



UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE CIENCIAS EXACTAS Y NATURALES
FACULTAD DE INGENIERÍA

Chatbot para la recomendación personalizada de contenido

Tesis de la *Maestría en Explotación de Datos y Descubrimiento del Conocimiento*

Proyecto en Gihub

Adrian Norberto Marino

Director: Roberto Abalde
Buenos Aires, Argentina, 2024

RESUMEN

Este trabajo de tesis experimenta con el uso de grandes modelos de lenguaje (*Large language Models o LLM*) como interfaz de comunicación con el usuario final. Estos modelos se emplean para la búsqueda de contenido personalizado para el usuario en respuesta a consultas en lenguaje natural. El sistema de recomendaciones es capaz de ofrecer contenido personalizado al usuario y aborda el problema del *Cold start* mediante un *ensamble* de modelos de recomendación basados en contenido, filtros colaborativos y modelos de lenguaje.

Palabras claves: *LLM, Cold Start, Sistemas de recomendación basados en Filtro Colaborativos, Sistemas de recomendación en basados en contenido, API, RAG, ChromaDB, Apache Airflow, Llama, Máquinas de factorización profundas (DeepFM)*.

AGRADECIMIENTOS

Principalmente a mis padres, siempre fueron un gran apoyo en mi carrera, alentándome incansablemente para seguir adelante en todo momento. Gran parte de mi disciplina de constante persistencia se la debo a ellos. En segundo lugar a mis profesores de la especialización y maestría, por entregarnos su conocimiento dia a dia, siempre enfocados en que comprendamos todos los temas expuesto de la mejor forma posible. A mis compañeros de la especialización, siempre fueron un gran grupo de apoyo, un grupo en el que nos ayudamos uno al otro para comprender los temas expuestos.

Índice general

1.. Introducción	1
2.. Modelo de datos y análisis exploratorio	3
2.1. El conjunto de datos <i>MovieLens 25M</i>	3
2.2. El conjunto de datos <i>TMDB</i>	4
2.3. Preprocesamiento de datos	5
2.3.1. Tabla de interacciones	5
2.3.2. Tablas de metadatos de películas	6
2.3.3. Valores faltantes	6
2.3.4. Base de interacciones para <i>API Rec Chatbot</i>	7
2.4. Análisis exploratorio	8
2.4.1. Variable <i>Rating</i>	8
2.4.2. Correlaciones	11
2.4.3. Variables de tipo texto	12
2.4.4. Análisis de Componentes Principales	14
3.. Arquitectura General de <i>Rec Chatbot API</i>	19
3.1. Componentes	21
4.. Flujo de inferencia de <i>Rec Chatbot API</i>	23
4.1. Solicitud de recomendaciones	24
4.2. Consulta de interacciones del usuario	24
4.3. ¿El usuario realizó el mínimo de interacciones?	24
4.4. El usuario no cumple con el número mínimo de interacciones	24
4.5. Construir <i>prompt</i> para consulta al modelo de lenguaje	25
4.5.1. <i>Prompt</i> para usuarios que no cumplen el mínimo de interacciones	26
4.5.2. Perfil del usuario	26
4.5.3. Historial de comportamiento del usuario	27
4.5.4. Solicitud del usuario	27
4.5.5. Items candidatos	27
4.5.6. Instrucciones y formato de la respuesta	27
4.6. Consulta a modelo de lenguaje	28
4.7. Búsqueda de items similares en base de datos <i>ChromaDb</i> y construcción de respuesta final	28
4.8. Usuarios de cumplen con el mínimo de interacciones	30
5.. Flujo de entrenamiento de <i>Rec Chatbot API</i>	33
5.1. Flujo de Entrenamiento	33
5.1.1. Flujo de actualización de búsquedas (Técnica <i>RAG(Retrieval Augmented Generation</i> o Generación Aumentada por Recuperación))	35

6.. Modelo interno basado en filtros colaborativos	37
6.1. Codificación <i>One-Hot</i> vs <i>Embeddings</i>	37
6.2. Capa o módulo <i>Embedding</i>	38
6.3. Máquinas de Factorización (<i>FM</i>)	40
6.4. Máquinas de factorización profundas (<i>DeepFM</i>)	42
7.. Pruebas de evaluación	43
7.1. Preparación del ambiente de evaluación	43
7.2. Proceso de evaluación	45
7.3. Métricas	46
7.3.1. NDCG (Normalized Discounted Cumulative Gain)	46
7.3.2. Exhaustividad (<i>Recall</i>)	47
7.3.3. Mean Reciprocal Rank (MRR)	48
7.3.4. Mean Average Precision (MAP)	49
7.3.5. Catalog Coverage	51
7.3.6. Serendipity	52
8.. Resultados	55
9.. Resultados utilizando el componente <i>LLM Llama2</i>	57
10..Resultados utilizando el componente <i>LLM Llama3</i>	69
11..Comparación de resultados	83
12..Conclusiones	85
13..Glosario	87

