultados y elección de algoritmos	28
4.3 Experimentos	28
4.3.1 SRGNN	33
4.3.2 HRM	35
4.3.3 SHAN	36
4.4 Análisis de los resultados	38
5 Conclusión y trabajo futuro	39
5.1 Conclusión	39
5.2 Trabajo futuro	40
Bibliografía	43
Apéndices	45
A Fichero requirements	47

LISTAS

Lista de figuras

2.1	Recomendador de Steam	5
2.2	Funcionamiento GRU4REC	9
2.3	Funcionamiento FOSSIL	10
2.4	Funcionamiento HRM	11
2.5	Funcionamiento SRGNN	13
2.6	Funcionamiento HGN	15
2.7	Funcionamiento SHAN	16
2.8	Funcionamiento NPE	17
3.1	Diagrama de estructura	22
3.2	Estructura MVT	25
3.3	Diagrama de secuencia	26
4.1	Características del equipo	27
4.2	Archivo de configuración	29
4.3	Proceso entrenamiento y valoración de modelo en RecBole	30
4.4	Mejores valoraciones de los modelos	30
4.5	Resultados y valoración de los modelos	31
4.6	Resultados con 100 iteraciones	32
4.7	Evolución de las métrica hasta 15 iteraciones	33
4.8	Resultados estudio hiperparámetros SRGNN	34
4.9	Resultados estudio hiperparámetros HRM	35
4.10	Resultados estudio hiperparámetros SHAN	37
4.11	Resultados finales valoraciones de modelos	38

Lista de tablas

INTRODUCCIÓN

La industria de los videojuegos, con Steam como uno de sus principales exponentes, desempeña un papel significativo en el mercado global, generando ingresos multimillonarios anualmente [1]. Este éxito ha resultado en un notable incremento en el tamaño del mercado, tanto en términos de demanda como de oferta [2]. Este crecimiento ha complicado la tarea de encontrar juegos que no solo sean apropiados sino también exitosos como recomendaciones. Para las empresas más pequeñas, lograr que sus juegos destaqueen se ha vuelto especialmente desafiante, considerando que, según datos del año 2023, solo en Steam se lanzaron más de 14,000 juegos, un incremento de 2,000 juegos respecto al año anterior [3]. Este escenario subraya la necesidad de explorar y desarrollar sistemas de recomendación más sofisticados y eficientes para la industria del videojuego.

En este capítulo, abordaremos la motivación, los objetivos y la estructura de este trabajo de final de grado. Además, realizaremos un repaso breve de los términos y conceptos importantes que se utilizarán a lo largo del documento. Este enfoque no solo contextualiza el estudio, sino que también establece el marco necesario para entender las soluciones propuestas y su impacto potencial en el mercado de videojuegos a través de plataformas como Steam.

1.1. Motivación

El sector de los videojuegos ha experimentado un crecimiento exponencial durante los últimos años, reflejado notablemente en el aumento del catálogo de plataformas de venta como Steam. Este crecimiento ha generado un entorno competitivo desafiante para los desarrolladores independientes, quienes enfrentan dificultades para ganar visibilidad en un mercado saturado de opciones. Sin una adecuada visibilidad mediática o inversión en publicidad, resulta complicado para los juegos menores destacar entre la multitud.

Ante este panorama, el avance en tecnologías de inteligencia artificial y aprendizaje automático presenta una oportunidad única para abordar estos retos. Específicamente, el uso de sistemas de recomendación puede ser una solución viable para mejorar la visibilidad de juegos menos conocidos. En este contexto, hemos seleccionado RecBole como herramienta principal para nuestra investigación.

RecBole, lanzado a finales de 2020, es un marco de trabajo unificado y eficiente basado en PyTorch, diseñado específicamente para facilitar la investigación y desarrollo de sistemas de recomendación. Este marco soporta más de 100 modelos de recomendación y proporciona acceso a múltiples conjuntos de datos de referencia, incluyendo el dataset de Steam, lo que justifica su elección para este estudio ¹.

Dentro de RecBole, se distinguen cuatro categorías principales de modelos de recomendación: recomendación general, recomendación basada en contexto, recomendación secuencial y recomendación basada en conocimiento. Para esta investigación, como se verá en detalle en el siguiente capítulo, nos centraremos en los modelos de recomendación secuencial debido a su adaptabilidad a la estructura de datos de Steam, con el objetivo de desarrollar sistemas robustos que entreguen recomendaciones precisas y relevantes para cada usuario en la plataforma. Este enfoque no solo busca mejorar la experiencia del usuario sino también ofrecer a los desarrolladores independientes una mejor oportunidad de destacar en un mercado altamente competitivo.

1.2. Objetivos

En el presente proyecto de fin de grado se plantean dos objetivos principales. En primer lugar, se busca comprender a fondo el funcionamiento de la herramienta RecBole, especializándose en el uso y comparación de sus modelos secuenciales de recomendación. Esto permitirá identificar cuáles son los modelos más efectivos para su aplicación en recomendaciones de videojuegos. Además, se realizará un estudio detallado de los modelos más prometedores que obtengan la mejor puntuación para tratar de maximizar sus valores mediante el ajuste de hiperparámetros.

El segundo objetivo consiste en el desarrollo de una aplicación web interactiva que permita al usuario final visualizar de manera gráfica la efectividad de los diferentes modelos de recomendación. Esta herramienta ofrecerá la funcionalidad de seleccionar un usuario y un modelo de recomendación, y como resultado, presentará una lista ordenada de videojuegos ajustada a las preferencias detectadas según el modelo elegido. Este aspecto del proyecto no solo tiene como fin exhibir los resultados reales obtenidos en los experimentos, sino también facilitar la interpretación y la comprensión de estos por parte de los usuarios.

¹ RecBole oficial page, <https://recbole.io/>. Accessed: Jul. 9, 2024.

ESTADO DEL ARTE

En este capítulo se explorarán diversos conceptos clave para el entendimiento de este trabajo.

2.1. Sistemas de recomendación

Los sistemas de recomendación (SR) son herramientas y técnicas de software que ofrecen sugerencias de artículos útiles para un usuario [4]. Estos sistemas son utilizados por numerosas plataformas para personalizar la experiencia del usuario al sugerir productos, servicios o información que se alinean con sus intereses y comportamientos previos. Estos sistemas emplean algoritmos que analizan grandes volúmenes de datos para identificar patrones y hacer predicciones sobre lo que podría interesar a cada usuario.

El funcionamiento de los sistemas de recomendación ha experimentado una evolución significativa gracias a los avances en el aprendizaje automático (Machine Learning). Inicialmente, los motores de búsqueda, las plataformas de contenido y las ventas de productos operaban mediante rankings o listas de popularidad. Si bien estos sistemas eran funcionales hasta cierto punto, no podían personalizar la experiencia del usuario y a menudo mostraban elementos que no se correspondían con sus intereses específicos [5].

Los sistemas de recomendación tienen varios objetivos clave que contribuyen a mejorar tanto la experiencia del usuario como el rendimiento de la plataforma. En primer lugar, buscan mejorar la experiencia del usuario mediante la personalización del contenido mostrado, adaptándose a sus preferencias y comportamientos. En segundo lugar, estos sistemas tienen como objetivo aumentar las ventas y la retención de usuarios, proporcionando recomendaciones relevantes que fomentan la compra y el uso continuado del servicio. Además, optimizan la interacción del usuario con la plataforma, reduciendo el tiempo necesario para encontrar contenido de interés. Finalmente, recopilan y analizan datos de los usuarios para mejorar continuamente las recomendaciones y otros aspectos del servicio, permitiendo una retroalimentación constante y mejoras en el algoritmo [6].

Los sistemas de recomendación se pueden clasificar en varios tipos principales:

- **Basados en contenido:** Recomendaciones según las características de los ítems que el usuario ha mostrado interés.
- **Colaborativos:** Basados en el comportamiento y las preferencias de usuarios similares.
- **Híbridos:** Combinan métodos basados en contenido y colaborativos para mejorar la precisión.
- **Basados en conocimiento:** Utilizan información explícita sobre los usuarios y los ítems para realizar recomendaciones.

Para nuestro estudio utilizaremos la biblioteca RecBole donde le daremos un enfoque al proyecto para tratar de resolver el problema utilizando los recomendadores secuenciales.

2.1.1. Recomendadores secuenciales

Los modelos secuenciales de RecBole se enmarcan en los sistemas de recomendación basados en contenido y colaborativos, específicamente en un enfoque híbrido. Estos modelos utilizan la secuencia de interacciones del usuario con los ítems para hacer recomendaciones más precisas, capturando patrones temporales y de comportamiento a lo largo del tiempo. Así, pueden predecir mejor las futuras preferencias del usuario en función de su historial de actividades.

Estos modelos son particularmente útiles en escenarios donde el orden y el contexto temporal de las interacciones son cruciales para generar recomendaciones precisas y relevantes.

Estos modelos suelen utilizarse principalmente por dos razones. La primera es la predicción de clics en una página web, anticipando cuál será el próximo elemento en el que el usuario hará clic, generalmente almacenando los datos mediante cookies. La segunda es la predicción de los carritos de compra de los usuarios, analizando sus historiales de compra para recomendar productos relevantes. Estos enfoques permiten personalizar la experiencia del usuario y mejorar la eficiencia de las recomendaciones.

Los modelos secuenciales de RecBole, que poseen estructuras y funcionamientos muy variados, utilizan diversas técnicas que abarcan desde métodos basados en cadenas de Markov hasta redes neuronales recurrentes y transformadores. Estas técnicas permiten capturar la secuencia temporal y los patrones de comportamiento de los usuarios, mejorando así la precisión de las recomendaciones al anticipar futuras interacciones basadas en el historial de actividades de cada usuario. Cada enfoque tiene sus propias ventajas y desafíos, lo que permite seleccionar la técnica más adecuada según el contexto y los objetivos específicos del sistema de recomendación.

2.1.2. Aplicaciones para Steam

Debido al ya comentado exponencial crecimiento del tamaño del mercado de los videojuegos, y en concreto de Steam, dar a conocer los videojuegos de sobre todo pequeños desarrolladores es algo

especialmente desafiante. Sin embargo, Steam también es conocedora de esto por lo que en verano de 2019 decidió publicar la herramienta de “Steam Interactive Recommender” [7].

El recomendador interactivo de Steam utiliza técnicas de aprendizaje automático (red neuronal) para ofrecer sugerencias personalizadas basadas en el historial de juegos del usuario. El sistema analiza los juegos que el usuario ha jugado y disfrutado, y luego utiliza estos datos para recomendar títulos similares. Los usuarios pueden ajustar las recomendaciones mediante filtros, como el nivel de popularidad y la antigüedad de los juegos, lo que permite una mayor personalización. Además, el recomendador se actualiza continuamente con datos en tiempo real para mejorar la precisión de las sugerencias. Este enfoque ayuda a los usuarios a descubrir nuevos juegos adaptados a sus preferencias individuales, mejorando su experiencia general en la plataforma.

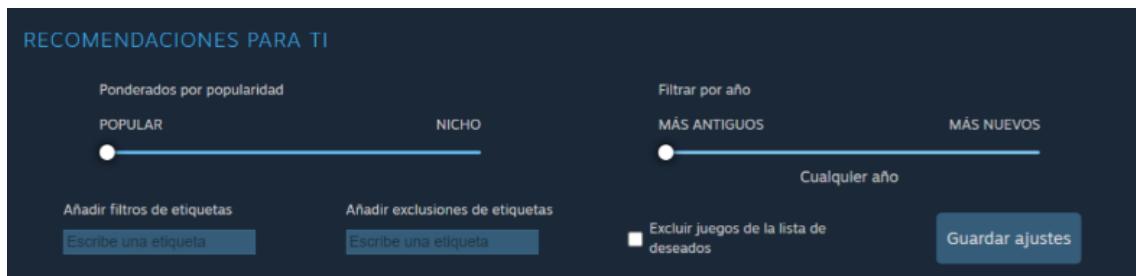


Figura 2.1: Filtros de recomendación en Steam Recommender [7].

Además, el recomendador no se nutre de metadatos de las publicaciones de Steam ni de las reseñas de los usuarios provocando así que sea el propio algoritmo el que aprenda de los juegos, los únicos datos que tiene de los ítems son la fecha de salida para poder filtrar por antigüedad como se muestra en la Figura 2.1.

“A diferencia de otras estrategias más tradicionales, nosotros no alimentamos de forma explícita nuestro modelo con información sobre los juegos. En su lugar, el modelo aprende de los juegos por sí mismo durante el proceso de entrenamiento. De hecho, la única información sobre el juego que se incorpora de forma explícita al proceso es la fecha de lanzamiento, lo cual permite hacer la selección del marco temporal en el selector de fecha de lanzamiento.” [7]

Para nuestro estudio se decidió darle la vuelta al concepto de los recomendadores secuenciales, y sustituir esa idea de utilizarlos para secuencias de clics por sustituirlo por una secuencia de acciones, viendo en nuestro caso concreto las acciones como el hecho de jugar un videojuego de la plataforma de Steam. Esto permite una interpretación y enfoque diferentes a los modelos tradicionales, ofreciendo una perspectiva más detallada y personalizada de las preferencias de los usuarios, y viendo si resulta en un enfoque prometedor para los recomendadores secuenciales.

2.1.3. Evaluación de los recomendadores

Las métricas de evaluación son fundamentales en el estudio de sistemas de recomendación porque permiten cuantificar la eficacia y eficiencia de los algoritmos. Estas métricas proporcionan una base objetiva para comparar diferentes modelos y técnicas, identificando cuáles ofrecen las mejores recomendaciones en términos de precisión, exhaustividad y relevancia. Además, ayudan a entender cómo las recomendaciones se alinean con las expectativas y necesidades de los usuarios, permitiendo optimizar los sistemas para mejorar la experiencia del usuario. Sin métricas de evaluación, sería difícil medir el impacto y la calidad de las recomendaciones, limitando la capacidad de mejorar y adaptar los sistemas de recomendación a contextos específicos [8].

Son estas métricas las que nos permitirán comparar cuál de los algoritmos a probar se adaptan mejor a las necesidades del proyecto y cuales dan mejor desempeño.

- **Precision:** La precisión mide la proporción de recomendaciones relevantes entre las recomendaciones hechas. Es una métrica utilizada muy frecuentemente en la comunidad.
- **Recall:** El recall mide la proporción de elementos relevantes que fueron recomendados entre todos los elementos relevantes disponibles. Es otra de las métricas más comunes en sistemas de recomendación.
- **MRR (Mean Reciprocal Rank):** El MRR evalúa la calidad de las recomendaciones basándose en la posición del primer elemento relevante en la lista recomendada.
- **Normalized Discounted Cumulative Gain (NDCG):** El NDCG mide la calidad de las recomendaciones considerando la posición de todos los elementos relevantes en la lista recomendada.
- **Tasa de Aciertos (Hit):** La tasa de aciertos mide la proporción de usuarios para los que al menos un elemento relevante se encuentra en la lista de recomendaciones.

