Proyecto de Sistemas de Recuperación de Información (SRI) - Recomendación Secuencial Colaborativa

Paula Silva Lara (C-312), Ricardo Cápiro Colomar (C-312), and Ariel González Gómez (C-312)

Facultad de Matemática y Computación (MATCOM), Universidad de La Habana, Cuba

Abstract. A diferencia de los Sistemas de Recomendación (RS) convencionales, los Sistemas de Recomendación Secuencial (SRS) intentan comprender y modelar los comportamientos secuenciales de los usuarios, las interacciones entre los usuarios y los elementos, y la evolución de las preferencias de los usuarios y la popularidad de los elementos a lo largo del tiempo. Los SRS involucran los aspectos anteriores para una caracterización más precisa de los contextos, intenciones y objetivos del usuario, y la tendencia de consumo de artículos, lo que lleva a recomendaciones más precisas, personalizadas y dinámicas.
En este proyecto se investigó la aplicación de las redes neuronales recurrentes (*Recurrent Neural Networks* - RNN) para la recomendación secuencial colaborativa y se utilizó para predecir el próximo producto más probable que un usuario comprará en una plataforma de comercio electrónico, centrándose específicamente en productos electrónicos. Se utilizó un conjunto de datos de Kaggle para entrenar un modelo RNN de Memoria Larga a Corto Plazo (*Long Short-Term Memory* - LSTM). Se evaluó el rendimiento y la precisión del modelo utilizando varias métricas, y se creó una aplicación visual. El código está disponible en: `https://github.com/Pola03/SRI-Project/tree/main`.

Keywords: Sequencial Recommendation System · Recurrent Neural Network · Gated Recurrent Unit

1 Introducción

Los sistemas de recomendación secuencial (SRS) sugieren productos que podrían interesar a un usuario basándose en las interacciones previas entre el usuario y los productos (como ver o comprar en una plataforma de comercio electrónico). A diferencia de los sistemas de recomendación tradicionales, los SRS modelan estas interacciones como una secuencia temporal, capturando las dependencias entre cada acción para ofrecer recomendaciones más precisas. Los sistemas de recomendación tradicionales (RS), incluyendo los basados en popularidad, los de recomendación demográfica, los basados en contenido y los de filtrado colaborativo, modelan las interacciones usuario-artículo de forma estática y solo

pueden capturar las preferencias generales del usuario. En contraste, los SRS tratan las interacciones usuario-artículo como una secuencia dinámica y consideran las dependencias secuenciales para capturar las preferencias actuales y recientes del usuario, lo que permite obtener recomendaciones más precisas [Chen et al., 2018]. Para mejorar la comprensión de los SRS, a continuación se presentan las motivaciones que los sustentan.

Las interacciones usuario-artículo están esencialmente interconectadas de forma secuencial. En el mundo real, los comportamientos de compra (u otras interacciones) de los usuarios generalmente suceden sucesivamente en una secuencia, en lugar de forma aislada.

Tanto las preferencias de los usuarios como la popularidad de los artículos son dinámicas en lugar de estáticas con el tiempo. De hecho, la preferencia y el gusto de un usuario pueden cambiar con el tiempo. Estas dinámicas son de gran importancia para la creación de perfiles precisos de usuarios o artículos y, en consecuencia, para obtener recomendaciones más precisas, y solo pueden ser capturadas por los SRS.

Las interacciones usuario-artículo generalmente ocurren bajo un cierto contexto secuencial. Diferentes contextos conducen a diferentes interacciones de los usuarios con los artículos y, sin embargo, a menudo es ignorado por los RS tradicionales como el filtrado colaborativo [Kang et al., 2018]. En contraste, un SRS considera las interacciones secuenciales anteriores como un contexto para predecir qué artículos serán interactuados en el futuro cercano. Como resultado, es mucho más fácil diversificar los resultados de la recomendación al evitar recomendar repetidamente aquellos artículos idénticos o similares a los que ya han sido elegidos.

1.1 Antecedentes

En los últimos años, los Sistemas de Recomendación Secuencial (SRS) han evolucionado significativamente gracias a la implementación de redes neuronales profundas (Deep Neural Networks), que han demostrado una capacidad excepcional para modelar patrones secuenciales complejos. A diferencia de los modelos tradicionales de recomendación, los SRS buscan captar la naturaleza dinámica de las interacciones entre los usuarios y los productos a lo largo del tiempo, lo que resulta en recomendaciones más precisas y personalizadas. Dentro de este campo, las redes neuronales recurrentes (RNN), convolucionales (CNN) y las redes neuronales de grafos (GNN) han desempeñado un papel crucial.