1. INTRODUCCIÓN

En la era digital actual, la personalización de contenido se ha convertido en un aspecto crucial para mejorar la experiencia del usuario en diversas plataformas de entretenimiento y servicios en línea. Este trabajo de investigación se centra en el desarrollo de un *chatbot* innovador para la recomendación personalizada de contenido, específicamente enfocado en películas, utilizando técnicas avanzadas de procesamiento de lenguaje natural y sistemas de recomendación. Basándose en investigaciones previas [3], este trabajo busca expandir y mejorar estos conceptos mediante la implementación de un sistema de recomendación que combina grandes modelos de lenguaje (en inglés *Large Language Model* o simplemente *LLM*) con técnicas de filtrado colaborativo. Esta propuesta se distingue por su enfoque en la utilización de modelos de lenguaje de código abierto, que pueden ejecutarse en máquinas de especificaciones medias, en contraste con los servicios de *API* comerciales utilizados en trabajos anteriores. Este enfoque no solo reduce los costos asociados, sino que también permite una mayor flexibilidad y control sobre el proceso de recomendación. Este trabajo no solo busca mejorar la precisión de las recomendaciones, sino también explorar cómo los grandes modelos de lenguaje pueden enriquecer la experiencia del usuario al proporcionar recomendaciones más contextuales. A lo largo de este documento, se detallará la arquitectura del sistema, los flujos de inferencia y entrenamiento, los modelos utilizados, y se presentarán los resultados obtenidos. Finalmente, se discutirán las implicaciones de este enfoque y posibles direcciones para futuras investigaciones en el campo de los sistemas de recomendación basados en *chatbots*.

En trabajos previos, se abordaron distintas líneas de investigación con las que se busca mejorar la experiencia de usuario con los sistemas de recomendación mediante el uso de los denominados grandes modelos de lenguaje. El trabajo *ChatREC* [3] aborda el uso de un *LLM*, con el cual el usuario puede entablar una conversación para realizar la búsqueda de contenido personalizado. Basándose en el perfil del usuario y su historial de conversaciones, el modelo realiza recomendaciones de acuerdo a lo que solicite el usuario mediante lenguaje natural. El trabajo mencionado [3] utiliza *LLM*'s a los cuales se puede acceder como un servicio por medio de una *API* de pago, como aquellas ofrecidas por *OpenAI*, *Anthropic* o *DeepSeek* entre otras.

Este trabajo de investigación busca aplicar esta misma técnica utilizando uno o más sistemas de recomendación. Entonces, se busca aplicar la misma técnica utilizando modelos *LLM Open Source* de dominio público, que puedan correr tanto en una máquina virtual o real, de especificaciones medias, y luego tratar de llegar a resultados aceptables bajo condiciones similares. En definitiva, esta investigación explora el uso de grandes modelos de lenguaje para mejorar los resultados de los modelos de recomendación previamente abordados en trabajos anteriores [8], así como también, encontrar nuevas formas de realizar recomendaciones personalizadas, que sean mas cercanas al usuario final.

Para concluir, se definió el alcance de la solución según las diferencias encontradas entre los grandes modelos de lenguaje utilizados previamente y los modelos locales que se aplicarán en este trabajo:

- Desarrollar un *API Rest* para interactuar y configurar el *chatbot* de manera sencilla, permitiendo su integración con sistemas existentes.

- El *chatbot* permite crear un perfil de usuario que contiene información inicial. Esta información ayuda al *chatbot* a hacer recomendaciones basadas en preferencias en género de películas y datos del usuario, abordando el problema del arranque en frío en sistemas de recomendación colaborativa, donde el sistema no tiene información sobre el comportamiento del usuario.
- El *chatbot* permite a los usuarios hacer preguntas en lenguaje natural sobre contenido que desean ver (películas en este caso) y devuelve una lista de películas ordenadas según sus preferencias iniciales o su comportamiento en la plataforma.
- Los usuarios pueden calificar las películas que han visto, y estas calificaciones son utilizadas por el *chatbot* para ajustar las recomendaciones futuras a las nuevas preferencias del usuario. Esto significa que la personalización de las recomendaciones depende de la cantidad de calificaciones que el usuario proporciona, afectando tanto el orden de las recomendaciones como el tipo de contenido presentado.

2. MODELO DE DATOS Y ANÁLISIS EXPLORATORIO

Para realizar este trabajo se seleccionó el dominio del cine, ya que existen conjuntos de datos bien definidos y actualizados. Estos conjuntos de datos en general están diseñados para probar modelos de recomendación. Por otro lado, es el dominio clásico en *papers* y literatura de sistemas de recomendación en general.

Dada la propuesta de este trabajo, fue necesario contar con datos de interacciones de usuarios con ítems (películas en este caso). Por otro lado, no es necesario tener información de los usuarios, ya que los perfiles de los usuarios son cargados por cada usuario y no es necesario realizar un entrenamiento previo para el uso de los modelos de lenguaje como modelo de recomendación basado en contenido. Aun así, se reutilizó la base de usuario, ítem e interacciones seleccionada en el trabajo de especialización [8].