2.1.4. Librería para el módulo de recomendadores

Para el desarrollo de este proyecto se van a utilizar diversas librerías, entre ellas contaremos con PyTorch, TensorFlow y, principalmente, RecBole.

PyTorch:

PyTorch es una biblioteca de código abierto para el desarrollo y entrenamiento de modelos de aprendizaje profundo. Desarrollada por el equipo de investigación de inteligencia artificial de Facebook (Facebook AI Research, FAIR), PyTorch proporciona una interfaz flexible y dinámica que permite a los investigadores y desarrolladores implementar de manera eficiente modelos de redes neuronales complejas [9].

Una de las características distintivas de PyTorch es su uso de grafos computacionales dinámicos, lo que facilita la construcción y modificación de arquitecturas de redes neuronales sobre la marcha. Esto contrasta con los grafos estáticos utilizados por otras bibliotecas como TensorFlow, ofreciendo una mayor flexibilidad y simplificación en el proceso de desarrollo de modelos.

PyTorch también incluye soporte nativo para operaciones tensoriales aceleradas por GPU, lo que permite un entrenamiento rápido y eficiente de los modelos. Además, cuenta con una amplia variedad de herramientas y bibliotecas complementarias, como torchvision para tareas de visión por computadora y torchaudio para procesamiento de audio, que amplían sus capacidades y facilitan su uso en diferentes aplicaciones.

Debido a su accesibilidad y potente funcionalidad, PyTorch se ha convertido en una herramienta esencial tanto para la investigación en inteligencia artificial como para el desarrollo de aplicaciones industriales, siendo ampliamente adoptada en la academia y la industria [10].

Tensorflow:

TensorFlow es una biblioteca de código abierto desarrollada por Google Brain para el desarrollo y entrenamiento de modelos de aprendizaje automático y aprendizaje profundo. TensorFlow proporciona una plataforma robusta y flexible para construir y desplegar modelos de inteligencia artificial en una amplia variedad de dispositivos, desde servidores hasta dispositivos móviles y navegadores web.

TensorFlow también ofrece un ecosistema rico de herramientas y bibliotecas complementarias, como Keras, una API de alto nivel que simplifica la construcción de modelos de aprendizaje profundo, y TensorFlow Lite, que permite la ejecución de modelos en dispositivos móviles y embebidos. Además, TensorFlow Serving facilita el despliegue de modelos en producción, permitiendo una integración fluida con aplicaciones empresariales [11].

Gracias a su versatilidad y amplio soporte comunitario, TensorFlow se ha consolidado como una de las principales herramientas en el campo de la inteligencia artificial, siendo ampliamente utilizada en la investigación académica y en diversas industrias para desarrollar soluciones innovadoras y escalables.

Recbole:

RecBole es una biblioteca de código abierto diseñada para facilitar el desarrollo, entrenamiento y evaluación de sistemas de recomendación. Desarrollada por un equipo colaborativo de investigadores de varias universidades, RecBole proporciona un marco integral y flexible para implementar diversos algoritmos de recomendación, abarcando desde métodos tradicionales hasta enfoques avanzados de aprendizaje profundo.

Una de las características distintivas de RecBole es su arquitectura modular, que permite a los usuarios combinar fácilmente diferentes componentes y técnicas para construir sistemas de recomendación personalizados. La biblioteca incluye una amplia gama de modelos de recomendación predefinidos, que cubren métodos basados en contenido, filtrado colaborativo, modelos secuenciales y enfoques híbridos. Esto facilita la comparación y evaluación de diferentes algoritmos bajo un mismo entorno.

RecBole también ofrece soporte nativo para operaciones tensoriales aceleradas por GPU, lo que

permite un entrenamiento eficiente de los modelos. Además, proporciona herramientas para la gestión de datos, preprocesamiento, ajuste de hiperparámetros y evaluación de rendimiento, simplificando así todo el ciclo de vida del desarrollo de sistemas de recomendación.

Gracias a su facilidad de uso y amplia funcionalidad, RecBole se ha convertido en una herramienta valiosa tanto para investigadores como para desarrolladores en el ámbito de los sistemas de recomendación. Su capacidad para integrar y comparar múltiples algoritmos bajo un mismo framework facilita la innovación y optimización de soluciones de recomendación en una variedad de aplicaciones industriales y académicas [12].

2.2. Modelos seleccionados

Ahora abarcaremos los algoritmos que van a ser estudiados para poder comprender su funcionamiento en la sección de pruebas. En un inicio, se iban a analizar otros módulos también, sin embargo, debido a la ausencia de documentación y de pruebas funcionales se decantó por descartarlos.

Durante esta sección se van a estudiar el funcionamiento de los algoritmos pasando por sus componentes clave y los hiperparámetros más relevantes de cada modelo.

2.2.1. GRU4REC

GRU4Rec (Gated Recurrent Units for Recommendations) es un modelo de recomendación basado en redes neuronales recurrentes (RNNs) específicamente diseñado para recomendaciones basadas en sesiones. Utiliza GRUs para capturar dependencias temporales en las secuencias de interacciones del usuario [13].

Componentes Clave:

- **Embeddings de Ítems:** Transforma los ítems en vectores de dimensión reducida.
- **Capa GRU:** Procesa la secuencia de interacciones del usuario, actualizando su estado oculto para capturar la información temporal.
- **Capa de Salida:** Genera las recomendaciones a partir del estado oculto final.

Parámetros Importantes:

- **embedding_size:** Tamaño del vector de embeddings.
- **hidden_size:** Número de unidades en el estado oculto de la GRU.
- **num_layers:** Número de capas en la GRU.
- **dropout_prob:** Tasa de abandono para evitar el sobreajuste.
- **loss_type:** Tipo de función de pérdida, ya sea clasificación múltiple ('CE') o optimización por pares ('BPR').

Funcionamiento:

- **Input:** Secuencia de interacciones del usuario.
- **Embedding:** Cada ítem se convierte en un vector de embedding.
- **Procesamiento Secuencial:** La GRU procesa la secuencia de embeddings, actualizando su estado oculto en cada paso temporal.
- **Generación de Recomendaciones:** A partir del estado oculto final, el modelo predice los siguientes ítems relevantes.

Este modelo es eficaz para capturar patrones de comportamiento secuencial y proporcionar recomendaciones personalizadas basadas en el historial de interacciones de los usuarios. Sin embargo, ya se verá en los resultados que no se generan los mejores resultados. La Figura 2.2 muestra el funcionamiento del algoritmo.

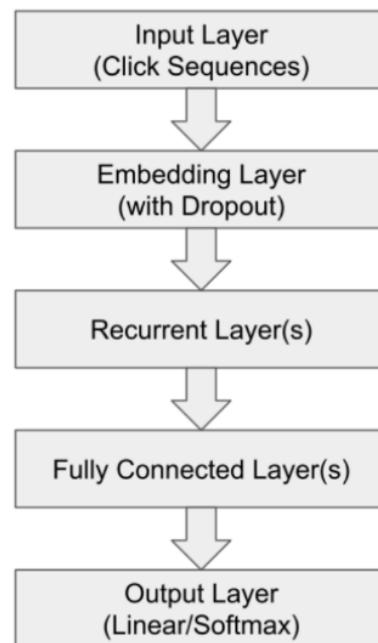


Figura 2.2: Funcionamiento de GRU4REC [14].

2.2.2. FOSSIL

FOSSIL (Fusing Similarity Models with Markov Chains) es un modelo de recomendación diseñado para abordar la predicción de comportamiento secuencial en sistemas con datos escasos. Combina métodos de factorización matricial con cadenas de Márkov para capturar tanto las preferencias a largo plazo como las dinámicas secuenciales a corto plazo [15].

Componentes Clave:

- **Modelos de Similitud:** Capturan interacciones usuario-ítem y las relaciones entre ítems.
- **Cadenas de Márkov:** Una cadena de Márkov es un modelo matemático que representa un sistema donde el

estado futuro depende únicamente del estado actual y no de los estados previos (propiedad de Márkov). Este modelo se utiliza para predecir una secuencia de eventos, donde cada evento está condicionado solo al evento inmediatamente anterior. Las cadenas de Márkov son útiles en diversos campos, como la economía, la física y el aprendizaje automático, para modelar sistemas estocásticos y procesos temporales.

Parámetros Importantes:

- **embedding_size:** Tamaño del vector de embeddings de usuarios e ítems.
- **order_len:** Número de ítems recientes considerados en las recomendaciones.
- **reg_weight:** Peso de regularización L2.
- **alpha:** Parámetro para calcular la similitud.
- **loss_type:** Tipo de función de pérdida ('CE' para clasificación múltiple o 'BPR' para optimización por pares).

Funcionamiento:

- **Input:** Historial de interacciones del usuario.
- **Embedding:** Transformación de usuarios e ítems en vectores.
- **Modelos de Similitud y Cadenas de Márkov:** Combinación de estos enfoques para capturar tanto preferencias a largo plazo como patrones secuenciales.
- **Generación de Recomendaciones:** Predicción de ítems relevantes basándose en el historial del usuario y la dinámica secuencial.

Esta combinación entre predicciones a largo plazo y cadenas de Márkov aporta una visión muy interesante para nuestro problema, combinando así los juegos que ha jugado una persona en su histórico y el o los juegos que ha jugado más recientemente.

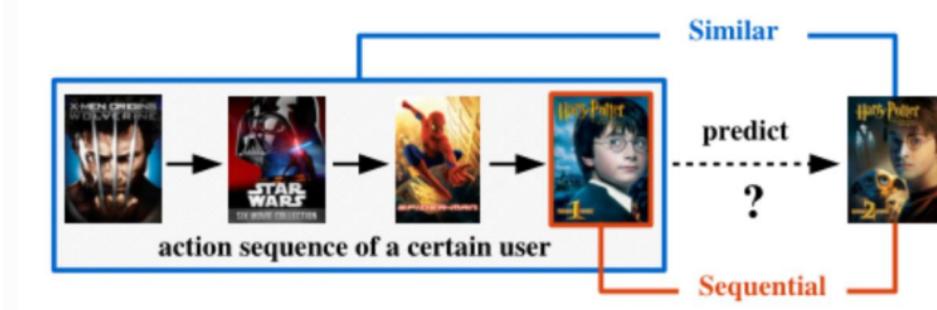


Figura 2.3: Funcionamiento de FOSSIL [16].

2.2.3. HRM

HRM (Hierarchical Representation Model) es un modelo de recomendación diseñado para capturar las interacciones complejas y jerárquicas entre ítems en secuencias de usuario. Utiliza una arquitectura jerárquica para representar y aprender tanto las relaciones locales como globales en las secuencias de ítems [17].

Componentes Clave:

- **Representación Jerárquica:** Captura relaciones a diferentes niveles de la secuencia de ítems.
- **Modelo de Factores Latentes:** Utiliza embeddings para representar usuarios e ítems.
- **Redes Neuronales:** Procesa y combina las representaciones jerárquicas para predecir las preferencias del usuario.

Parámetros Importantes:

- **embedding_size:** Tamaño del vector de embeddings.
- **hidden_size:** Número de unidades en las capas ocultas de la red neuronal.
- **dropout_prob:** Tasa de abandono para evitar el sobreajuste.
- **loss_type:** Tipo de función de pérdida (por ejemplo, 'BPR' o 'CE').

Funcionamiento:

- **Input:** Secuencia de interacciones del usuario.
- **Representación Jerárquica:** La secuencia se descompone en diferentes niveles jerárquicos.
- **Modelo de Factores Latentes:** Genera embeddings para usuarios e ítems.
- **Procesamiento con Redes Neuronales:** Combina las representaciones jerárquicas utilizando redes neuronales para predecir las preferencias.
- **Generación de Recomendaciones:** A partir de las representaciones combinadas, se generan las recomendaciones.

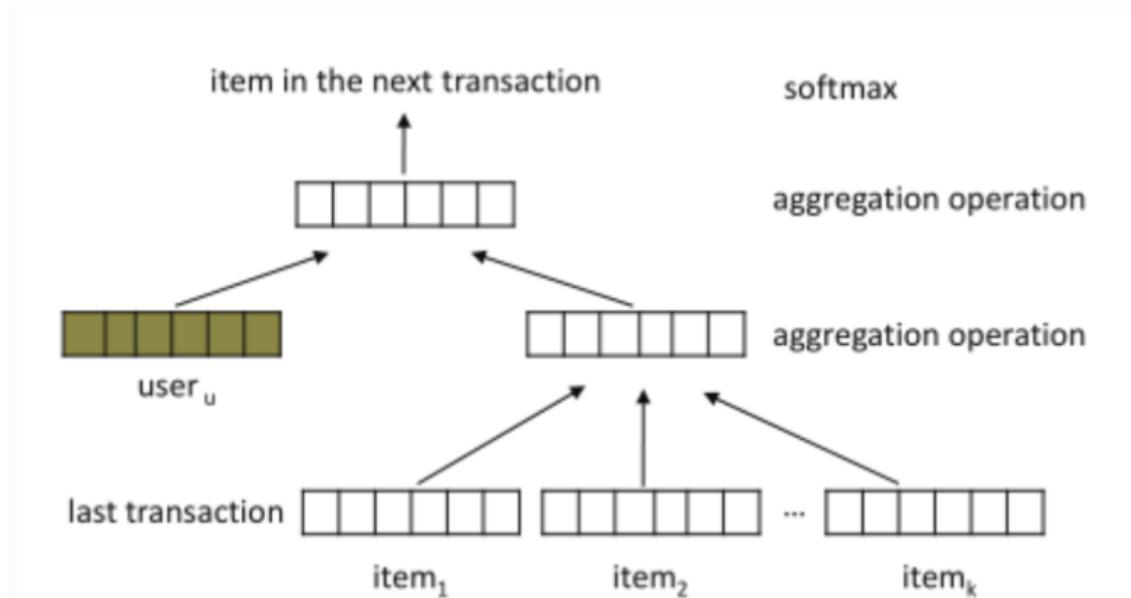


Figura 2.4: Funcionamiento de HRM [18].

2.2.4. SRGNN

SRGNN (Session-based Recommendation with Graph Neural Networks) es un modelo de recomendación que utiliza redes neuronales de grafos (GNN) para capturar las relaciones complejas entre ítems en secuencias de sesión. Este enfoque permite modelar las dependencias no solo secuenciales sino también estructurales en los datos [19].

Componentes Clave:

- **Representación Gráfica de Sesiones:** Cada sesión se representa como un grafo dirigido, donde los nodos son los ítems y las aristas representan la transición entre ítems.
- **Red Neuronal de Grafos (GNN):** Procesa el grafo de la sesión para aprender una representación vectorial de cada nodo (ítem) teniendo en cuenta su contexto en la sesión.
- **Agregación y Predicción:** Las representaciones de los ítems se agregan para predecir el próximo ítem en la sesión.