Modelos de Redes Neuronales Profundas para SRS: Las redes neuronales profundas se han consolidado como una herramienta poderosa para capturar las relaciones entre diferentes entidades (usuarios, artículos, interacciones) en una secuencia temporal. Esta capacidad ha permitido que los SRS basados en estos modelos dominen la investigación reciente. Estos enfoques se pueden agrupar en dos categorías principales: los SRS construidos sobre redes neuronales profundas básicas y aquellos que incorporan modelos avanzados en su estructura.

Redes Neuronales Profundas Básicas: Entre los modelos más utilizados para SRS destacan las redes neuronales recurrentes (RNN), conocidas por su capacidad para modelar secuencias temporales. Sin embargo, a pesar de sus ventajas, las RNN presentan limitaciones, como la incapacidad de manejar eficientemente secuencias largas. Para abordar estos defectos, se han introducido otras arquitecturas, como las redes neuronales convolucionales (CNN) y las redes neuronales de grafos (GNN), que complementan y mejoran las deficiencias de las RNN.

SRS basados en RNN: Los SRS basados en RNN intentan predecir la próxima interacción de un usuario con un artículo modelando las dependencias secuenciales de sus interacciones pasadas. Aparte de la RNN básica, se han desarrollado variantes como las redes LSTM (Memoria a Largo Plazo) [Wu et al., 2017] y las Unidades Recurrentes con Puerta (GRU) [Hidasi et al., 2016], que capturan de manera más efectiva las dependencias a largo plazo dentro de las secuencias. En los últimos años, estos enfoques han dominado el campo de los SRS basados en aprendizaje profundo. Además, se han propuesto variantes más sofisticadas, como las RNN jerárquicas [Quadrana et al., 2017], diseñadas para capturar dependencias más complejas y multinivel en secuencias de interacción.

2 Preliminares

Para tener una fundamentación teórica de los elementos que se tratan en las secciones posteriores, se procede a hacer una formulación formal del problema de recomendación secuencial. Luego, se brinda una breve explicación de las Redes Neuronales Recurrentes (RNN) y de las Unidades de Memoria Larga a Corto Plazo (*Long Short-Term Memory* - LSTM), que se utilizaron en este proyecto.

2.1 Formulación del problema

En una tarea de recomendación secuencial, se tiene un conjunto de $N = |\mathcal{U}|$ usuarios $\mathcal{U} = \{u_1, u_2, ..., u_N\}$ que han tenido interacciones históricas con un conjunto de $M = |\mathcal{V}|$ artículos $\mathcal{V} = \{v_1, v_2, ..., v_M\}$. Entre estos usuarios, el usuario i-ésimo tiene una secuencia de artículos preferidos denotada como $s_i = [v_1^{(i)}, v_2^{(i)}, ..., v_{n_i}^{(i)}]$, donde n_i es la longitud de la lista de artículos con la que el usuario i-ésimo interactúa. El objetivo es diseñar un marco eficiente y predecir el siguiente artículo con el que el usuario tiene mayor probabilidad de interactuar.

2.2 Redes Neuronales Recurrentes (RNN)

Las Redes Neuronales Recurrentes están ideadas para modelar datos secuenciales de longitud variable. La principal diferencia entre las RNN y los *Feedforward Deep Models* es la existencia de un estado oculto interno en las unidades que componen la red. Los RNN estándar actualizan su estado oculto h utilizando la siguiente función de actualización:

$$h_t = g(Wx_t + Uh_{t-1}) \tag{1}$$

Donde g es una función suave y acotada, como una función sigmoidea logística, x_t es la entrada de la unidad en el momento t. Una RNN genera una distribución de probabilidad sobre el siguiente elemento de la secuencia, dado su estado actual h_t.[Sherstinsky, 2020]

2.3 *Long Short-Term Memory* (LSTM)

Las Unidades de Memoria Larga a Corto Plazo (*Long Short-Term Memory* - LSTM) son una mejora de las RNN diseñadas específicamente para resolver el problema del desvanecimiento del gradiente (*vanishing gradient*), que ocurre durante el entrenamiento de redes neuronales profundas. Este problema es particularmente evidente en modelos recurrentes o de muchas capas, donde los gradientes de las capas más cercanas a la entrada se vuelven extremadamente pequeños, lo que impide que los pesos se actualicen adecuadamente. Como resultado, la red puede aprender de manera ineficiente o no aprender en absoluto, afectando su capacidad para modelar dependencias a largo plazo en secuencias. Las LSTM abordan este problema mediante el uso de una estructura de celda especializada que puede retener información durante largos períodos de tiempo y decidir cuándo almacenar o eliminar datos del estado de la celda [Sherstinsky, 2020].