2.1. El conjunto de datos *MovieLens 25M*

El conjunto de datos *MovieLens 25M* [4] prácticamente no tiene características para los ítems (películas), pero sí tiene las calificaciones realizadas por los usuarios. También cuenta con un conjunto de *tags* o palabras clave cargadas por los usuarios para cada ítem (película). Otro punto importante, es que todos los usuarios tienen al menos 20 interacciones, lo cual asegura no tener problemas de bajo rendimiento por falta de estas. De esta forma, este conjunto de datos sera muy útil para entrenar modelos de recomendación basados en filtros colaborativos.

Por último, este conjunto de datos contiene 25 millones calificaciones, 1 millón de *tags* y 62,423 películas. Estos datos fueron registrados por 162,541 usuarios entre el 9 de enero de 1995 y el 21 de noviembre de 2019. A continuación en la tabla 2.1, se especifican las columnas del conjunto de datos:

Columna	Descripción
<i>userId</i>	Identificador unívoco de un usuario.
<i>movieId</i>	Identificador unívoco de una película.
<i>timestamp</i>	Fecha en la cual el usuario calificó el ítem(<i>movieId</i>). Es un string con formato año-mes. Existen valores entre 1997-09 y 2019-11 inclusive.
<i>rating</i>	Calificación del usuario. Es un valor discreto numérico: 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5 y 5.
<i>tags</i>	Lista de palabras definidas por cada usuario, para una película. Dicho de otra forma existen N <i>tags</i> por cada par usuario-película.

Tab. 2.1: Columnas del conjunto de datos *Movie Lens*, relevantes para este trabajo.

2.2. El conjunto de datos *TMDB*

El conjunto de datos *TMDB* [2] no tiene calificaciones personalizadas para los ítems como sucede con el conjunto de datos anterior (*MovieLens*) , pero si tiene varias características referentes a películas, Este conjunto de datos es utilizado para filtrar los ítems del conjunto de datos *MovieLens* [4] y así tener una base de ítems mas rica en características. Estas características son utilizados en el proceso de búsqueda de ítems por el *chatbot* utilizando la técnica *RAG(Retrieval Augmented Generation o Generación Aumentada por Recuperación)* [7] que se explicara mas adelante. El conjunto contiene datos de 5,000 películas y sus columnas se especifican en la tabla 2.2 a continuación:

Columna	Descripción
<i>imdb_id</i>	Identificador unívoco de una película en la base de datos de <i>IMDB</i> .
<i>title y original_title</i>	Título y título original.
<i>release_date</i>	Fecha de estreno.
<i>status</i>	Define si la película fue estrenada o está en desarrollo.
<i>overview</i>	Sinopsis de la película.
<i>poster_path</i>	URL de imagen de portada de la película.
<i>languages</i>	Lenguaje original y doblaje.
<i>genres</i>	Géneros.
<i>adult</i>	¿Solo es apta para adultos?
<i>popularity</i>	Índice de popularidad.
<i>vote_count</i>	Cantidad de votos.
<i>vote_average</i>	Promedio de votos.
<i>keywords/tagline y tags</i>	<i>tags</i> o palabras clave ingresadas por los usuarios para definir una película.
<i>budget</i>	Presupuesto destinado a la realización de la película.
<i>revenue</i>	Retorno de inversión o ganancias.
<i>production_companies</i>	Compañías que produjeron la película.
<i>production_countries</i>	Países donde se produjo la película.
<i>homepage</i>	Sitio web oficial.

Tab. 2.2: Columnas del conjunto de datos *TMDB*, relevantes para este trabajo.

2.3. Preprocesamiento de datos

Como parte inicial de la etapa de pre-procesamiento de datos, se utilizo una base de datos *MongoDB*. Se uso *MongoDB* y no *Pandas*, dado que *Pandas* requiere cargar todo el conjunto de datos en memoria *RAM*. Si bien se podría lidiar con este problema, aumentando el tamaño de la memoria *Swap* (Linux) o la memoria virtual (*Windows*), podría ocasionar caídas de procesos y lentitudes innecesarias. En este caso, se selecciono una base de datos de tipo documento, la cual no necesita cargar todos los datos en memoria y por otro lado, existe la posibilidad de escalar la base de datos a mas nodos en caso de ser necesario. Ambos conjuntos de datos contienen varios archivos *csv*, los cuales vamos a llamar *tablas*. En la tabla 2.3 se especifican las columnas utilizadas:

File	Descripción
<i>movie_metadata</i>	Pertenece al conjunto de datos <i>TMDB</i> . Es la fuente de verdad de la cual se toman columnas referentes a características de una película.
<i>tags</i>	Pertenece al conjunto de datos <i>Movie Lens</i> . De esta tabla se tomaron los tags o palabras clave dadas de alta por los usuarios por cada película.
<i>ratings</i>	Pertenece al conjunto de datos <i>Movie Lens</i> . De esta tabla se tomaron las calificaciones de los usuario para las películas que fueron calificadas.

Tab. 2.3: *files* o tablas utilizadas en este trabajo, para construir un conjunto de datos unificado, el cual sirve como base para entrenamiento y evaluación de modelos.