Parámetros Importantes:

- **embedding_size:** Tamaño del vector de embeddings de los ítems.
- **hidden_size:** Número de unidades en las capas ocultas de la GNN.
- **num_layers (step):** Número de capas en la GNN.
- **dropout_prob:** Tasa de abandono para evitar el sobreajuste.
- **loss_type:** Tipo de función de pérdida ('BPR' o 'CE').

Funcionamiento:

- **Construcción del Grafo:** Cada sesión de usuario se convierte en un grafo dirigido.
- **Propagación en la GNN:** Los nodos del grafo (ítems) se procesan mediante la GNN, que actualiza sus representaciones considerando la estructura del grafo.
- **Agregación de Representaciones:** Las representaciones de los nodos se combinan para obtener una representación global de la sesión.
- **Predicción del Próximo Ítem:** A partir de la representación agregada, se predice el próximo ítem en la sesión.

Este algoritmo puede funcionar especialmente bien para nuestro problema ya que analiza los objetos, la relación entre los ítems y la secuencialidad de las interacciones, por lo que entiende el orden en el que se ha jugado a un juego y encuentra similares, pudiendo proporcionar un enfoque interesante al problema.

2.2.5. BERT4Rec

BERT4Rec (Bidirectional Encoder Representations from Transformers for Sequential Recommendation) es un modelo de recomendación secuencial que utiliza transformadores bidireccionales profundos para modelar las secuencias de comportamiento de los usuarios. A diferencia de los modelos unidireccionales, BERT4Rec captura las dependencias entre ítems en ambas direcciones de la se-

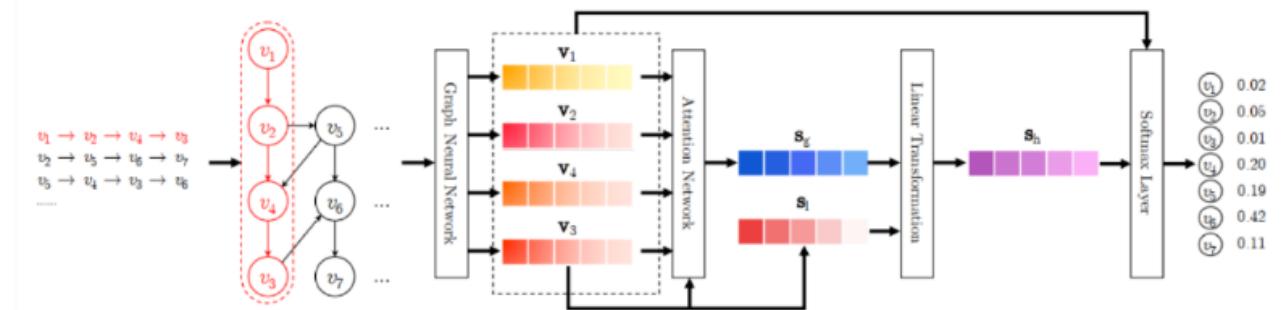


Figura 2.5: Funcionamiento de SRGNN [20].

cuencia, mejorando la precisión de las recomendaciones [21].

Componentes Clave:

- **Autoatención Bidireccional:** Captura dependencias en ambas direcciones de la secuencia.
- **Objetivo Cloze:** Predice ítems enmascarados utilizando el contexto de izquierda y derecha.

Parámetros Importantes:

- **hidden_size:** Tamaño del vector de características (64).
- **inner_size:** Tamaño de la capa intermedia (256).
- **n_layers:** Número de capas de transformadores (2).
- **n_heads:** Número de cabezas de atención (2).
- **hidden_dropout_prob:** Probabilidad de dropout para las capas ocultas (0.5).
- **attn_dropout_prob:** Probabilidad de dropout para la atención (0.5).
- **hidden_act:** Función de activación (gelu).
- **layer_norm_eps:** Epsilon para normalización de capas (1e-12).
- **initializer_range:** Rango del inicializador (0.02).
- **mask_ratio:** Proporción de ítems enmascarados (0.2).
- **loss_type:** Tipo de función de pérdida ('CE' o 'BPR').

Funcionamiento:

- **Input:** Secuencia de interacciones del usuario.
- **Autoatención Bidireccional:** Modela las dependencias bidireccionales de la secuencia.
- **Predicción de ítems enmascarados:** Utiliza el objetivo Cloze para predecir ítems enmascarados en la secuencia.
- **Generación de Recomendaciones:** Predice las preferencias del usuario basándose en las representaciones bidireccionales.

BERT4Rec podría ser un algoritmo interesante para nuestro problema debido a su uso de la moderna tecnología de transformadores bidireccionales, que permite capturar dependencias en ambas direcciones de la secuencia de interacciones del usuario. Esto mejora significativamente la precisión

de las recomendaciones en comparación con los modelos unidireccionales tradicionales. Además, su objetivo Cloze para la predicción de ítems enmascarados utiliza el contexto completo de la secuencia, ofreciendo una comprensión más profunda y personalizada del comportamiento del usuario [22].

2.2.6. HGN

HGN es un modelo de recomendación secuencial que emplea una red jerárquica de compuertas para capturar la dinámica de las secuencias de comportamiento de los usuarios. Este modelo se enfoca en resaltar las interacciones importantes mediante una estructura jerárquica que mejora la representación de las secuencias, proporcionando recomendaciones más precisas y personalizadas [23].

Componentes Clave:

- **Estructura Jerárquica:** Organiza las interacciones de los usuarios en una estructura jerárquica para mejorar la captación de la dinámica secuencial.
- **Red de Compuertas:** Utiliza compuertas para controlar el flujo de información, destacando las interacciones relevantes.

Parámetros Importantes:

- **embedding_size:** Tamaño del vector de características (64).
- **n_layers:** Número de capas de la red jerárquica (3).
- **dropout_prob:** Probabilidad de dropout para las capas ocultas (0.5).
- **activation:** Función de activación (relu).
- **learning_rate:** Tasa de aprendizaje para el entrenamiento del modelo (0.001).
- **weight_decay:** Parámetro de decaimiento de peso para regularización (0.01).
- **max_seq_length:** Longitud máxima de la secuencia de entrada (50).

Funcionamiento:

- **Input:** Secuencia de interacciones del usuario.
- **Estructura Jerárquica:** Organiza las interacciones en diferentes niveles jerárquicos.
- **Red de Compuertas:** Controla el flujo de información para resaltar interacciones importantes.
- **Generación de Recomendaciones:** Utiliza la representación jerárquica para predecir las preferencias del usuario.

Este algoritmo permite mediante su estructura jerárquica resaltar las interacciones más importantes, lo cual con nuestro problema podría ser darles más importancia a las interacciones con más horas de juego ya que debería ser más significativo sobre el gusto de un usuario un juego que tenga muchas horas jugadas que otro que simplemente haya jugado un par de horas. Este enfoque debería permitir darnos recomendaciones más precisas y personalizadas.

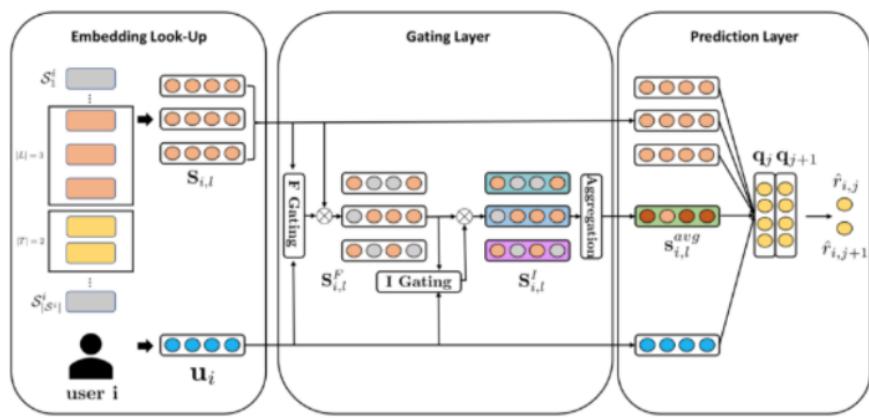


Figura 2.6: Funcionamiento de HGN [24].

2.2.7. SHAN

SHAN (Self-Attentive Sequential Recommendation Model) es un modelo de recomendación secuencial diseñado para capturar las dependencias a largo plazo en las secuencias de interacción de los usuarios. Utiliza una red auto-atentiva para asignar pesos a las interacciones pasadas, permitiendo al modelo enfocarse en los elementos más relevantes para la predicción [25].

Componentes Clave:

- **Auto-Atención:** Captura relaciones a largo plazo en las secuencias de interacción.
- **Embeddings:** Representa a los usuarios y los ítems en un espacio latente.
- **Redes Neuronales:** Procesa las representaciones y aprende las preferencias del usuario.

Parámetros Importantes:

- **embedding_size:** Tamaño del vector de embeddings.
- **hidden_size:** Número de unidades en las capas ocultas de la red neuronal.
- **attention_size:** Tamaño del vector de atención.
- **dropout_prob:** Tasa de abandono para evitar el sobreajuste.
- **loss_type:** Tipo de función de pérdida (por ejemplo, 'BPR' o 'CE').

Funcionamiento:

- **Input:** Secuencia de interacciones del usuario.
- **Embeddings:** Genera representaciones latentes para usuarios e ítems.
- **Mecanismo de Auto-Atención:** Asigna pesos a las interacciones pasadas basados en su relevancia.
- **Red Neuronal:** Procesa las representaciones ponderadas para predecir las preferencias del usuario.
- **Generación de Recomendaciones:** Utiliza las representaciones combinadas para generar las recomendaciones.

Este algoritmo puede resultar efectivo para nuestro problema, ya que contempla tanto las interacciones

ciones a corto y a largo plazo haciendo que el algoritmo contemple tanto los juegos que has jugado recientemente como los juegos que has jugado hace tiempo pero que han tenido un gran impacto en el usuario (muchas horas jugadas).

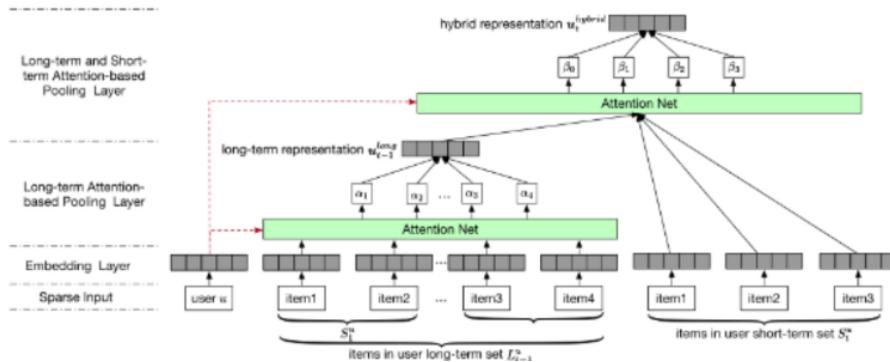


Figura 2.7: Funcionamiento de SHAN [26].

2.2.8. NPE

NPE (Neural Personalized Embedding) es un modelo de recomendación secuencial que emplea una red neuronal para generar embeddings personalizados de usuarios e ítems. Este enfoque permite capturar tanto la información secuencial como las características específicas de los usuarios y los ítems, mejorando la precisión de las recomendaciones [27].

Componentes Clave:

- **Embeddings Personalizados:** Genera representaciones latentes específicas para usuarios e ítems.
- **Redes Neuronales:** Utiliza una red neuronal para procesar las secuencias de interacción y aprender las preferencias del usuario.
- **Atención:** Aplica mecanismos de atención para capturar dependencias importantes en la secuencia de interacciones.

Parámetros Importantes:

- **embedding_size:** Tamaño del vector de embeddings.
- **hidden_size:** Número de unidades en las capas ocultas de la red neuronal.
- **dropout_prob:** Tasa de abandono para evitar el sobreajuste.
- **learning_rate:** Tasa de aprendizaje para la optimización del modelo.
- **loss_type:** Tipo de función de pérdida (por ejemplo, 'BPR' o 'CE').

Funcionamiento:

- **Input:** Secuencia de interacciones del usuario.
- **Embeddings Personalizados:** Genera representaciones latentes para usuarios e ítems.
- **Red Neuronal:** Procesa las secuencias de interacción mediante una red neuronal.

- **Mecanismo de Atención:** Asigna pesos a las interacciones basados en su relevancia.
- **Generación de Recomendaciones:** Predice las preferencias del usuario utilizando las representaciones combinadas.

Neural Personalized Embedding puede funcionar bien para nuestro problema ya que es un algoritmo que brinda una recomendación muy personalizada, sobre todo en problemas con mucha diversidad de ítems como podría ser nuestro caso al contar con más de 30,000 juegos.

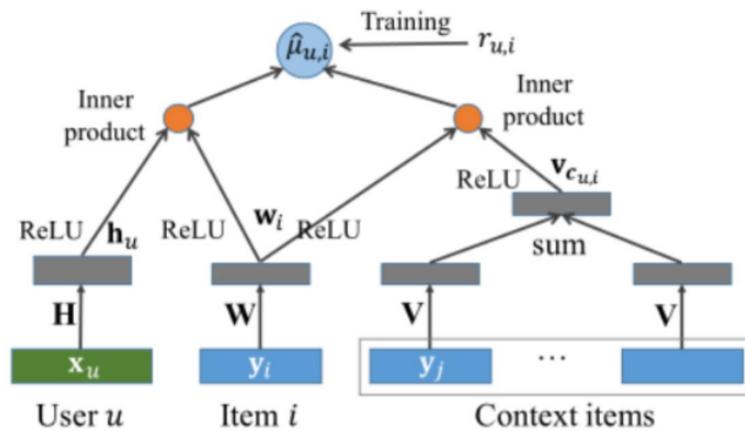


Figura 2.8: Funcionamiento de NPE [28].

2.3. Desarrollo web

En esta sección se aborda la selección del framework web Django como la herramienta principal para la implementación del desarrollo web. Se analizarán las ventajas que ofrece Django en comparación con otros frameworks web como Flask, destacando sus características distintivas y beneficios específicos. Además, se explicará detalladamente el funcionamiento del Object Relational Mapping (ORM) de Django, resaltando cómo facilita la interacción con la base de datos y optimiza el desarrollo del proyecto. Esta discusión proporcionará una comprensión profunda de por qué Django es una opción adecuada y eficiente para este propósito.

2.3.1. La elección de Django

En este proyecto, hemos decidido usar Django, un framework web de Python, utilizado para desarrollar aplicaciones web escalables y seguras, para la implementación del modelo. Aunque es una biblioteca más pesada que otras, como FastAPI o Flask, ofrece más herramientas para futuras mejoras y cambios. Django proporciona una estructura robusta y completa que facilita el desarrollo de aplicaciones complejas, asegurando una base sólida para la evolución del proyecto.