El componente clave de las LSTM es la celda de memoria, que es capaz de mantener información durante muchos pasos de tiempo. La interacción entre la celda de memoria y las puertas de entrada, salida y olvido es lo que permite que las LSTM aprendan dependencias a largo plazo. La actualización del estado de la celda c_t y el estado oculto h_t se rige por las siguientes ecuaciones:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c_t} \tag{2}$$

donde $\tilde{c_t}$ es la nueva candidata para el estado de la celda, y las puertas de entrada i_t y olvido f_t se definen como:

$$i_t = \sigma(W_i x_t + U_i h_{t-1}) \tag{3}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1}) \tag{4}$$

El estado oculto h_t de la LSTM se calcula a partir del estado de la celda y la puerta de salida o_t, usando la siguiente fórmula:

$$h_t = o_t \odot \tanh(c_t) \tag{5}$$

donde la puerta de salida se define como:

$$o_t = \sigma(W_o x_t + U_o h_{t-1}) \tag{6}$$

En este modelo, las puertas permiten que la LSTM aprenda cuándo permitir el paso de nueva información (puerta de entrada), cuándo olvidar información antigua (puerta de olvido), y cuándo permitir que el estado de la celda actual afecte la salida (puerta de salida). Gracias a esta estructura de control, las LSTM pueden modelar relaciones temporales complejas en datos secuenciales, reteniendo de manera efectiva información importante durante períodos largos. Sin embargo, aunque las LSTM son efectivas para capturar dependencias a largo plazo, también son más costosas computacionalmente en comparación con las GRU debido a su arquitectura más compleja.

3 Problemática específica a abordar

En este proyecto se planteó una problemática concreta dentro de un entorno ficticio pero alineado con situaciones comunes en la realidad. Este enfoque permite definir con precisión cada componente del problema y su correspondiente solución, lo cual facilita su exposición y comprensión. Aunque el contexto utilizado es hipotético, la estructura del problema y la solución propuesta son lo suficientemente versátiles como para ser aplicadas en diversos escenarios similares del mundo real, lo que demuestra la generalidad del modelo desarrollado.

La problemática a tratar consiste en predecir cuál será el próximo producto electrónico que un usuario comprará en una plataforma de comercio electrónico. Para entrenar el modelo, se utilizó un subconjunto de datos provenientes del conjunto *eCommerce behavior data from multi-category store*, disponible en Kaggle: https://www.kaggle.com/datasets/mkechinov/ecommerce-behavior-data-from-multi-category-store.

Este conjunto de datos abarca un periodo de 7 meses, desde octubre de 2019 hasta abril de 2020, y contiene información detallada sobre el comportamiento de los usuarios en una gran tienda en línea con diversas categorías de productos. Cada fila del archivo representa un evento, donde todos los eventos están vinculados a productos y usuarios, conformando una relación de muchos a muchos entre ellos [Kechinov, 2019].

La base de datos proporciona un registro detallado de las compras realizadas por los usuarios en la plataforma, junto con las características de los productos adquiridos. Para este trabajo, se seleccionó específicamente el archivo correspondiente a las compras realizadas durante noviembre de 2019. De esta manera, se trabajó con una versión filtrada del conjunto de datos original, centrándose exclusivamente en las entradas relacionadas con productos electrónicos.

Para la tarea de predicción mediante redes neuronales, solo se consideraron los eventos de tipo "purchase", eliminando las columnas que no se utilizaron en este análisis, como *event_time*, *event_type*, *category_id*, y *user_session*. Estas columnas, aunque no fueron empleadas en esta fase del proyecto, podrían ser útiles en trabajos futuros para mejorar el modelo y ampliar el análisis.

4 Descripción de la propuesta

En esta sección se explica el flujo del trabajo, y se presenta el modelo de red neuronal como propuesta de solución, dando una explicación breve de su arquitectura, funcionamiento y flujo de datos, seguida de una exploración más profunda de cada uno de sus componentes técnicos.