Para iniciar, se realizo un *merge* o *join* de las tablas *ratings* y *tags* por las columnas *user_id* y *movie_id*, ya que tenemos dos columnas que representan interacciones de usuarios:

- *rating*: Pertenece a la tabla *ratings*.
- *tags*: Pertenece a la tabla *tags*.

En segundo lugar, se realizo un *merge* entre las tablas *ratings_tags_v1* y *movie_metadata* utilizando la columna *imdb_id*, la cual es identificador unívoco de una película en ambas tablas.

Finalmente, se obtienen dos tablas/*files* como resultado:

Filename	Descripción
<i>movies_v4.csv</i>	Contiene toda la información de las películas, incluidos todos los <i>tags</i> cargados por los usuarios que calificaron un película.
<i>ratings_tags_v1.csv</i>	Contiene tanto las calificaciones como los <i>tags</i> para cada usuario y película.

Tab. 2.4: *files* resultado del pre-procesamiento. Estos forman parte del conjunto de datos de entrenamiento y evaluación en este trabajo practico.

2.3.1. Tabla de interacciones

La tabla *ratings_tags_v1* tiene datos a nivel interacción usuario-ítem. De esta forma, a nivel usuario-ítem se cuenta con la calificación de la película realizada por el usuario, ademas de los *tags* que el usuario cargo para esa películas. Estos *tags* no son mas que una lista de palabras representativas de la película en cuestión. Por ejemplo, para la película *Toy Story* deberíamos tener palabras referente a la misma como: *boss*, *woody*, *animation*, *3d*, etc.. Finalmente, contamos con la fecha en la cual se realizaron esta interacciones. Se entiende que la calificación y los *tags* se ingresaron en el mismo momento.

Columna	Descripción
<i>user_id</i>	Existen 13,281 usuarios.
<i>movie_id</i>	Existen 33,444 películas.
<i>timestamp</i>	Fecha en la cual el usuario califico el ítem(<i>movie_id</i>). Es un <i>string</i> de formato años-mes. Existen valores entre 1997-09 y 2019-11 inclusive.
<i>rating</i>	Calificación. Es un valor discreto numérico: 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5 y 5.
<i>tags</i>	Lista de palabras definidas por cada usuario para una película. Se cuenta con los <i>tags</i> a nivel usuario-película.

Tab. 2.5: Definición de tabla *ratings_tags_v1* o tabla de interacciones.

2.3.2. Tablas de metadatos de películas

La tabla *movies_v4* 2.6 cuenta con información de cada película seleccionada en ambos conjuntos de datos.

Columna	Descripción
<i>title</i>	Título de la película.
<i>native_language</i>	Lenguaje original en el cual fue filmada la película.
<i>genres</i>	Ambos conjuntos de datos cuentan con una lista de géneros a los que adiere la película.
<i>overview</i>	Sinopsis de la película.
<i>poster</i>	Enlaces al detalle de la película en <i>imdb</i> y <i>TMDB</i> . Estos enlaces permiten hacer join con mas datos que se encuentren en la descripción de estos sitios. En este trabajo solo se utilizará la imagen de la tapa de la película a modo de visualización.
<i>release</i>	Fecha de lanzamiento.
<i>budget</i>	Presupuesto destinado para la realización del film.
<i>popularity</i>	Popularidad.
<i>vote_count</i>	Cantidad de votos por película.
<i>vote_mean</i>	Cantidad media de votos por película.
<i>tags</i>	Son los <i>tags</i> cargados por todos los usuarios que interactuaron con la película. Es la misma información que tenemos en la tabla de interacción pero ahora a nivel ítem.

Tab. 2.6: Definición de tabla *movies_v4* o tabla de películas.

2.3.3. Valores faltantes

Una vez generadas ambas tablas, se procedió a buscar *missing values* o valores faltantes. A continuación, en la tabla 2.7 se pueden ver las columnas con valores faltantes:

Columna	% Valores Faltantes
<i>budget</i>	70
<i>poster</i>	0.00085
<i>release</i>	0.0085
<i>popularity</i>	0.00085
<i>vote_mean</i>	4.8
<i>vote_count</i>	4.6

Tab. 2.7: Porcentaje de valores faltantes por columna en la tabla *movies_v4*.

Luego se removieron las filas de la tabla para aquellas columnas del reporte anterior que tuvieran hasta 6 % de valores faltantes. A continuación se removió la columna *budget* por tener un porcentaje muy alto de valores faltantes, lo que la volvió inutilizable. Por otro lado, la tabla *ratings_tags_v1* no se modificó, ya que no tenía valores faltantes en ninguna de sus columnas.