En cuanto a la interfaz gráfica, Django cuenta con numerosas herramientas para el desarrollo de

páginas web, lo que nos permite crear una interfaz atractiva y sencilla para que los usuarios se sientan cómodos al usarla. Utilizando plantillas escritas en HTML y CSS, podemos darle una estructura y un diseño claro a la aplicación. Estas plantillas contienen etiquetas que permiten la inserción dinámica de contenido y son procesadas por Django en tiempo de ejecución para generar páginas dinámicas.

En cuanto a la elección entre Django y Flask, debemos considerar que Flask es un framework minimalista y flexible, ideal para proyectos pequeños y simples. Sin embargo, dado que nuestra intención es que el proyecto evolucione a una mayor escala, Django es una mejor opción debido a que ofrece una estructura más sólida y completa. Otra ventaja importante de Django es su sistema de Object Relational Mapping (ORM), que facilita el desarrollo de aplicaciones con bases de datos grandes y complejas. El ORM permite mapear objetos de un sistema orientado a objetos a tablas de una base de datos relacional, simplificando la gestión y persistencia de datos.

Por todas estas razones, Django se presenta como la opción más adecuada y eficiente para implementar la funcionalidad de recomendación de videojuegos en nuestra aplicación, permitiendo gestionar usuarios y generar rankings personalizados de manera efectiva.

2.3.2. El ORM de Django

El ORM de Django es una de sus características más destacadas, permitiendo a los desarrolladores interactuar con la base de datos relacional mediante objetos Python en lugar de escribir SQL manualmente. En lugar de manejar directamente las operaciones en la base de datos, el ORM de Django permite definir clases de objetos Python que heredan de `django.db.models.Model`, facilitando así la creación y manipulación de tablas sin necesidad de utilizar SQL. Esta abstracción simplifica significativamente el desarrollo y mantenimiento de aplicaciones.

Una de las grandes ventajas del ORM de Django es su compatibilidad con varias bases de datos, incluyendo PostgreSQL, SQLite, Oracle y MySQL, lo que le confiere una gran flexibilidad. Esto permite a los desarrolladores elegir y cambiar de base de datos según las necesidades del proyecto sin grandes complicaciones.

Otro aspecto destacado del ORM de Django es su capacidad para manejar migraciones de forma sencilla. Las migraciones representan cambios en la estructura de la base de datos realizados a través de código en lugar de directamente en la base de datos. Este enfoque permite aplicar cambios de manera segura y controlada, facilitando el mantenimiento y evolución de la base de datos a lo largo del tiempo.

Los modelos en Django son extremadamente flexibles, permitiendo definir una amplia variedad de tipos de campos, como números enteros, cadenas, fechas y valores booleanos, entre otros. Además, el ORM de Django soporta la definición de relaciones entre modelos, lo que permite representar relaciones entre tablas de datos de manera intuitiva. También se pueden establecer restricciones sobre

los datos y definir reglas de validación personalizadas para los campos, asegurando así la integridad y consistencia de los datos almacenados.

El ORM de Django permite realizar consultas eficientes y complejas de manera sencilla. Los desarrolladores pueden filtrar, ordenar y agrupar los datos fácilmente, facilitando la implementación de consultas avanzadas sin necesidad de recurrir a SQL. Esta capacidad de manejar consultas complejas de forma eficiente es especialmente útil en aplicaciones que requieren un manejo intensivo de datos.

En resumen, el ORM de Django ofrece una forma poderosa y flexible de interactuar con bases de datos relacionales, simplificando el desarrollo y mantenimiento de aplicaciones y proporcionando una gran capacidad para manejar datos de manera eficiente y segura.

DISEÑO, ANALISIS E IMPLEMENTACIÓN

En este capítulo se abordará el diseño, análisis e implementación del proyecto. Se describirá el diseño del software central, incluyendo su estructura general y el ciclo de vida elegido. Además, se identificarán los requisitos funcionales y no funcionales del sistema.

3.1. Diseño del problema

En esta sección se mostrará el diseño de la aplicación, describiendo su estructura y los módulos que la integran.

3.1.1. Estructura

El sistema está compuesto por tres módulos principales. El primer módulo se denomina “Preprocesamiento de los datos” y se encarga de procesar los datos de tal forma que sean legibles por los otros módulos, en concreto por el denominado “Modelos de recomendación”, que se encarga de gestionar todos los algoritmos de RecBole y prepararlos para su uso. El tercer módulo, denominado “Aplicación Web”, se utiliza para mostrar los resultados de los algoritmos procesados por el primer módulo. Adicionalmente, cada módulo cuenta con un preprocesamiento de los datos pero que es diferente para cada módulo.

Preprocesamiento de los datos:

En este módulo se llevan a cabo las tareas para procesar los datos para ser usados por los otros dos módulos.

Módulo de recomendación:

En este módulo se realizan todas las tareas de entrenamiento y selección de algoritmos, validación y comparación y pruebas entre ellos. Tras todo el estudio pertinente, este módulo se centrará en hacer las pruebas necesarias para llegar a los valores de hiperparámetros y algoritmos óptimos para nuestro problema y una vez conseguido, se guardan estos objetos mediante la librería “pickle” pudiendo de

esta forma enviar los algoritmos a la aplicación web.

Aplicación web:

Este módulo presenta los resultados y proporciona una interfaz gráfica que permite comparar las diferentes recomendaciones reales para cada usuario, con la opción de seleccionar entre varios algoritmos. Incluye la implementación de Django, con sus vistas, modelos y plantillas.

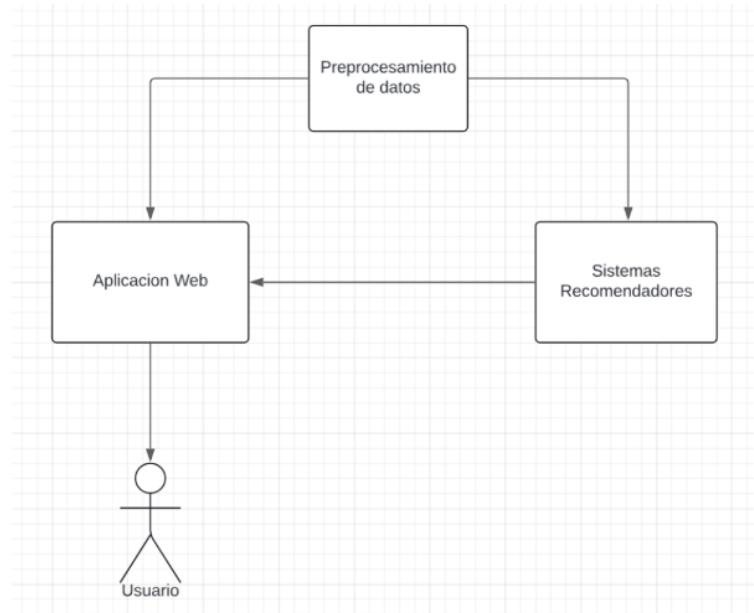


Figura 3.1: Diagrama mostrando la estructura del proyecto.

3.1.2. Análisis

En esta sección, se especificarán los requisitos que el sistema debe cumplir para satisfacer las necesidades de los usuarios. Estos requisitos se han clasificado en dos categorías: funcionales y no funcionales. Además, los requisitos funcionales se han subdividido de acuerdo con los módulos del sistema.

Requisitos funcionales

Preprocesamiento de los datos:

- **RF-1:** El módulo debe ser capaz de descargar los ficheros de las bases de datos de Steam proporcionados por RecBole.
- **RF-2:** El módulo tiene que ser capaz de leer tanto un fichero .item como un fichero .iter con la información de Steam.
- **RF-3:** El módulo tiene que ser capaz de procesar el fichero .item para que sea legible por los algoritmos de recomendación de RecBole.

- **RF-4:** El módulo tiene que ser capaz de procesar la información de los ítems para la aplicación de Django.

Modelos de recomendación:

- **RF-5:** Los modelos de recomendación deben implementar modelos de recomendación secuencial.
- **RF-6:** Los modelos de recomendación deben ser entrenados con bases de datos de videojuegos (Steam) para aprender patrones y preferencias del usuario.
- **RF-7:** Los modelos de recomendación deben cargar la información tanto de las interacciones de Steam (.iter) como de los objetos (.item).

Aplicación web:

- **RF-8:** La aplicación web tendrá una interfaz sencilla y clara, siendo accesible para el usuario promedio.
- **RF-9:** La aplicación web debe permitir al usuario elegir un id de usuario y dos modelos y mostrarle por pantalla los 50 primeros elementos recomendados con cada modelo para poder observar las diferencias reales entre los diferentes modelos.
- **RF-10:** Al mostrar las recomendaciones se tendrá que mostrar por pantalla la información de cada juego como son su precio, título, id y géneros.

Requisitos no funcionales

- **RNF-1:** El sistema debe tratar de ser lo más eficiente a la hora de procesar los datos.
- **RNF-2:** Los modelos de recomendación deberán ser lo más acertados posibles en cuanto a las puntuaciones obtenidas de sus recomendaciones.
- **RNF-3:** Los modelos de recomendación tienen que guardarse una vez entrenados para luego ser llamados desde la aplicación web sin tener que repetir su entrenamiento.
- **RNF-4:** La aplicación web deberá tener una interfaz sencilla y limpia que permita buena accesibilidad al usuario promedio.

3.2. Obtención y preprocesamiento de los datos

Para llevar a cabo la implementación y el estudio necesario en este trabajo, hemos utilizado la base de datos de Steam proporcionada por RecBole. Siguiendo la documentación y las instrucciones, llegamos a una sección de GitHub donde se encuentra el archivo de descarga. En este punto, se nos ofrece la opción de descargar la base de datos en su versión estándar o en la versión “steam-merged”, que es la que hemos elegido. La diferencia principal entre estas dos versiones es que la versión “steam-merged” elimina las interacciones duplicadas. Esto es especialmente relevante dado que, inicialmente, la base de datos contiene casi ocho millones de interacciones, pero con la versión “steam-merged” se reduce a aproximadamente tres millones de interacciones, lo que resulta en tiempos de procesamiento más cortos.

Para el desarrollo del proyecto, se descargó un archivo comprimido que contenía dos ficheros. El primero, con extensión .item, incluye información sobre los ítems (juegos) de Steam, proporcionan-

do datos variados como géneros, precios, títulos, etc. El segundo, con extensión .iter, almacena las interacciones de los usuarios con los juegos, así como las horas jugadas y las marcas de tiempo correspondientes.

Aunque la base de datos fue proporcionada por RecBole, fue necesario realizar modificaciones en estos ficheros para asegurar su funcionamiento correcto con la propia librería. Inicialmente, se presentaron numerosos errores porque las funciones de RecBole podían leer el archivo .iter, pero fallaban al procesar el archivo .item. Esta dificultad se vio agravada por la falta de documentación y ejemplos específicos en RecBole, ya que la mayoría de las consultas documentadas utilizaban otra base de datos ("ml-100k") con parámetros predeterminados.

Una vez configurado correctamente el entorno para ejecutar una base de datos diferente a la pre-terminada, se identificó la necesidad de modificar ciertos aspectos de la base de datos proporcionada por RecBole. Estos ajustes fueron esenciales para garantizar la compatibilidad y el correcto procesamiento de los datos dentro del marco del proyecto.

Para identificar y solucionar los problemas que impedían el correcto funcionamiento del fichero, se analizó su contenido detalladamente. Utilizando Excel, se observó que había varias columnas incompletas y una forma inusual de procesar las comillas en ciertos casos específicos. Para corregir estos errores, se desarrolló un script que reemplazaba las comillas problemáticas y completaba las columnas faltantes, permitiendo así la correcta ejecución del fichero.

Para el módulo de recomendación en la aplicación web, fue necesario desarrollar un script que procesara los datos y los cargara en el sistema de datos de Django. Dado que el formato .item no es típicamente soportado, el script debía extraer manualmente los datos del archivo. Este script se implementó dentro de la carpeta "management/commands" de Django, lo que permitió que se pudiera invocar como un comando propio de Django desde la línea de comandos (CLI).

3.3. Módulo de recomendación

En este apartado se detalla la relación del módulo de recomendación con el resto de la aplicación y su implementación.

El módulo de recomendación actúa como una capa de servicios externos, generando un objeto serializado mediante la librería de Python "pickle", que posteriormente se utiliza para entregar el algoritmo que realizará las recomendaciones en la aplicación.

Para este proceso, se han utilizado diversos archivos en los que se realizaron pruebas hasta identificar los algoritmos óptimos. Estas pruebas se describirán con mayor detalle en el apartado de experimentación.

Para llevar a cabo estas pruebas y ejecutar las funciones de entrenamiento y evaluación de un

recomendador en RecBole, es necesario crear archivos .yaml (o definirlos como objetos dentro de un bloque de código en Python). Estos archivos configuran los parámetros del recomendador, incluyendo la configuración de la base de datos, la ruta de acceso y los parámetros específicos del algoritmo, como el número de épocas y el método de evaluación.

Una vez se ha realizado toda la configuración con solo unas pocas funciones de RecBole podemos iniciar, entrenar y valorar los recomendadores, como veremos en el capítulo de experimentación.

3.4. Aplicación web

Para esta aplicación se ha utilizado Django como framework de desarrollo y una base de datos basada en el dataset de Steam, utilizando solo la información de los ítems (juegos) para la aplicación web. Django sigue un patrón de diseño llamado Model View Template (MVT), que es similar al conocido patrón Model View Controller (MVC), con la diferencia de que la parte del controlador es gestionada por el propio Django.

3.4.1. Arquitectura

Nuestra arquitectura, basada en Django, sigue el modelo MVT (Model View Template), que consta de los siguientes elementos:



Figura 3.2: Diagrama de funcionamiento del patrón MVT [29].

- **Modelo:** El modelo define los datos que se almacenan, representados mediante clases de Python. Cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros y puede incluir métodos. Esto facilita la definición y el control del comportamiento de los datos.
- **Vista:** La vista se implementa mediante funciones en Python. Su objetivo es determinar qué datos serán visualizados y realizar otras tareas como el envío de correos electrónicos, la autenticación con servicios externos y la validación de datos a través de formularios. Gracias al ORM de Django, es posible escribir código Python en lugar de SQL para realizar las consultas necesarias. Es importante destacar que la vista no se encarga de la presentación de los datos, ya que esto es responsabilidad de las plantillas.
- **Plantilla (Template):** La plantilla es esencialmente una página HTML con etiquetas adicionales específicas de Django, y puede generar contenido en varios formatos como HTML, XML, CSS, JavaScript, CSV, entre otros. La plantilla recibe los datos de la vista y los organiza para su presentación en el navegador web.

La base de datos de la aplicación cuenta solo con la información imprescindible para ser mostrada al hacer una recomendación. Estos son el *product_id* para una vez obtenida los scores de los objetos recomendados con mayor puntuación poder encontrar cuál es el juego, su título, géneros y precio. Al tener el modelo entrenado no nos hace falta almacenar más información en la aplicación web.