4.1 Arquitectura del modelo y flujo de la red

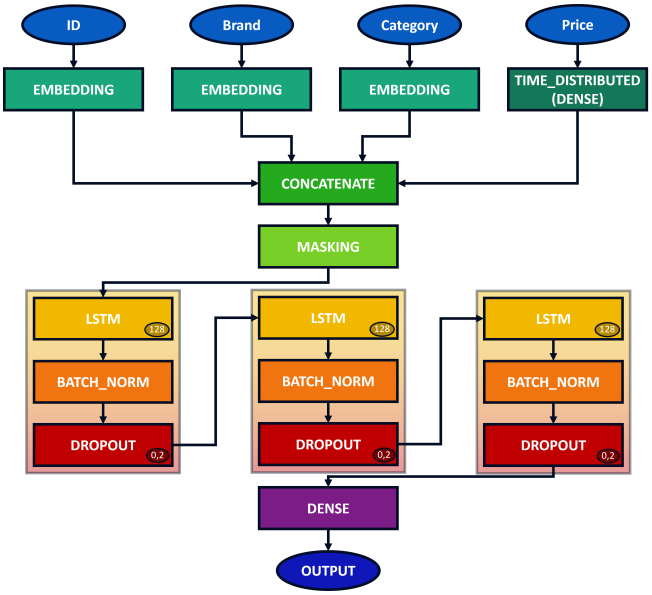

Fig. 1. Arquitectura del modelo propuesto.

El flujo de la red se organiza en los siguientes pasos:

1. **Entrada (*Input*)**: La red recibe, para cada usuario, cuatro secuencias relacionadas con sus compras anteriores. Estas secuencias incluyen los identificadores de los productos adquiridos, las marcas de dichos productos, las categorías a las que pertenecen y sus respectivos precios. Estas entradas capturan información clave del historial de compras del usuario, proporcionando un contexto valioso para predecir su próxima adquisición.
2. ***Embeddings* y *Time Distributed (Dense)***: Las entradas categóricas (ID, marca, categoría) se convierten en vectores densos utilizando capas de *embeddings*, mientras que los precios se procesan a través de una capa *Time Distributed (Dense)* para obtener una representación numérica embebida.

3. **Concatenación (*Concatenate*)**: Los vectores embebidos para cada paso de tiempo se concatenan junto con los precios, creando una secuencia de vectores de entrada para las capas recurrentes.
4. **Enmascaramiento (*Masking*)**: Se utiliza una capa de *Masking* para ignorar los valores de relleno en las secuencias de entrada, que varían en longitud debido al diferente número de productos comprados por los usuarios.
5. **Bloques Recurrentes con Normalización**: A continuación, la secuencia concatenada pasa por tres bloques consecutivos, cada uno compuesto por una capa LSTM de 128 neuronas, seguida de una capa de *Batch Normalization* y una capa de *Dropout*. Estos bloques permiten que la red capture las dependencias temporales complejas en la secuencia de productos, mientras que la normalización estabiliza las activaciones y el *dropout* ayuda a prevenir el sobreajuste.
6. **Capa Densa Final (*Dense*)**: La salida del último bloque LSTM pasa a una capa densa, encargada de producir una distribución de probabilidad sobre todos los productos posibles.
7. **Salida *Output***: La capa de salida aplica una función de activación *softmax* para predecir el próximo producto más probable que el usuario comprará, seleccionando aquel con la mayor probabilidad.

4.2 Explicación de componentes principales

Capas de *Embeddings*: La capa de *embedding* es esencial para que una red neuronal pueda "entender" datos categóricos, como los IDs de producto, marcas y categorías, que no son directamente procesables por una red neuronal. La capa de *embedding* funciona como un diccionario que asigna a cada palabra (en este caso, cada ID de producto, marca o categoría) un vector numérico de dimensiones fijas. Este vector, llamado embedding, representa la "definición" de la palabra en un espacio multidimensional.

La capa de *embedding* es una tabla de búsqueda que mapea cada palabra (categoría) a un vector de tamaño fijo. La red neuronal aprende la posición de cada vector en este espacio durante el entrenamiento, y las distancias entre los vectores representan la similitud entre las categorías.

La red neuronal, durante el entrenamiento, aprende a posicionar estos vectores en el espacio de tal manera que categorías similares estén cerca y categorías diferentes estén lejos. Esto permite que la red neuronal capture la relación entre las categorías y use esta información para realizar predicciones.