Por otro lado, encontraron valores faltantes en los campos *genres* y *overview*. Aunque no eran nulos, tenían marcadores de ausencia de información, como "(no genres listed)" en el campo *genres* y "No overview found" en el campo *overview*. Se identificaron 127 ítems sin *overview*, de los cuales se completaron 96, dejando 31 sin este dato. En el campo *genres*, había 286 ítems con valores faltantes; se llenaron 245, resultando en 41 sin género asignado. Existían 6581 ítems con un solo género, de los cuales 2531 se actualizaron a dos o más géneros, dejando 4050 con un único género. Para completar los valores faltantes en *overview* y *genres*, se creó un script que extrae datos de un servicio REST proporcionado por la API de TDMB. Los campos *title*, *overview* y *genres* son fundamentales para el proceso de búsqueda por cercanía(utilizando la técnica *RAG*(*Retrieval Augmented Generation* o Generación Aumentada por Recuperación) [7] en la base de embeddings de la API *Rec Chatbot*.

2.3.4. Base de interacciones para *API Rec Chatbot*

Hasta este momento, se contaba con las tablas *ratings_tags_v1.csv* y *movies_v4.csv*. Con base en estas tablas, se generó una base de datos inicial para *API Rec Chatbot*. Se extrajeron los datos necesarios para poblar dos colecciones en la instancia de base de datos *MongoDB* de *API Rec Chatbot*: *interactions* e *items*. Es importante mencionar que se realizó un *merge join* entre las tablas *ratings_tags_v1.csv* y *movies_v4.csv* generando la tabla *rec_chat_bot_data.csv* para asegurar que las interacciones incluyeran ítems con datos completos, como *title*, *overview*, *genres*, etc.

La tabla *rec_chat_bot_data.csv* contiene interacciones virtuales que se utilizarán para entrenar los modelos del *API Rec Chatbot*, sin estar asociadas a ningún usuario real. Es fundamental para el modelo de filtros colaborativos. Aunque se podría optar por utilizar un modelo pre-entrenado y aplicar *transfer learning* con datos reales, se eligió un enfoque más sencillo para facilitar el desarrollo. Así, la base de datos *MongoDB* incluirá interacciones tanto virtuales como reales.

2.4. Análisis exploratorio

Una vez realizado el procesamiento de datos, se procedió a realizar un análisis exploratorio de todas las variables relevantes del conjunto de datos construido previamente. A continuación se analiza cada variable relevante.

2.4.1. Variable *Rating*

A continuación se puede apreciar una diagrama de barras el cual describe la frecuencia con la que los usuarios califican un ítem, segmentada por cada posible valor de calificación:

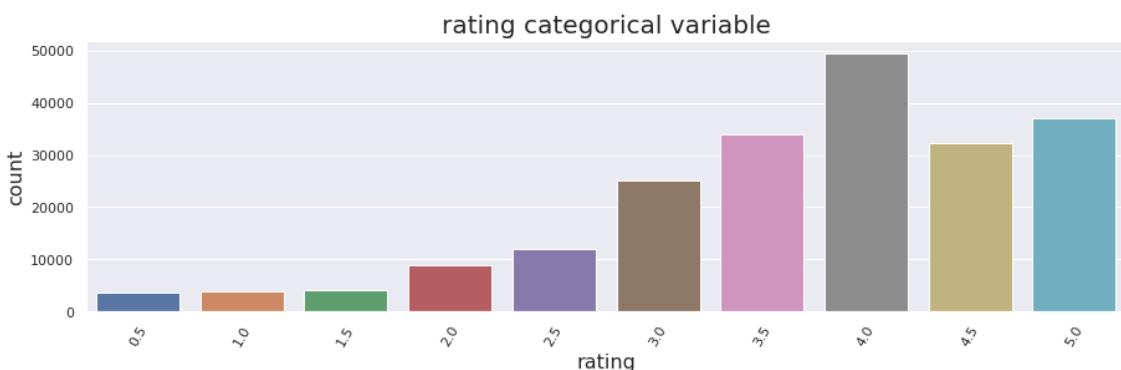


Fig. 2.1: Frecuencia o cantidad de observaciones para cada valor discreto de calificación o *rating*.

En la figura 2.1 se puede visualizar que el valor 4,0 tiene la mayor frecuencia (moda), seguido de 5,0 puntos y luego 3,5 puntos. Por otro lado, se debe tener en cuenta que estas calificaciones provienen de todo los usuarios. Cada usuario tiene una forma propia de calificar, algunos tienden a calificar de forma optimista, puntuando con valores altos, y otros por el contrario, son más pesimistas y tienden a puntuar con calificaciones bajas. Este es un comportamiento conocido en el ámbito de sistemas de recomendación. Se debe tener en cuenta que el valor 3,5 para un usuario podría ser un valor 4,5 para otro. Por otro lado, se aprecia que en general se tiende a puntuar valores a partir de 3,0 punto en adelante, habiendo muy pocas observaciones para puntuaciones menores a los 2,0 puntos.

Para analizar en mas detalle la variable *rating*, veamos a continuación un histograma y *boxplot* respectivamente:



Fig. 2.2: Histograma y *Boxplot* de la variable *rating*. Los *ratings* son las calificaciones realizadas por los usuario para cada ítem o película.