3.4.2. Funcionalidad

En este apartado se muestra la funcionalidad de la aplicación, así como un diagrama de secuencia que muestra detalladamente el funcionamiento.

Al iniciar la aplicación se muestra un menú principal que nos muestra unos desplegables, en uno se indica el numero usuario (id) para el que queremos simular la recomendación, así como dos desplegables más para elegir los recomendadores. Una vez elegidos los parámetros se pulsará el botón de “Buscar” y nos mostrará los 50 juegos mejor recomendados para el usuario por cada modelo, donde podremos ver si hubiera diferencias entre los juegos que recomienda cada uno.

En la figura 3.4 podemos ver un diagrama de secuencia que muestra el funcionamiento del mismo.

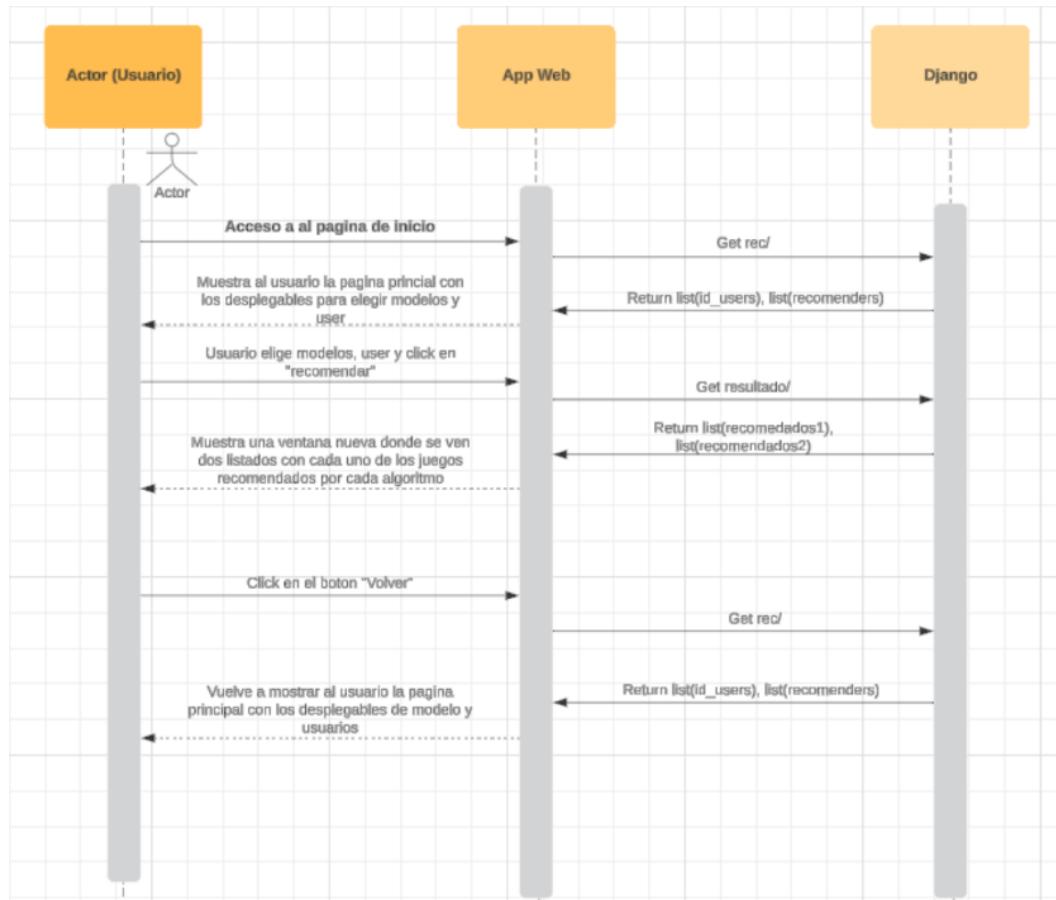


Figura 3.3: Diagrama de secuencia de la aplicación.

PRUEBAS Y RESULTADOS

4.1. Entorno de pruebas

Durante la fase de pruebas y el estudio del proyecto, se evaluaron diversas opciones para la ejecución de los experimentos. Inicialmente, se consideró el uso de un entorno remoto a través de Google Colab. Sin embargo, después de realizar varias pruebas y enfrentar múltiples errores, se determinó que esta opción no era viable. Una de las razones principales fue que RecBole interactúa con la GPU y, debido a la naturaleza y funcionamiento de Colab, estas interacciones no funcionaban correctamente.

Descartada esta vía, se decidió trabajar con una máquina virtual de Ubuntu 18.04. A pesar de que se lograron instalar RecBole y realizar algunas pruebas, se observó que el rendimiento no era óptimo en un entorno Linux. Aunque es posible hacer que funcione, resultó ser más problemático y menos eficiente, esto también se pudo deber a que, al ser una máquina virtual, se cuentan con menos recursos computacionales.

Finalmente, se optó por utilizar el sistema operativo Windows del propio dispositivo, permitiendo así el uso completo de todos los recursos y el potencial computacional disponibles. Para ello, se creó un entorno virtual donde se descargaron todas las librerías necesarias, como se detalla en el apéndice A (fichero requirements.txt).

Después de realizar diversas mediciones de tiempos, se concluyó que el entorno de pruebas planteado era adecuado. Además, al mantener el dispositivo constantemente conectado a la corriente, se pudo utilizar para pruebas de alta demanda computacional, que en algunos casos requirieron entre 2 y 3 días para completarse.

Componente	Características
CPU	AMD Ryzen 5 5600H
GPU	AMD Radeon Graphics 7
RAM	16 GB
S.O.	Windows 11 Home 64 bits
Python Environment	Python 3.9.13

Figura 4.1: Especificación del sistema.

4.2. Primeros resultados y elección de algoritmos

Los primeros pasos para nuestro experimento consisten en revisar la documentación de RecBole y seleccionar los modelos que ofrecen un enfoque más interesante para nuestro estudio. Después de explorar las documentaciones, se realizó una preselección de los algoritmos que se utilizarán en un primer análisis. Estos algoritmos se ejecutarán bajo condiciones justas e igualitarias y, posteriormente, comparando las métricas utilizadas (Recall, MRR, NDCG, Hit y Precisión), evaluaremos estos valores y realizaremos un estudio más exhaustivo de los tres algoritmos con mejores resultados.

Para este estudio previo, los algoritmos elegidos fueron: GRU4Rec, FOSSIL, HRM, SRGNN, BERT4Rec, HGN, SGAN y NPE.

Es importante tener en cuenta que, al tratarse de una base de datos formada por las interacciones de los usuarios que solo contemplan las horas de juego, es una base de datos muy grande y con valores muy variados. Por ello, las diferencias entre los valores de las métricas de los recomendadores pueden parecer pequeñas, pero son relevantes. No es posible alcanzar valores muy altos debido a la naturaleza tanto de los datos como de los recomendadores. Además, el enfoque del experimento es observar cómo se comportan los recomendadores secuenciales con este tipo de datos, considerando que, típicamente, estos recomendadores se aplican a secuencias de clics en páginas web o elecciones de carritos de compra en tiendas en línea.

4.3. Experimentos

Para nuestro primer experimento para ver cuáles de los algoritmos nos proporcionan un mejor rendimiento se decidió trabajar con la configuración por defecto de cada algoritmo, solo estableciendo de forma personalizada la elección del parámetro “epochs” el cual establece el número de iteraciones del algoritmo. Se estableció un valor de 10, ya que estamos trabajando con una base de datos con aproximadamente 2,5 millones de usuarios y 3 millones de interacciones, con una media de 1,2 interacciones por usuario. Aun con este valor bajo de “epochs” los algoritmos tardan varias horas cada uno en ejecutar.

Los modelos en RecBole se configuran mediante archivos conocidos como archivos de configuración (config files) con extensión .yaml, que establecen los parámetros del algoritmo, así como la forma de evaluarlo, entre otros aspectos. Dado que las pruebas se llevaron a cabo utilizando cuadernos de Jupyter, resultó más conveniente declarar estos archivos dentro del propio código en lugar de crearlos externamente. En la Figura 4.2 se muestra la estructura de estos archivos de configuración.

El resto de los parámetros del modelo se mantienen sin especificar, utilizando así sus valores por defecto. Los valores de “user_inter_num_interval” y “item_inter_num_interval” se establecen entre 1 e infinito. Esto se debe a que, tras realizar varias pruebas, se observó que la media

```

parameter_dict = {
    #Configuracion de algoritmo
    'data_path': 'C:/Users/lukit/Desktop/Universidad/TFG/RecPruebas/MyDataset',
    'USER_ID_FIELD': 'user_id',
    'ITEM_ID_FIELD': 'product_id',
    'TIME_FIELD': 'timestamp',
    'user_inter_num_interval': "[1,inf)",
    'item_inter_num_interval': "[1,inf)",

    #Configuracion de los datos a cargar
    'load_col': {'inter': ['user_id', 'product_id', 'timestamp', 'play_hours', 'found_funny'],
                 'item': ['app_name', 'developer', 'product_id', 'genres', 'publisher', 'sentiment', 'specs', 'tags'],
                 'selected_features': ['user_id', 'product_id', 'timestamp', 'play_hours', 'found_funny', 'app_name'],
                 },
    'neg_sampling': None,

    #Configuracion parametros del modelo
    'epochs': 10,
    'train_neg_sample_args': None,

    #Configuracion evaluacion del modelo
    'eval_args': {
        'split': {'RS': [9, 0, 1]},
        'group_by': 'user',
        'order': 'TO',
        'mode': 'full'}
}

```

Figura 4.2: Archivo de configuración.

de interacciones por usuario es de 1.2. Si se aplicaba un límite inferior más alto, la mayoría de las interacciones quedaban fuera del análisis del algoritmo, impidiendo que el recomendador se ajustara correctamente. Por esta razón, estos parámetros podrían eliminarse del archivo de configuración en futuros experimentos.

Una vez establecido el método para las pruebas, el siguiente paso fue crear y ejecutar los archivos pertinentes para observar cuáles de los modelos se adaptaban mejor a nuestro problema. La parte más desafiante es lograr que el primer modelo se ejecute correctamente, ya que suelen surgir numerosos errores de configuración hasta encontrar los parámetros y funciones adecuados. Sin embargo, una vez que se consigue que un modelo funcione, los demás tienden a dar menos problemas, salvo errores puntuales.

En la Figura 4.3 se presenta la secuencia de funciones que se deben ejecutar para poner en marcha los algoritmos y obtener la evaluación. En este experimento, se optó por ejecutar desde Jupyter en lugar de utilizar la interfaz de línea de comandos (CLI). Esta decisión se tomó debido a diversos errores encontrados al intentar usar la CLI de RecBole, lo que llevó a descartar esta opción.

Además, este enfoque nos permite una mayor flexibilidad y control durante el proceso de configuración y ejecución de los algoritmos, facilitando la identificación y resolución de problemas de manera más eficiente. También proporciona un entorno más interactivo para la evaluación de los modelos, lo cual es esencial para un análisis detallado y preciso.

Una vez que se ejecutaron los algoritmos, proceso que tomó varios días, los valores obtenidos de cada algoritmo se recopilaron en un archivo Excel. Posteriormente, se generaron tablas para evaluar

```
config = config(model='NPE', dataset='steam-merged', config_dict=parameter_dict)

# init random seed
init_seed(config['seed'], config['reproducibility'])

# logger initialization
init_logger(config)
logger = getLogger()
# Create handlers
c_handler = logging.StreamHandler()
c_handler.setLevel(logging.INFO)
logger.addHandler(c_handler)

#Dataset creation
dataset = create_dataset(config)
logger.info(dataset)

# dataset splitting
train_data, valid_data, test_data = data_preparation(config, dataset)

# model loading and initialization
model = NPE(config, train_data.dataset).to(config['device'])
logger.info(model)

# trainer loading and initialization
trainer = Trainer(config, model)

# model training
best_valid_score, best_valid_result = trainer.fit(train_data)

#evaluation and result printing
test_result = trainer.evaluate(test_data)
print(test_result)
```

Figura 4.3: Proceso de entrenamiento y valoración de un modelo en RecBole.

visualmente cuáles eran los algoritmos más óptimos.

Como se puede observar en la Figura 4.5, aunque los valores no muestran diferencias muy grandes, es posible identificar los tres algoritmos con los mejores resultados. La Figura 4.4 presenta los valores exactos de los mejores modelos, permitiendo una comparación clara y precisa.

	recall@10	mmr@10	ndc@10	hit@10	precision@10
1º= SRGNN	0.1231	0.0435	0.0618	0.1231	0.0123
2º= HRM	0.1231	0.432	0.0616	0.1231	0.0123
3º= SHAN	0.1185	0.042	0.0597	0.1185	0.0118

Figura 4.4: Mejores valoraciones de los modelos.

En el desarrollo del experimento, se observó que los modelos FOSSIL y BERT4Rec obtuvieron los peores resultados. FOSSIL, que utiliza cadenas de Márkov para analizar tanto la similitud global entre ítems como la secuencialidad con el último objeto, mostró un rendimiento inferior. Este comportamiento puede explicarse por la baja media de interacciones por usuario, que es de 1.2. En un entorno con tan pocas interacciones, FOSSIL no encuentra suficientes objetos anteriores para realizar predicciones precisas, lo que conduce a un desajuste del modelo.