Beneficios de la capa de *embedding*:

- Conversión de datos categóricos a numéricos: Permite que la red neuronal procese datos categóricos.
- Representación semántica: Captura relaciones entre las categorías.
- Reducción de la dimensionalidad: La capa de *embedding* reduce la dimensionalidad del espacio categórico, simplificando el aprendizaje.
- Mejora del rendimiento: Permite a la red neuronal aprender representaciones más ricas y precisas de los datos categóricos, lo que conduce a una mejor precisión predictiva.

Capa *Masking*: La capa *Masking* se utiliza en redes neuronales cuando se hace necesario procesar secuencias de diferente longitud, como es común en problemas de procesamiento de secuencias. Esta identifica los valores de *padding* (ceros en este caso) en las secuencias de entrada y marca estos valores para que las capas posteriores, los ignoren. Su uso es crucial para que el modelo no interprete estos valores como información válida y no aprenda patrones incorrectos basados en ellos.

Capa LSTM: Explicada en la sección de Preliminares. El módulo LSTM contribuye a la tarea de recomendación al capturar las dependencias a largo plazo entre los artículos en la secuencia y ajustar dinámicamente el contenido de su memoria a través de diferentes partes de la secuencia.

Capa *Dropout*: La capa *Dropout* es una técnica de regularización utilizada en redes neuronales para reducir el sobreajuste (*overfitting*). En redes profundas, es común que el modelo se ajuste demasiado a los datos de entrenamiento, lo que provoca una disminución del rendimiento en datos no vistos. *Dropout* mitiga este problema desconectando aleatoriamente una fracción de las neuronas durante el entrenamiento, lo que obliga a la red a no depender de conjuntos específicos de características y aprender representaciones más generales.

Propósito: Prevenir el sobreajuste de manera que la red no dependa demasiado de ciertos patrones o combinaciones específicas de pesos.

Cómo funciona: *Dropout* desconecta aleatoriamente una proporción de neuronas en cada paso de entrenamiento (en este caso, el 20% con Dropout(0.2)). Esto significa que, en cada paso, las neuronas que queden activas deben compensar la información ausente, lo que fortalece el modelo general al hacerlo menos sensible a patrones ruidosos o irrelevantes.

Beneficio: Permite que el modelo generalice mejor, especialmente cuando se tiene un conjunto de datos limitado o cuando hay muchas características que pueden llevar a sobreajuste.

Capa *Dense*: La capa *Dense* (o capa completamente conectada) es una capa fundamental en las redes neuronales, donde cada neurona de la capa está conectada con todas las neuronas de la capa anterior. Esta capa es responsable de aprender combinaciones no lineales de las características extraídas por capas anteriores (en este caso, de los bloques LSTM->$Batch\_Norm$->$Dropout$) y tomar decisiones sobre la salida.

Propósito: La capa *Dense* actúa como un "decisor" en la red, combinando la información aprendida en capas anteriores y produciendo una salida final. En este caso, se usó una capa con activación $softmax$ para producir una distribución de probabilidad sobre las posibles clases de productos que pueden ser recomendados.

Cómo funciona: Cada neurona en la capa *Dense* toma una combinación ponderada de las entradas (en el proyecto, la salida de los bloques LSTM->$Batch\_Norm$->$Dropout$), aplica una función de activación (en este caso, $softmax$

para convertir las salidas en probabilidades) y genera una predicción. El modelo ajusta estos pesos durante el entrenamiento para minimizar la pérdida.

Beneficio: Las capas *Dense* permiten realizar una clasificación efectiva en un espacio de características de alto nivel. En el proyecto, la última capa *Dense* con $n\_products$ unidades y activación $softmax$ devuelve las probabilidades de cada producto, lo que permite al modelo predecir cuál es el producto más probable que un usuario compre.

Relación entre las capas *Dropout* y *Dense*: *Dropout* regulariza las salidas de la capa LSTM, haciendo que la red sea más robusta y menos propensa al sobreajuste. *Dense* utiliza estas salidas regularizadas para realizar la tarea final de clasificación, determinando cuál es el siguiente producto que debe recomendarse al usuario. La combinación de ambas capas mejora el rendimiento y la capacidad de generalización del modelo. Estas dos capas son comunes en arquitecturas de redes neuronales por su capacidad de combinar regularización (*Dropout*) y clasificación efectiva (*Dense*).