En la figura 2.2 se aprecia que la variable *rating* tiene valores discretos entre 0,5 y 5,0 con un paso de 0,5. De esta forma, contamos con 10 valores discretos de tipo real, siendo claramente una variable categórica. Nuevamente vemos algo parecido al diagrama de barras 2.1, el 50 % de las observaciones se encuentran entre los cuantiles Q_1 y Q_3 con 3,0 y 4,5 puntos (rango inter-cuantil). La mediana (Q_2) esta claramente sobre los 4,0 puntos, coincidiendo con la moda. La media se encuentra en los 3,5 puntos a izquierda de la mediana, debido a que tenemos puntos con frecuencia considerables a izquierda que mueven a la media en esa dirección. Por otro lado, tenemos valores atípicos en el extremo izquierdo en los 0,5 puntos. Esto se debe a que esta puntuación esta muy alejada del centro de los datos, el cual se encuentra entre el cuantiles Q_1 y Q_3 , donde tenemos el 50 % las calificaciones con mayor probabilidad de ocurrencia. No se encuentran valores atípicos por sobre el máximo. Se puede apreciar un sesgo a izquierda, ya que existe mayor separación o dispersión de las observaciones entre Q_1 y Q_2 que entre Q_2 y Q_3 . De esta forma, ambos intervalos conservan su 25 % de las observaciones pero hay menor dispersión entre Q_2 y Q_3 .

A continuación segmentemos el anterior histograma por año:

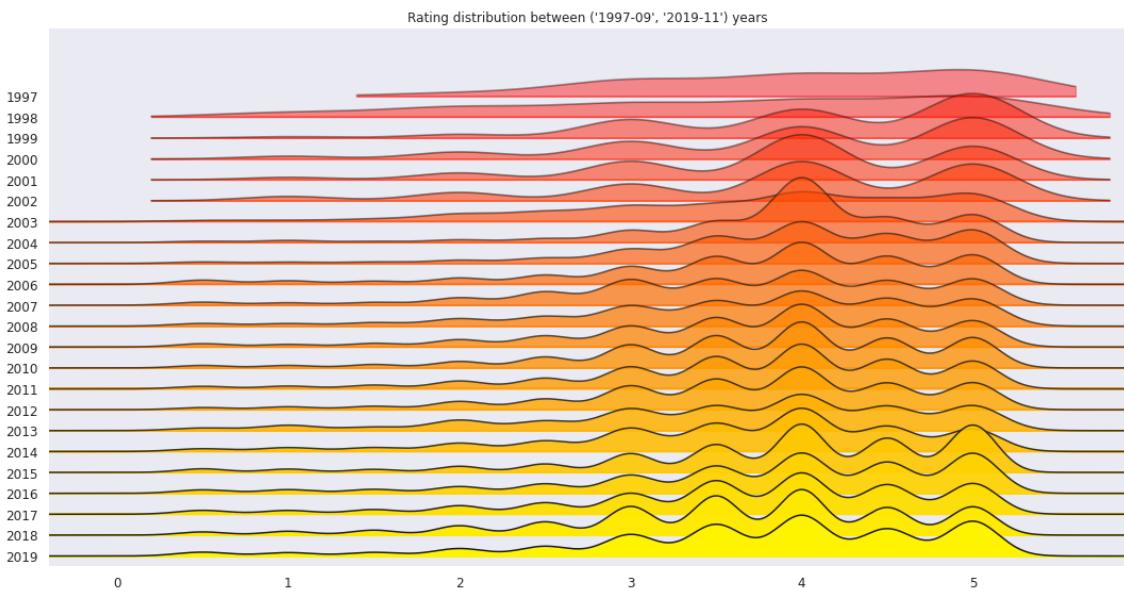


Fig. 2.3: Histograma de calificaciones segmentado por año. En esta gráfica se muestran histogramas como funciones de densidad, a pesar de que la variable *rating* es categóricas. Las funciones de densidad se graficaron utilizando el método de estimación *KDE* o *Kernel Density Estimation*. Si bien este método se utiliza para estimar la función de densidad de una variable aleatoria continua, en este caso permite apreciar con mayor claridad la diferencia en la cantidad de observaciones y el grado de dispersión de cada niveles de la variable *rating* por año.

En la figura 2.3 inicialmente vemos que en los años 1997, 1998 y 2003 la curva tiende a ser mas lineal. Esto indica que la forma de calificar es mas dispersa, es decir, no se encuentra un perfil de puntuación claro por parte de los usuarios (si un item es un 4 o un 5 por ejemplo). Entre 1999 y 2022 vemos que las puntuaciones 3, 4 y 5 toman mayor importancia siendo estas las mas utilizadas. Es decir, los usuarios realizan en su mayoría puntuaciones en esos tres niveles. La mayor frecuencia se puede ver claramente en los 4 puntos en el año 2004, donde fue prácticamente la mas utilidades decayendo los 3 y 5 puntos, en comparación a años anteriores. A partir del año 2005 se nota un aumento cada vez más demarcado en los niveles de puntuaciones entre los 3 y 5 puntos, donde los usuarios cada vez más usan los niveles 3,5 y 4,5. Debemos tener en cuenta que el aumento en los niveles de puntuación con el tiempo, probablemente sean debidos a un aumento año a año en la base de usuarios de *Movie Lens* y tal vez también sea el motivo por el cual en los primeros años vemos mucha dispersión en las puntuaciones.