Por otro lado, BERT4Rec, que implementa tecnología de transformadores, también mostró resultados desfavorables. Este modelo está diseñado para captar relaciones complejas en secuencias largas mediante un mecanismo de atención. Sin embargo, debido a la naturaleza dispersa de la base de da-

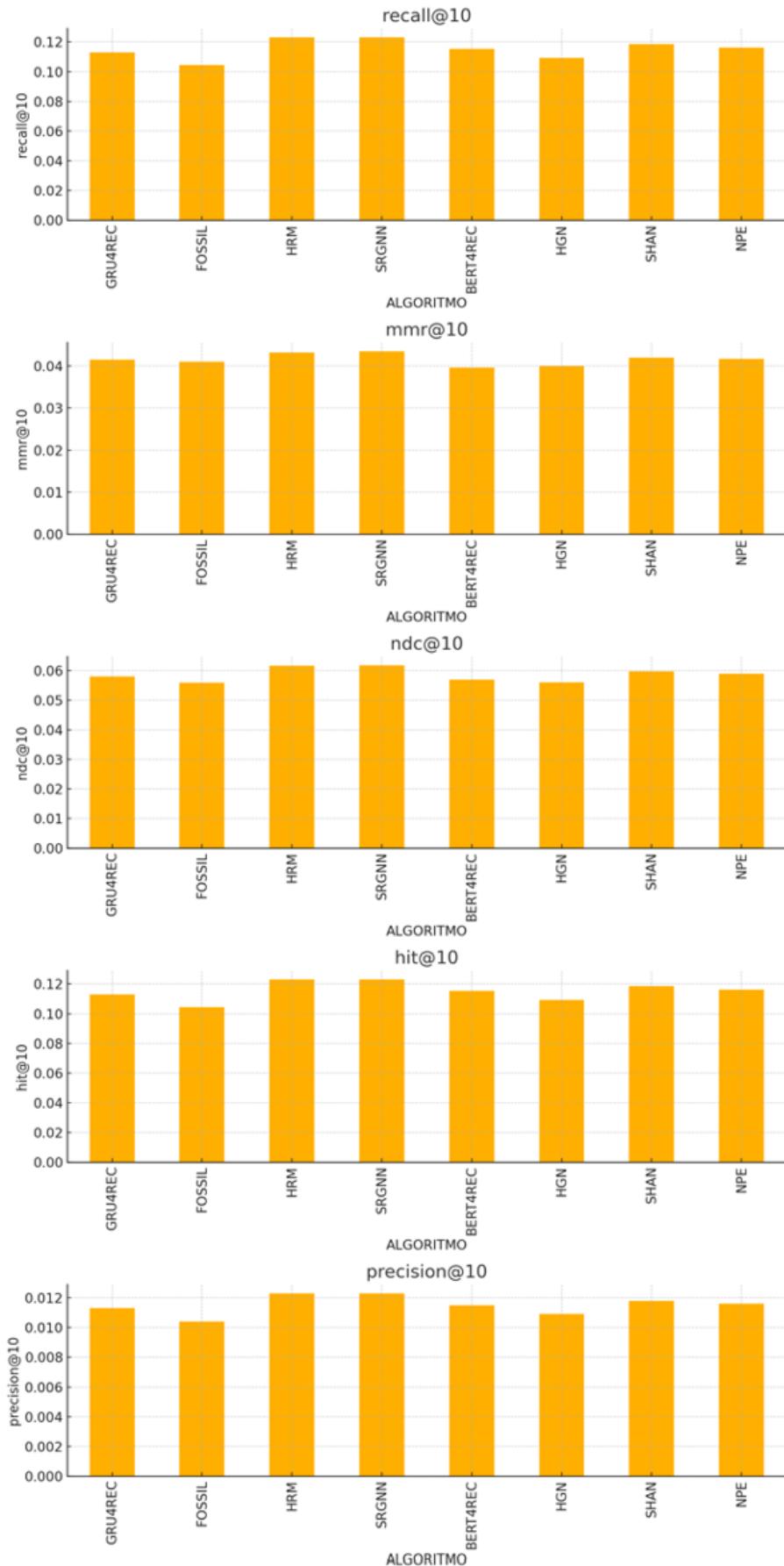


Figura 4.5: Resultados y valoración de los modelos.

tos y la escasa cantidad de interacciones por usuario, BERT4Rec no puede aprovechar su arquitectura basada en atención, que requiere más información y una menor dispersión de datos para funcionar correctamente.

En cuanto a los modelos con mejores resultados, se discutirán las razones de su buen rendimiento en secciones posteriores. Estos análisis subrayan la importancia de la adecuación del modelo al tipo de datos disponibles, demostrando cómo las características específicas de cada modelo pueden influir significativamente en su desempeño.

Finalmente, antes de empezar con los experimentos de los mejores algoritmos en profundidad, falta encontrar cuál era realmente el valor óptimo del parámetro “epochs” ya que al ser tan solo tres algoritmos no habría tanto problema con los largos tiempos de ejecución. Para ello se realizaron pruebas sobre las valoraciones del mejor algoritmo (SRGNN) con diversos valores para el parámetro.

Inicialmente, se probó con 100 iteraciones esperando que hubiera una mejoría notable en los resultados. Sin embargo, como ya hemos comentado debido a la poca diversidad de los datos, esto provocó un fuerte sobreajuste, obteniendo unos resultados mucho peores de los ya obtenidos con diez iteraciones, además de tardar más de ocho horas en terminar la ejecución, ver Figura 4.6.

Métrica	Valor
recall@10	0.0882
mrr@10	0.0321
ndcg@10	0.045
hit@10	0.0882
precision@10	0.0088

Figura 4.6: Resultados del mejor algoritmo (SRGNN) con 100 iteraciones.

Tras esto se hicieron varias pruebas, para elegir el número de iteraciones finales se hicieron varias ejecuciones y se hizo la media entre ellas para saber qué valor obtenía valores más altos. La gráfica que se observa en la Figura 4.7, muestra la comparación entre los valores de recall@10 en distintas ejecuciones, llegando hasta la iteración número 15 porque a partir de este valor el algoritmo tiende a sobreajustar, bajando considerablemente sus valores de la métrica en la mayoría de ejecuciones.

Con los resultados obtenidos, se decidió elegir el valor “8” para el número de iteraciones. Aunque se observó que estos valores pueden variar ligeramente en cada ejecución, se concluyó que el valor óptimo se encuentra entre 8 y 10 iteraciones. Sin embargo, se prefirió el valor 8 porque, para las siguientes pruebas, se iban a realizar muchas ejecuciones, y esta elección podría representar una diferencia significativa en el tiempo total de procesamiento.

Una vez decidido el número de épocas, se procedió a realizar las pruebas con los mejores algoritmos seleccionados. Estas pruebas consistieron en un estudio de hiperparámetros con el objetivo de obtener los mejores resultados posibles. El propósito era identificar cómo optimizar el rendimiento al trabajar con una base de datos tan dispersa como la del problema planteado. Este análisis nos permi-

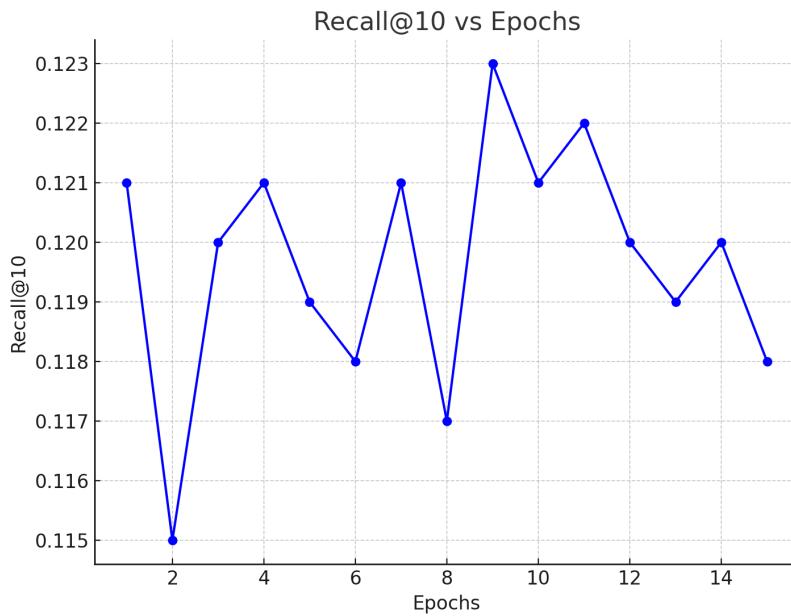


Figura 4.7: Evolución por época de la media de los valores de recall@10 hasta 15 iteraciones.

tirá concluir sobre las mejores prácticas y estrategias para manejar eficazmente bases de datos con características similares y recomendadores secuenciales.

4.3.1. SRGNN

Comenzamos con SRGNN, el cual obtuvo las mejores puntuaciones. Inicialmente, se intentó realizar el estudio de hiperparámetros utilizando una herramienta proporcionada por RecBole denominada *HyperTuning*. Sin embargo, debido a la ausencia de ejemplos reales de personas utilizando esta herramienta y la escasa documentación proporcionada por RecBole, se terminó por descartarla. Además, se observó que *HyperTuning* ofrecía un rendimiento limitado y, en ocasiones, generaba errores de código que no podían ser manejados por el usuario, sino que provenían de la propia librería.

Una vez descartada esta opción, se optó por crear una función que permitiera generar archivos de configuración mediante argumentos, facilitando así la automatización de las pruebas de hiperparámetros. A continuación, se seleccionaron los hiperparámetros a testear. Se decidió probar la tasa de aprendizaje (`learning_rate`) y el tamaño del *embedding* (`embedding_size`), parámetros comunes a todos los algoritmos. Además, se eligió el parámetro más significativo dentro de cada algoritmo.

Para SRGNN, se optó por testar el parámetro “step”, que mide el número de capas dentro de la GNN, con un valor por defecto de “1”.

Los valores experimentados fueron: para “embedding size”, 32, 64 y 128; para “learning rate”, 0.01, 0.001 y 0.0001; y para “step”, 1, 2, 3 y 4. Esto resultó en un total de 36 combinaciones de

parámetros. Es importante destacar que algunas pruebas tuvieron que repetirse debido a que, en ocasiones, el ordenador sufrió el conocido “pantallazo azul” de Windows, formalmente denominado “Error de detención” (en inglés, “Stop Error” o “Blue Screen of Death (BSOD)”). Esto ocurrió debido a la alta demanda de recursos computacionales de las pruebas, lo que interrumpió el proceso después de aproximadamente 17 horas de ejecución, obligando a repetirlas.

Una vez el algoritmo funcionó correctamente se obtuvo una tabla donde se comparan todos los valores y se podían ver cuáles tenían mejor desempeño. La prueba tardó aproximadamente 3700 minutos (62 horas). En la Figura 4.8 se puede observar la tabla con unos colores simulando un ranking de medallas que muestran los mejores valores de ejecución con los hiperparámetros.

embedding_size	learning_rate	step	recall@10	mrr@10	ndcg@10	hit@10	precision@10
32	0.01	1	0.1142	0.042	0.0587	0.1142	0.0114
32	0.01	2	0.1165	0.0426	0.0596	0.1165	0.0116
32	0.01	3	0.116	0.0433	0.0601	0.116	0.0116
32	0.01	4	0.1179	0.044	0.0611	0.1179	0.0118
32	0.001	1	0.1211	0.0436	0.0615	0.1211	0.0121
32	0.001	2	0.1219	0.0434	0.0615	0.1219	0.0122
32	0.001	3	0.1219	0.0434	0.0615	0.1219	0.0122
32	0.001	4	0.1211	0.0433	0.0612	0.1211	0.0121
32	0.0001	1	0.1148	0.0418	0.0588	0.1148	0.0115
32	0.0001	2	0.1156	0.0417	0.0589	0.1156	0.0116
32	0.0001	3	0.1177	0.0419	0.0595	0.1177	0.0118
32	0.0001	4	0.1177	0.0419	0.0594	0.1177	0.0118
64	0.01	1	0.1068	0.0389	0.0545	0.1068	0.0107
64	0.01	2	0.1075	0.0394	0.0551	0.1075	0.0108
64	0.01	3	0.1093	0.0401	0.0561	0.1093	0.0109
64	0.01	4	0.1092	0.0405	0.0564	0.1092	0.0109
64	0.001	1	0.1234	0.044	0.0623	0.1234	0.0123
64	0.001	2	0.1233	0.044	0.0623	0.1233	0.0123
64	0.001	3	0.1225	0.0439	0.0621	0.1225	0.0123
64	0.001	4	0.1247	0.0444	0.0629	0.1247	0.0125
64	0.0001	1	0.1171	0.0419	0.0593	0.1171	0.0117
64	0.0001	2	0.1175	0.0421	0.0596	0.1175	0.0118
64	0.0001	3	0.1175	0.0421	0.0596	0.1175	0.0118
64	0.0001	4	0.1185	0.0422	0.0599	0.1185	0.0118
128	0.01	1	0.0943	0.0345	0.0483	0.0943	0.0094
128	0.01	2	0.1014	0.0369	0.0518	0.1014	0.0101
128	0.01	3	0.1053	0.0387	0.054	0.1053	0.0105
128	0.01	4	0.098	0.0349	0.0494	0.098	0.0098
128	0.001	1	0.1254	0.0452	0.0638	0.1254	0.0125
128	0.001	2	0.121	0.043	0.061	0.121	0.0121
128	0.001	3	0.121	0.0433	0.0612	0.121	0.0121
128	0.001	4	0.1198	0.0434	0.0611	0.1198	0.012
128	0.0001	1	0.1176	0.0421	0.0597	0.1176	0.0118
128	0.0001	2	0.1168	0.042	0.0594	0.1168	0.0117
128	0.0001	3	0.1156	0.0418	0.0589	0.1156	0.0116

Figura 4.8: Resultados del estudio sobre hiperparámetros de SRGNN.

1.– **Embedding_size:** Un tamaño grande de *embedding* tiende a proporcionar mejores resultados que los tamaños más pequeños. Podemos ver cómo el *embedding* de 32 no tiene ningún resultado que sea muy alto, y esto se puede deber a que con la base de datos tan dispersa como la tenemos cuanta más información pueda obtener de las interacciones mejor ya que puede encontrar mejores relaciones entre los ítems y las preferencias de los usuarios.

2.– **Learning_rate:** La tasa de aprendizaje más efectiva resulta ser 0.001. Esto se puede deber a que con una tasa de aprendizaje grande como es 0.1 el modelo converja rápidamente en un óptimo local y deje de explorar mejores soluciones en el espacio de parámetros, y una tasa de aprendizaje demasiado pequeña puede llevar a una convergencia muy lenta que no permita llegar a una solución óptima en las iteraciones que se tienen. Por ello 0.001 proporciona el mejor equilibrio entre velocidad de convergencia y capacidad de exploración.

3.– Step: El valor óptimo para `step` es 1, el valor por defecto. Podemos ver cómo generalmente el valor de `step` 1 suele dar mejores resultados. Esto se puede deber a que al tener una base de datos tan dispersa, el añadir más capas pueden sobrevalorar la información y que a partir de una capa tiendan a añadir ruido innecesario o sobreajuste, aunque haya casos en los que muestre un buen desempeño como es el caso del segundo mejor caso que tiene un valor de `step` de 4.