5 Resultados

Se entrenó el modelo propuesto sobre el conjunto de datos filtrados extraido, durante 10 épocas de entrenamiento. Las gráficas 2 y 3 muestran la evolución de la precisión (*accuracy*) y error o pérdida (*loss*) del modelo durante el entrenamiento, al igual que la validación.

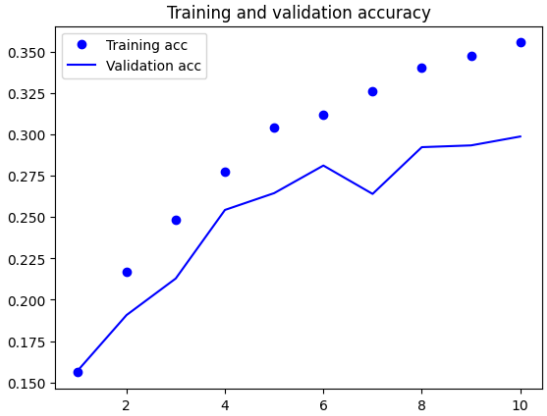

Fig. 2. Precisión durante el entrenamiento y la validación.

Los resultados obtenidos muestran una clara presencia de sobreajuste (*overfitting*).

Como resultado del entrenamiento se obtuvieron los valores finales siguientes:

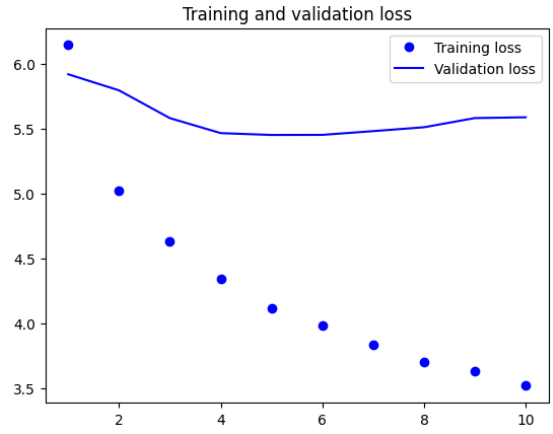

Fig. 3. Pérdida durante el entrenamiento y la validación.

- ***Loss* en el conjunto de prueba**: 5.7129106521606445
- ***Accuracy* en el conjunto de prueba**: 0.3037187457084656

6 Conclusiones

En este proyecto, se desarrolló un modelo de recomendación secuencial utilizando redes neuronales recurrentes (RNN) con la arquitectura de Memoria a Largo Plazo (LSTM). Se entrenó el modelo sobre un conjunto de datos filtrados extraído de una fuente pública, llevando a cabo un total de 10 épocas de entrenamiento. Los resultados obtenidos fueron evaluados a través de métricas de precisión (*accuracy*) y error o pérdida (*loss*), las cuales se visualizaron mediante gráficas que mostraron la evolución de estas métricas tanto en el conjunto de entrenamiento como en el de validación.

Los experimentos revelaron una clara presencia de sobreajuste (*overfitting*), evidenciado por la discrepancia entre el rendimiento en los conjuntos de entrenamiento y validación. A pesar de que el modelo alcanzó un nivel de precisión de aproximadamente 0.30 en el conjunto de prueba, este valor sugiere que el modelo no logró capturar de manera efectiva las complejidades de las interacciones entre los usuarios y los productos.

Desde una perspectiva crítica, se identificaron varias áreas para la mejora del modelo. En primer lugar, se sugiere la posibilidad de implementar técnicas de regularización más robustas, como la reducción de la tasa de aprendizaje o el aumento de la proporción de *dropout*, para mitigar el sobreajuste. Asimismo, se propone la inclusión de características adicionales en los datos de entrada, como información demográfica de los usuarios o datos de contexto en tiempo real, lo que podría enriquecer las representaciones aprendidas por el modelo.

Adicionalmente, se consideró para trabajos futuros la opción de experimentar con diferentes arquitecturas de red, como el uso de capas convolucionales

antes de las LSTM o la implementación de modelos basados en atención, que podrían mejorar la captura de las dependencias a largo plazo en las secuencias de interacciones. En resumen, a pesar de los desafíos encontrados, este trabajo nos sentó las bases para futuros desarrollos en sistemas de recomendación secuencial, brindando valiosas lecciones y líneas de investigación a seguir.