2.4.2. Correlaciones

Para realizar un analizar de correlación sobre todas la variables, se realizo un *merge-join* de las tabla *movies* e *interactions*, incluyendo solamente las columna numéricas. A continuación podemos visualizar un diagrama de correlación de *Pearson* de las mismas:

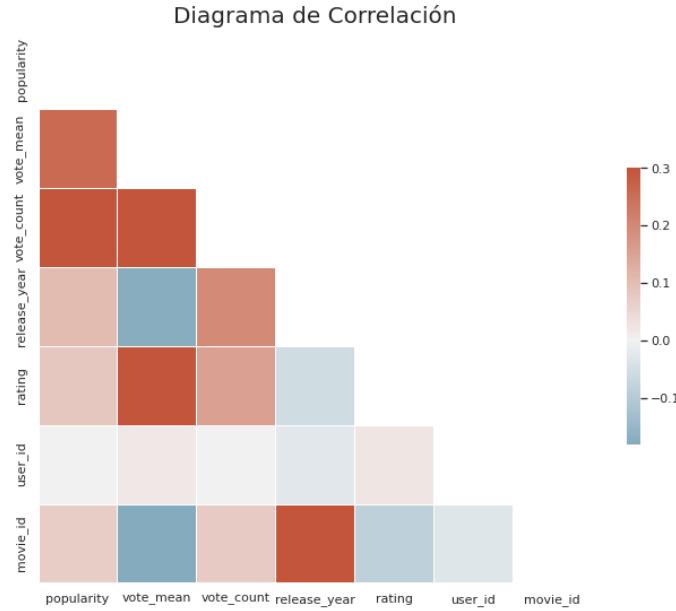


Fig. 2.4: Diagrama de correlación de *Pearson* aplicado a todas las variables numéricas resultado del *merge* entre las tablas *movies* e *interactions*.

En la figura 2.4 se aprecia de las variables *vote_count/vote_mean* (Cantidad de votos/-Media de votos) y *popularity* (Popularidad) tiene alta correlación debido a que las películas mas votadas en general son las mas populares. Las variables *vote_count* y *vote_mean* están altamente co-relacionadas entre si, ya que la media se calcula en base a la variable *vote_count*. Por otro lado, también es de esperar que las variables *rating* (Calificaciones) y la *vote_mean* estén co-relacionadas, ya que a medida de aumenta la *vote_mean* tenemos calificaciones mas altas. Se encuentra una alta correlación entre la variable que identifica a una película y la variable *release_year* (Fecha de estreno). Esto se debe a que al momento de estrenarse una película, días después a mas tardar, se da de alta la película en el sitio de *Movie Lens*. Esto también nos dice que los identificadores son secuenciales. Las correlaciones en general son muy bajas llegando a 0,3 como máximo. Esto es una buena señal, ya que ayuda a diminuir el fenómeno de co-linealidad de las variables. Las variables que son combinaciones lineales de otras variables puede producir que los modelos de *Machine Learning* sobre-ajusten a los datos de entrenamiento. Las variables *vote_mean* y *movie_id* (Identificador de película) tienen una correlación negativa muy baja. En algún sentido nos dice que algunas películas mas nuevas tiende a tener una media de votos menor. Lo mismo sucede entre las variables *rating* y *movie_id* en menor medida. Ambas con correlaciones negativas muy bajas.

Si bien, en esta primera entrega no se están utilizando otras variables distintas a *user_id* (Identificador del usuario), *movie_id* y *rating*, es de interés analizar las variables correspondiente a características de películas, ya que en el siguiente entrega (tesis) se planea

implementar modelos de recomendación híbridos, los cuales son ensambles de sistemas de recomendación basados en contenido y filtros colaborativos.

2.4.3. Variables de tipo texto

Las variables de tipo texto son muy útiles para generar *embeddings*, los cuales se puede sumar o promediar, generando representaciones compactas de ítems a recomendar. Luego es posible utilizar estas representaciones para agrupar ítems según su sinopsis, *tags*, título u otras variables como pueden ser popularidad, año de estreno, votos, *rating*, etc...

A continuación se puede visualizar la frecuencia de aparición de cada palabra en cada uno de los campos de tipo texto. Cuanto mayor tamaño tenga la palabra mayor frecuencia de aparición tendrá.

Tags

Cada usuario puede asociar *tags* o palabras clave a una película. Un *tag* es una palabra o frase corta que identifica o describe una película. Es una forma simple de agrupar, ya que los usuarios en general describen categorías como animación 3d, cual es el director, una frase que describe algún tipo de categoría o algo relevante en el contenido de la misma.