4.3.2. HRM

Con HRM se sigue un procedimiento similar al anterior simplemente había que elegir cuál es el hiperparámetro más relevante dentro del algoritmo. Tras estudiar la documentación de RecBole y hacer unos testeos se llegó a la conclusión de que la mejor opción era estudiar el parámetro `high_order`. Este parámetro no solo influye en la capacidad del modelo para capturar dependencias de alto orden y adaptarse a datos dispersos, sino que también afecta el equilibrio entre complejidad y generalización y la efectividad de las representaciones jerárquicas. Por estas razones, `high_order` es un hiperparámetro clave que merece una atención particular en el proceso de ajuste de hiperparámetros para HRM.

embedding_size	learning_rate	high_order	recall@10	mrr@10	ndcg@10	hit@10	precision@10
32	0.0100	1	0.0781	0.0270	0.0388	0.0781	0.0078
32	0.0100	2	0.1048	0.0375	0.0531	0.1048	0.0105
32	0.0100	3	0.1092	0.0394	0.0555	0.1092	0.0109
32	0.0100	4	0.1101	0.0398	0.0560	0.1101	0.0110
32	0.0010	1	0.1174	0.0419	0.0593	0.1174	0.0117
32	0.0010	2	0.1173	0.0430	0.0603	0.1173	0.0117
32	0.0010	3	0.1174	0.0431	0.0604	0.1174	0.0117
32	0.0010	4	0.1174	0.0431	0.0604	0.1174	0.0117
32	0.0001	1	0.1093	0.0417	0.0574	0.1093	0.0109
32	0.0001	2	0.1114	0.0430	0.0590	0.1114	0.0111
32	0.0001	3	0.1116	0.0421	0.0583	0.1116	0.0112
32	0.0001	4	0.1049	0.0404	0.0555	0.1049	0.0105
64	0.0100	1	0.0560	0.0188	0.0274	0.0560	0.0056
64	0.0100	2	0.0785	0.0256	0.0377	0.0785	0.0079
64	0.0100	3	0.0856	0.0287	0.0418	0.0856	0.0086
64	0.0100	4	0.0888	0.0299	0.0434	0.0888	0.0089
64	0.0010	1	0.1198	0.0425	0.0604	0.1198	0.0120
64	0.0010	2	0.1238	0.0437	0.0622	0.1238	0.0124
64	0.0010	3	0.1237	0.0436	0.0621	0.1237	0.0124
64	0.0010	4	0.1246	0.0438	0.0625	0.1246	0.0125
64	0.0001	1	0.1201	0.0409	0.0591	0.1201	0.0120
64	0.0001	2	0.1226	0.0417	0.0604	0.1226	0.0123
64	0.0001	3	0.1188	0.0423	0.0601	0.1188	0.0119
64	0.0001	4	0.1188	0.0423	0.0601	0.1188	0.0119
128	0.0100	1	0.0489	0.0160	0.0235	0.0489	0.0049
128	0.0100	2	0.0628	0.0198	0.0297	0.0628	0.0063
128	0.0100	3	0.0672	0.0211	0.0317	0.0672	0.0067
128	0.0100	4	0.0693	0.0217	0.0327	0.0693	0.0069
128	0.0010	1	0.1204	0.0427	0.0606	0.1204	0.0120
128	0.0010	2	0.1230	0.0436	0.0620	0.1230	0.0123
128	0.0010	3	0.1226	0.0436	0.0618	0.1226	0.0123
128	0.0010	4	0.1229	0.0436	0.0619	0.1229	0.0123
128	0.0001	1	0.1233	0.0440	0.0623	0.1233	0.0123
128	0.0001	2	0.1234	0.0447	0.0629	0.1234	0.0123
128	0.0001	3	0.1234	0.0447	0.0629	0.1234	0.0123
128	0.0001	4	0.1234	0.0447	0.0629	0.1234	0.0123

Figura 4.9: Resultados del estudio sobre hiperparámetros de HRM.

Para la ejecución de las pruebas, se añadió un parámetro adicional al fichero de configuración: `reproducibility=False`. Este ajuste permite aumentar significativamente la velocidad de entrenamiento

del algoritmo. Se continuó probando con los mismos valores para los parámetros `embedding_size` y `learning_rate`. Además, para el parámetro `high_order`, se utilizaron los mismos valores que para `step` de SRGNN, ya que este tiene un valor por defecto de 2.

Destacar que gracias al parámetro `reproducibility=False` esta prueba tardó 1800 minutos (30 horas), reduciendo el tiempo casi a la mitad.

1.– `Embedding_size`: En este caso podemos ver cómo el tamaño óptimo ha sido el de 64. Sin embargo, el de 128 también ha funcionado bien, pero con el algoritmo HRM y su funcionamiento ha demostrado que 64 es más que suficiente y que más información de `embedding` tiende a añadir ruido al algoritmo. El de 32 se ve cómo es notablemente insuficiente y nos devuelve los resultados más pequeños de todos.

2.– `Learning_rate`: Al igual que antes, la tasa de aprendizaje más efectiva resultó ser 0.001. Se entiende que se puede aplicar el mismo razonamiento que con el modelo anterior.

3.– `High_order`: Se puede observar cómo el peor valor con diferencia para este parámetro es el 1. Esto se debe a que al elegir como parámetro el 1 es similar a ignorar parte del algoritmo que revisa los elementos anteriores ya que, si solo se escoge valor 1, solo se observa el último elemento de cada usuario. Luego entre los otros valores difiere un poco más de la ejecución. Si bien el valor más alto se obtiene con valor 4, depende mucho de cada caso y si nos fijamos en otros valores que también funcionen relativamente bien podemos ver cómo el valor 2 obtiene muchas veces valores más altos que el 3 y el 4.

4.3.3. SHAN

Para el algoritmo SHAN, se tendría que seguir el mismo procedimiento. Para ello se tiene que elegir primeramente cuál va a ser el hiperparámetro a probar además de `embedding_size` y `learning_rate`. Si exploramos su documentación en RecBole encontramos un parámetro denominado `short_item_length`. Este parámetro se encarga de elegir la longitud de la ventana de historial del usuario que se tiene en cuenta a la hora de hacer una predicción. Esto para un parámetro con el funcionamiento que tiene SHAN debido al mecanismo de atención que aplica este parámetro influye directamente en el funcionamiento y en los pesos que le asigna para cada interacción. También es muy importante por la diversidad de la base de datos de Steam, puede hacer que el algoritmo se sobrecargue mucho con ruido o que realmente encuentre patrones e información relevante entre los datos.

El algoritmo SHAN además también cuenta con el parámetro `reproducibility=False` el cual permitió mejorías interesantes en cuanto al tiempo de ejecución. Para los valores a testear se utilizarán los mismos valores, pero aplicados al parámetro seleccionado.

Antes de mostrar los resultados destacar que esta, al ser la última prueba que se ejecutó, empezó a dar problemas y quedarse a medias cuando se trataba de ejecutar. Tras analizar a qué se debían estos fallos, se observó que la razón era la falta de espacio en el disco duro, sin embargo, antes de comenzar esta prueba se contaba realmente con bastante espacio. Tras hacer una búsqueda de la causa de esto, se encontró que, en la carpeta de trabajo, RecBole crea un fichero de guardado del algoritmo que permite más adelante cargar un modelo ya entrenado y tras haber hecho tantas

ejecuciones la carpeta con los guardados de los 3 modelos seleccionados ocupaba un total de 115 GB, por lo que es importante estar atento de este detalle si se va a trabajar con RecBole.

Esta prueba tardó un total de 2400 minutos (40 horas) en ejecutarse aún con el parámetro de `reproducibility=False`, por lo que se puede ver que SHAN tiene un procedimiento más costoso que HRM. En la Figura 4.10 se pueden observar los resultados.

embedding_size	learning_rate	short_item_length	recall@10	mrr@10	ndcg@10	hit@10	precision@10
32	0.0100	1	0.1175	0.0416	0.0591	0.1175	0.0117
32	0.0100	2	0.1178	0.0420	0.0595	0.1178	0.0118
32	0.0100	3	0.1177	0.0422	0.0596	0.1177	0.0118
32	0.0100	4	0.1184	0.0424	0.0599	0.1184	0.0118
32	0.0010	1	0.1168	0.0424	0.0597	0.1168	0.0117
32	0.0010	2	0.1171	0.0426	0.0599	0.1171	0.0117
32	0.0010	3	0.1171	0.0427	0.0600	0.1171	0.0117
32	0.0010	4	0.1194	0.0428	0.0606	0.1194	0.0119
32	0.0001	1	0.1089	0.0399	0.0559	0.1089	0.0109
32	0.0001	2	0.1093	0.0406	0.0566	0.1093	0.0109
32	0.0001	3	0.1090	0.0400	0.0560	0.1090	0.0109
32	0.0001	4	0.1090	0.0400	0.0561	0.1090	0.0109
64	0.0100	1	0.1127	0.0383	0.0554	0.1127	0.0113
64	0.0100	2	0.1142	0.0398	0.0570	0.1142	0.0114
64	0.0100	3	0.1148	0.0399	0.0572	0.1148	0.0115
64	0.0100	4	0.1156	0.0398	0.0573	0.1156	0.0116
64	0.0010	1	0.1205	0.0428	0.0607	0.1205	0.0120
64	0.0010	2	0.1205	0.0428	0.0608	0.1205	0.0120
64	0.0010	3	0.1204	0.0427	0.0607	0.1204	0.0120
64	0.0010	4	0.1214	0.0428	0.0610	0.1214	0.0121
64	0.0001	1	0.1182	0.0423	0.0599	0.1182	0.0118
64	0.0001	2	0.1183	0.0424	0.0600	0.1183	0.0118
64	0.0001	3	0.1183	0.0424	0.0600	0.1183	0.0118
64	0.0001	4	0.1194	0.0428	0.0605	0.1194	0.0119
128	0.0100	1	0.1096	0.0367	0.0534	0.1096	0.0110
128	0.0100	2	0.1124	0.0387	0.0557	0.1124	0.0112
128	0.0100	3	0.1133	0.0389	0.0561	0.1133	0.0113
128	0.0100	4	0.1130	0.0387	0.0558	0.1130	0.0113
128	0.0010	1	0.1212	0.0427	0.0608	0.1212	0.0121
128	0.0010	2	0.1197	0.0426	0.0604	0.1197	0.0120
128	0.0010	3	0.1193	0.0424	0.0602	0.1193	0.0119
128	0.0010	4	0.1199	0.0425	0.0604	0.1199	0.0120
128	0.0001	1	0.1227	0.0433	0.0616	0.1227	0.0123
128	0.0001	2	0.1227	0.0434	0.0618	0.1227	0.0123
128	0.0001	3	0.1228	0.0433	0.0617	0.1228	0.0123
128	0.0001	4	0.1228	0.0432	0.0616	0.1228	0.0123

Figura 4.10: Resultados del estudio sobre hiperparámetros de SHAN.

- 1.- **Embedding_size:** En cuanto a este parámetro podemos ver que para el funcionamiento de SHAN sí que funciona perfectamente un tamaño de 128 sin generar ruido adicional, y que así se permite al algoritmo el encontrar más patrones e información relevante, sobre todo sabiendo lo dispersa que es la base de datos.
- 2.- **Learning_rate:** Podemos ver que para el algoritmo SHAN, el mejor valor es el 0.0001. Esto significa que, para el funcionamiento de este modelo, una tasa de aprendizaje tan pequeña no lleva a un mínimo relativo y sí permite profundizar más en el descenso de gradiente. Podemos ver cómo 0.001 también obtiene buenos resultados relativos, pero se queda más en un término medio, y que 0.01 es muy grande y obtiene las puntuaciones más bajas de todas.
- 3.- **Short_item_length:** Con este parámetro podemos ver que los valores que se obtienen son algo impredecibles y que en algunos casos depende de la ejecución, ya que podemos ver casos donde un valor de 1 devuelve el valor más alto de entre los otros casos que comparten valores de los primeros dos hiperparámetros. Sin embargo, podemos ver como norma general que los valores 2, 3 y 4 producen los mejores resultados y pueden captar contexto suficiente sin añadir mucho ruido al entrenamiento del modelo. Esto lo podemos ver en los mejores resultados donde el primero es el que tiene valor 3, el segundo el que tiene valor 2 y el tercero el que tiene valor 4.

4.4. Análisis de los resultados

Una vez realizados los experimentos de cada recomendador podemos concluir que el desempeño de los modelos secuenciales se debe mucho tanto a los parámetros elegidos como a la base de datos utilizada. Esto quiere decir que los mejores modelos son los mejores para este problema en específico, donde estamos trabajando con una base de datos muy dispersa en la cual es muy difícil llegar a recomendaciones que obtengan valoraciones muy altas.

Sabiendo esto, estos modelos han quedado en las mejores posiciones debido al funcionamiento y los principios de los mismos. SRGNN es un algoritmo que utiliza redes neuronales de grafo para captar las relaciones entre los ítems dentro de una sesión de usuario. Estas muchas veces son sesiones breves donde no hay muchas interacciones por lo que se acostumbra a encontrar patrones en secuencias cortas. Además, al modelar las interacciones en forma de grafos, SRGNN puede descubrir relaciones implícitas entre los ítems que pueden no ser tan evidentes en secuencias lineales, lo que mejora los resultados con bases de datos de mucha dispersión.

En cuanto HRM, este modelo está planteado con una estructura jerárquica que permite la captura tanto de un comportamiento secuencial como de gustos generales de los usuarios mediante representaciones jerárquicas. Esto hace que este modelo se adapte aun teniendo datos limitados. Además, HRM permite diferentes operaciones de agregación, lo que facilita el ajuste del modelo a las características específicas del dataset y mejora el rendimiento incluso con pocas interacciones por usuario.

Finalmente, SHAN es un modelo que cuenta con un mecanismo de atención que permite al modelo enfocarse en las interacciones más relevantes y recientes del usuario. Esto hace que, aunque haya mas interacciones, SHAN tenga tendencia a darle importancia solo a unas y desprecia las demás, por lo que si hay pocas interacciones les prestará atención a esas y debería desenvolverse correctamente, es por esto que SHAN es un modelo con alta adaptabilidad a las bases de datos.

Para finalizar, como se puede observar en la Figura 4.11 estos algoritmos han alcanzado valores muy similares, pero con diferencias que, aunque parezcan ínfimas deja como el mejor algoritmo para nuestro problema al algoritmo SRGNN.

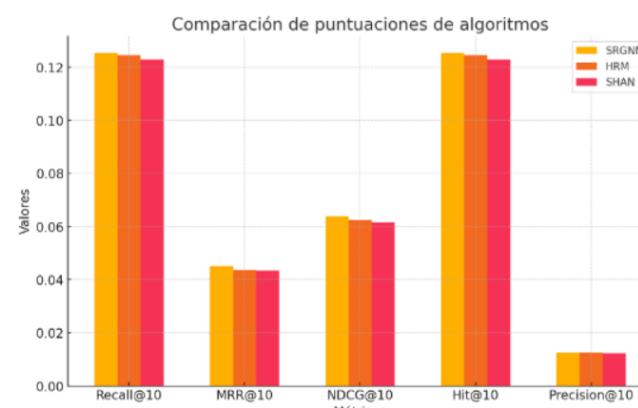


Figura 4.11: Resultados finales sobre las valoraciones de los modelos.

CONCLUSIÓN Y TRABAJO FUTURO

5.1. Conclusión

Actualmente, los sistemas de recomendación son una tecnología cada vez más relevante, haciendo que la experiencia en la web sea cada vez más personalizada y orientada a cada cliente en específico. Esto está afectando tanto a consumidores como a negocios, con un impacto positivo en las ventas y la satisfacción de los clientes. Dentro de este mundo creciente, hay diversas técnicas y cada día se aprende más sobre su funcionamiento y cómo obtener las mejores recomendaciones.

Durante este proyecto, se ha estudiado un nuevo enfoque para la recomendación: el uso de datos y algoritmos secuenciales, diseñados para problemas de secuencias de clics y la predicción de la disposición de ítems en las páginas web, brindando así los ítems con mayor probabilidad de ser comprados por el cliente. Este enfoque ha sido aplicado al ámbito de los videojuegos utilizando la base de datos de Steam, orientándonos hacia un problema con recomendadores secuenciales para bases de datos muy dispersas.