7 Anexos

Para probar el modelo y presentar una aplicación práctica del sistema desarrollado se creó una aplicación visual utilizando *Streamlit*: 4,5.

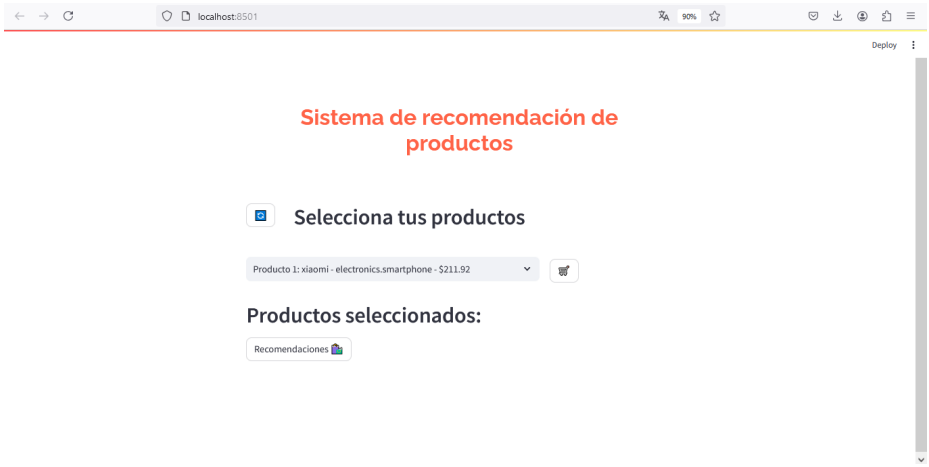

Fig. 4. Aplicación visual. Selección de productos.

References

Chen et al., 2018. Xu Chen, Hongteng Xu, Yongfeng Zhang, and et al. Sequential recommendation with user memory networks. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*, pages 108–116, 2018.

Hidasi et al., 2016. Balázs Hidasi, Alexandros Karatzoglou, and et al. Session-based recommendations with recurrent neural networks. In *Proceedings of the 4th International Conference on Learning Representations*, pages 1–10, 2016.

Kang et al., 2018. Wang-Cheng Kang, Mengting Wan, and Julian McAuley. Recommendation through mixtures of heterogeneous item relationships. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1143–1152, 2018.

Kechinov, 2019. Kechinov, Michael. eCommerce behavior data from multi category store. 2019. Disponible en: `https://www.kaggle.com/datasets/mkechinov/ecommerce-behavior-data-from-multi-category-store` [Consultado el 1 de octubre de 2024].

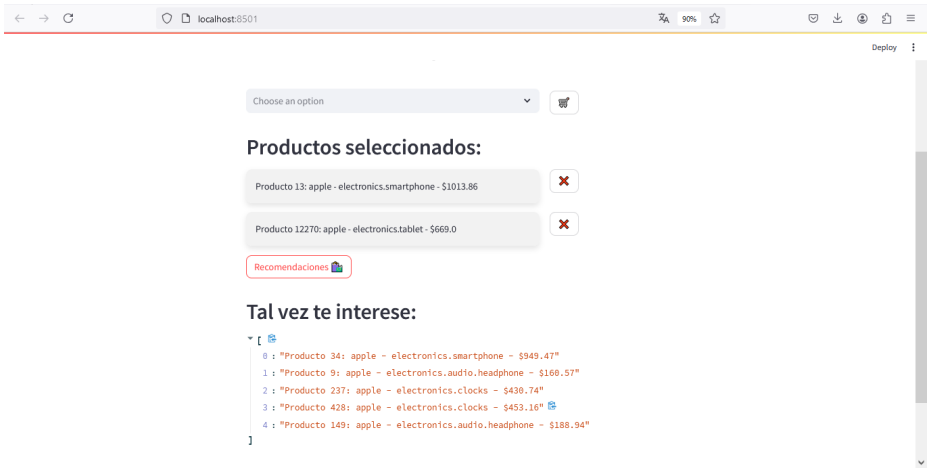

Fig. 5. Aplicación visual. Recomendación de productos.

Quadrana et al., 2017. Massimo Quadrana, Alexandros Karatzoglou, and et al. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *Proceedings of the 11th ACM Conference on Recommender Systems*, pages 130–137, 2017.

Sherstinsky, 2020. Sherstinsky, Alex. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

Wu et al., 2017. Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. Recurrent recommender networks. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining*, pages 495–503, 2017.