Este proyecto también ha permitido explorar la librería Recbole, una prometedora herramienta para recomendadores que presenta falta de documentación en muchos casos, lo cual ha sido uno de los principales obstáculos para la realización de este proyecto. Se han probado diversos modelos y se ha observado cuáles proporcionan un buen desempeño para este tipo de datos y cuáles, al tener una base de datos tan dispersa, simplemente generaban ruido y no alcanzaban resultados prometedores.

Es importante destacar la necesidad de realizar constantes pruebas y ajustar tanto los datos como los modelos para lograr buenos resultados, lo cual demuestra por qué estos procesos toman tanto tiempo en llevarse a cabo.

En conclusión, los resultados obtenidos indican que los recomendadores secuenciales han demostrado trabajar a un nivel aceptable con bases de datos tan dispersas y grandes, especialmente los modelos que no requieren muchas interacciones diferentes para obtener mejores predicciones. En este contexto, el modelo SRGNN destacó como el más eficaz, mientras que otros modelos más orientados a predicciones de largas secuencias de clics no se adaptaron bien a nuestro problema específico.

5.2. Trabajo futuro

Durante el desarrollo de este proyecto, se han identificado varias áreas y posibles mejoras que podrían realizarse en futuras expansiones de este TFG. Sin embargo, no fue posible explorarlas en profundidad debido a las limitaciones de recursos y tiempo.

Una posible mejora sería explorar otros tipos de recomendadores, lo que permitiría identificar cuáles de ellos se adaptan mejor a las características de nuestro problema. Además, implementarlos en la aplicación permitiría realizar comparativas reales con las recomendaciones brindadas por cada modelo.

Otra rama explorable para la ampliación del proyecto sería realizar un estudio de hiperparámetros para cada modelo, ya que es posible que algunos de los que no mostraron los mejores resultados puedan mejorar significativamente con una buena configuración de los hiperparámetros diferente a la por defecto.

Finalmente, también se podría plantear una migración a la nube del sistema completo. Sin embargo, esto tendría un coste importante debido a la gran carga de procesamiento requerida para entrenar los modelos. A cambio, nos permitiría aprovechar las ventajas de la nube para la distribución de esta herramienta como producto, facilitando las pruebas sobre recomendadores.

BIBLIOGRAFÍA

- [1] Newzoo, "Global games market report 2023," 2024. Accessed: 2024-07-09.
- [2] G. V. Research, "Video game market size, share and growth report, 2030," 2023. Accessed: 2024-07-09.
- [3] Infobae, "Más de 14,000 juegos llegaron a steam en 2023: ¿explosión creativa o problema a futuro?," 2024. Accessed: 2024-07-09.
- [4] F. Ricci, L. Rokach, and B. Shapira, eds., *Recommender Systems Handbook*. Springer, 2015. Link a Springer.
- [5] Grapheverywhere, "Sistemas de recomendación | qué son, tipos y ejemplos." <https://www.grapheverywhere.com/sistemas-de-recomendacion-que-son-tipos-y-ejemplos/>, 2023. Accessed: May 2024.
- [6] C. C. Aggarwal, *Recommender Systems: The Textbook*. Springer, 2016. Link a Google Books.
- [7] Steam, "Presentamos el recomendador interactivo." <https://steamcommunity.com/games/593110/announcements/detail/1612767708821405787?l=spanish>, 2019. Accessed: 16-May-2024.
- [8] "Evaluation metrics for search and recommendation systems." <https://weaviate.io/blog/evaluation-metrics-for-search-and-recommendation-systems>, 2024. Accessed: Jul. 11, 2024.
- [9] Meta, "Pytorch." <https://opensource.fb.com/projects/pytorch/>, 2024. Accessed: Jul. 11, 2024.
- [10] IBM, "What is pytorch?." <https://www.ibm.com/cloud/learn/pytorch>, 2024. Accessed: Jul. 11, 2024.
- [11] Google, "Tensorflow: An end-to-end open source platform for machine learning." <https://opensource.google/projects/tensorflow>, 2024. Accessed: Jul. 11, 2024.
- [12] RecBole, "Gru4rec — recbole 1.2.0 documentation." https://recbole.io/docs/user_guide/model/sequential/gru4rec.html, 2023. Accessed: May 16, 2024.
- [13] Y. K. Tan, X. Xu, and Y. Liu, "Improved recurrent neural networks for session-based recommendations," in *Proceedings of the 2016 ACM on Conference on Recommender Systems (RecSys)*, pp. 17–22, 2016. Accessed: Jul. 11, 2024.
- [14] RecBole, "Gru4rec — recbole 1.2.0 documentation." https://recbole.io/docs/user_guide/model/sequential/gru4rec.html, 2024. Accessed: Jul. 11, 2024.
- [15] R. He and J. McAuley, "Fusing similarity models with markov chains for sparse sequential recommendation," in *Proceedings of the 2016 IEEE International Conference on Data Mining (ICDM)*, pp. 191–200, 2016. Accessed: Jul. 11, 2024.

- [16] RecBole, “Fossil — recbole 1.2.0 documentation.” https://recbole.io/docs/user_guide/model/sequential/fossil.html, 2024. Accessed: Jul. 11, 2024.
- [17] P. Wang, J. Guo, Y. Lan, J. Xu, S. Wan, and X. Cheng, “Learning hierarchical representation model for next basket recommendation,” in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 403–412, 2015. Accessed: Jul. 11, 2024.
- [18] RecBole, “Hrm — recbole 1.2.0 documentation.” https://recbole.io/docs/user_guide/model/sequential/hrm.html, 2024. Accessed: Jul. 11, 2024.
- [19] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, “Session-based recommendation with graph neural networks,” in *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, pp. 346–353, 2019. Accessed: Jul. 11, 2024.
- [20] RecBole, “Srgnn — recbole 1.2.0 documentation.” https://recbole.io/docs/recbole/recbole.model.sequential_recommender.srgnn.html, 2024. Accessed: Jul. 11, 2024.
- [21] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang, “Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer,” in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*, pp. 1441–1450, 2019. Accessed: Jul. 11, 2024.
- [22] RecBole, “Bert4rec — recbole 1.2.0 documentation.” https://recbole.io/docs/recbole/recbole.model.sequential_recommender.bert4rec.html, 2024. Accessed: Jul. 11, 2024.
- [23] C. Ma, P. Kang, and X. Liu, “Hierarchical gating networks for sequential recommendation,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 825–833, 2019. Accessed: Jul. 11, 2024.
- [24] RecBole, “Hgn — recbole 1.2.0 documentation.” https://recbole.io/docs/recbole/recbole.model.sequential_recommender.hgn.html, 2024. Accessed: Jul. 11, 2024.
- [25] W.-C. Kang and J. McAuley, “Self-attentive sequential recommendation,” in *Proceedings of the 2018 IEEE International Conference on Data Mining (ICDM)*, pp. 197–206, 2018. Accessed: Jul. 11, 2024.
- [26] RecBole, “Shan — recbole 1.2.0 documentation.” https://recbole.io/docs/recbole/recbole.model.sequential_recommender.shan.html, 2024. Accessed: Jul. 11, 2024.
- [27] T. Nguyen and A. Takasu, “Npe: Neural personalized embedding for collaborative filtering,” in *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1583–1589, 2018. Accessed: Jul. 11, 2024.
- [28] RecBole, “Npe — recbole 1.2.0 documentation.” https://recbole.io/docs/user_guide/model/sequential/npe.html, 2024. Accessed: Jul. 11, 2024.
- [29] S. Curiel, “Curso django: Entendiendo cómo trabaja django.” <https://www.maestrosdelweb.com/curso-django-entendiendo-como-trabaja-django/>,

2021. Accessed: 11-Jun-2024.

APÉNDICES

FICHERO REQUIREMENTS

Código A.1: En esta figura se presentan los requerimientos del proyecto listados en el archivo requirements.txt.

```

1  absl-py==2.1.0
2  aiosignal==1.3.1
3  asgiref==3.8.1
4  asttokens @ file:///opt/conda/conda-bld/asttokens_1646925590279/work
5  attrs==23.2.0
6  backcall @ file:///home/ktietz/src/ci/backcall_1611930011877/work
7  brotlipy==0.7.0
8  certifi @ file:///C:/b/abs_4f5wo627a3/croots/recipe/certifi_1663615677642/work/certifi
9  cffi @ file:///C:/Windows/Temp/abs_6808y9x40v/croots/recipe/cffi_1659598653989/work
10  charset-normalizer @ file:///tmp/build/80754af9/charset-normalizer_1630003229654/work
11  click==8.1.7
12  cloudpickle==3.0.0
13  colorama==0.4.4
14  colorlog==4.7.2
15  comm==0.2.2
16  conda==22.9.0
17  conda-package-handling @
     file:///C:/b/abs_81m11h_i4r/croots/recipe/conda-package-handling_1663598470202/work
18  contourpy==1.0.5
19  cryptography @ file:///C:/ci/cryptography_1652083563162/work
20  cycler==0.11.0
21  debugpy @ file:///C:/ci/debugpy_1637091961445/work
22  decorator @ file:///opt/conda/conda-bld/decorator_1643638310831/work
23  Django==4.2.13
24  entrypoints @ file:///C:/ci/entrypoints_1649926621128/work
25  executing @ file:///opt/conda/conda-bld/executing_1646925071911/work
26  filelock==3.13.3
27  fonttools==4.37.4
28  frozenlist==1.4.1
29  fsspec==2024.3.1
30  future==1.0.0
31  grpcio==1.62.1
32  hyperopt==0.2.5
33  idna @ file:///tmp/build/80754af9/idna_1637925883363/work
34  importlib-metadata==5.0.0
35  ipykernel @ file:///C:/b/abs_21ykzkm7y_croots/recipe/ipykernel_1662361803478/work
36  ipython @ file:///C:/ci/ipython_1657634415474/work
37  ipywidgets==8.1.2
38  jedi @ file:///C:/ci/jedi_1644315428289/work
39  Jinja2==3.1.3
40  joblib==1.2.0
41  jsonschema==4.22.0
42  jsonschema-specifications==2023.12.1
43  jupyter-core @ file:///C:/ci/jupyter_core_1651656286743/work
44  jupyter_client @ file:///C:/b/abs_8fbm7986b_croots/recipe/jupyter_client_1662504374117/work
45  jupyterlab_widgets==3.0.10
46  kiwisolver==1.4.4
47  kmeans-pytorch==0.3
48  Mako==1.2.3
49  Markdown==3.4.1
50  MarkupSafe==2.1.1
51  matplotlib==3.6.1

```

Código A.2: En esta figura se presentan los requerimientos del proyecto listados en el archivo requirements.txt parte 2

```
1 matplotlib-inline @ file:///C:/ci/matplotlib-inline_1661915841596/work
2 menuinst @ file:///C:/ci/menuinst_1631733438520/work
3 mkl-fft==1.3.1
4 mkl-random @ file:///C:/ci/mkl_random_1626186184308/work
5 mkl-service==2.4.0
6 mpmath==1.3.0
7 msgpack==1.0.8
8 nest-asyncio @ file:///C:/ci/nest-asyncio_1649829929390/work
9 networkx==3.2.1
10 numpy==1.23.3
11 packaging @ file:///tmp/build/80754af9/packaging_1637314298585/work
12 pandas==2.2.1
13 parso @ file:///opt/conda/conda-bld/parso_1641458642106/work
14 pdoc3==0.10.0
15 pickleshare @ file:///tmp/build/80754af9/pickleshare_1606932040724/work
16 Pillow==9.2.0
17 plotly==5.20.0
18 prompt-toolkit @ file:///tmp/build/80754af9/prompt-toolkit_1633440160888/work
19 protobuf==5.26.1
20 psutil @
   file:///C:/Windows/Temp/abs_b2c2fd7f-9fd5-4756-95ea-8aed74d0039flsd9qufz/croots/recipe/psutil_1656431277748/work
21 pure-eval @ file:///opt/conda/conda-bld/pure_eval_1646925070566/work
22 py4j==0.10.9.7
23 pyarrow==16.1.0
24 pycosat==0.6.3
25 pycparser @ file:///tmp/build/80754af9/pycparser_1636541352034/work
26 Pygments @ file:///opt/conda/conda-bld/pygments_1644249106324/work
27 pyOpenSSL @ file:///opt/conda/conda-bld/pyopenssl_1643788558760/work
28 pyparsing @
   file:///C:/Users/BUILDE~1/AppData/Local/Temp/abs_7f_7lba6rl/croots/recipe/pyparsing_1661452540662/work
29 PySocks @ file:///C:/ci/pysocks_1605307512533/work
30 python-dateutil @ file:///tmp/build/80754af9/python-dateutil_1626374649649/work
31 pytz==2024.1
32 pywin32==302
33 PyYAML==6.0.1
34 pyzmq @ file:///C:/ci/pyzmq_1657615952984/work
35 ray==2.6.3
36 recbole==1.2.0
37 referencing==0.35.1
38 requests @ file:///C:/ci/requests_1657735342357/work
39 rpds-py==0.18.1
40 ruamel-yaml-conda @ file:///C:/ci/ruamel_yaml_1616016898638/work
41 scikit-learn==1.1.2
42 scipy==1.9.2
43 six @ file:///tmp/build/80754af9/six_1644875935023/work
44 sqlparse==0.5.0
45 stack-data @ file:///opt/conda/conda-bld/stack_data_1646927590127/work
46 sympy==1.12
47 tabulate==0.9.0
48 tenacity==8.2.3
49 tensorflow==2.16.2
50 tensorflow-data-server==0.7.2
```

Código A.3: En esta figura se presentan los requerimientos del proyecto listados en el archivo requirements.txt parte 3

```
1 tensorboardX==2.6.2.2
2 texttable==1.7.0
3 thop==0.1.1.post2209072238
4 threadpoolctl==3.1.0
5 toolz @ file:///tmp/build/80754af9/toolz_1636545406491/work
6 torch==2.2.2
7 tornado @ file:///C:/ci/tornado_1662458743919/work
8 tqdm @ file:///C:/ci/tqdm_1650636210717/work
9 traitlets @ file:///tmp/build/80754af9/traitlets_1636710298902/work
10 typing_extensions==4.11.0
11 tzdata==2024.1
12 urllib3 @ file:///C:/Windows/TEMP/abs_65ynz4fdmi/croots/recipe/urllib3_1659110473919/work
13 vboxapi==1.0
14 wcwidth @ file:///Users/ktietz/demo/mc3/conda-bld/wcwidth_1629357192024/work
15 Werkzeug==3.0.2
16 widgetsnbextension==4.0.10
17 win-inet-pton @ file:///C:/ci/win_inet_pton_1605306162074/work
18 wincerystore==0.2
19 zipp==3.9.0
```




Universidad Autónoma
de Madrid