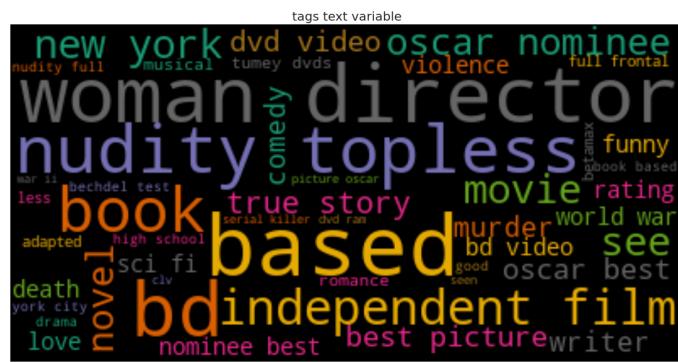


Fig. 2.5: Frecuencia de frases encontradas en la variable *Tags*. El tamaño de cada frase representan la cantidad de apariciones de la misma.

En la figura 2.5 las palabras *based* (basada/o en), *Book*, *Bd*, *Director*, *Nudity*, *Topless*, *Independent*, *Film*, son aquella palabras mas cargan por los usuarios en cada película. Otras palabras de menor frecuencia: *Movie*, *Murder*, *Novel*, *Funny*, *Oscar nominee*, *Violent*, *World war*, *True story*.

Overview

La variable *overview* es la sinopsis de la película. Esta describe brevemente el contenido de la misma, sin revelar su desenlace.

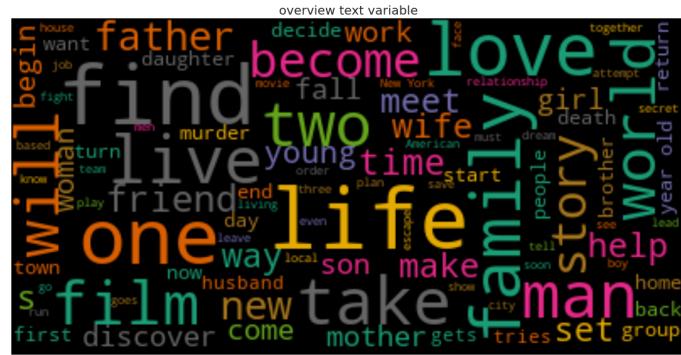


Fig. 2.6: Frecuencia de palabras encontradas en la variable *Overview*. El tamaño de cada palabra representan la cantidad de apariciones de la misma.

En la figura 2.5 podemos visualizar las siguientes palabras con mayor frecuencia: *Find, Live, Love, Wife, One, Man, Film, Become, Family, Story, World, Work, Father, New, Friend* y *Story*. Se aprecia una diferencia notoria entre las variables *tags* y *overview*(Sinopsis). La variable *tags* parece categorizar las películas desde distintas perspectivas. Por otro lado, la variable *overview* contiene palabras que son mas utilizadas en la descripción de la misma. A simple vista, los clusters (*Embeddings*) que pudieran generarse utilizando esta variable, podrían ser mas específicos que aquellos *clusters* generados a partir de la variable *tags*.

Title

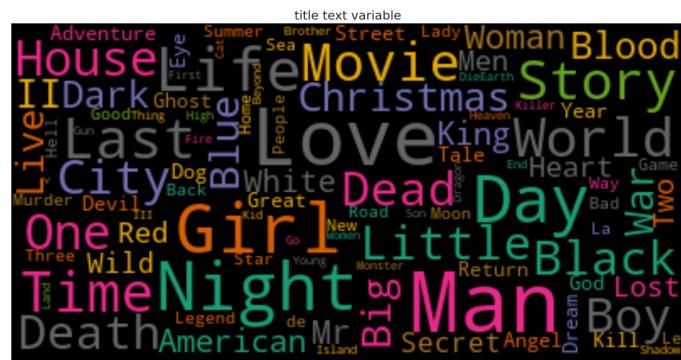


Fig. 2.7: Frecuencia de palabras encontradas en la variable *Title*. El tamaño de cada palabra representa la cantidad de apariciones de la misma.

En la figura 2.5 podemos visualizar las siguientes palabras con mayor frecuencia: *Girl*, *Man*, *Day*, *Dead*, *Movie*, *Time*, *Night*, *Life*, *House*, *Dark*, *II*, *Blood*, *Christmas*, *World*, *War*, *Black*, *Boy*, *Blue*, *One* y *King*. A simple vista, una clusterización realizada con esta variable puede ser mas general que la lograda con la variable *overview*, pero menos general que la variable *tags*. En trabajos posteriores se realizaran experimentos para ver resultado en este sentido.

2.4.4. Análisis de Componentes Principales

En esta sección se describe el análisis de componentes principales realizado sobre las variables numéricas, resultado de fusionar las tablas *movies* y *interactions*.

Varianza Explicada

Las componentes principales son las variables resultado al aplicar el algoritmo *PCA* [10]. Estas nuevas variables tienen la particularidad de ser ortogonales entre sí, lo que implica que no poseen ninguna correlación. Además, a lo largo de todas las variables, la acumulación de varianza disminuye. Esto indica que la primera componente tiene la mayor varianza y la última la menor posible (en orden descendente).

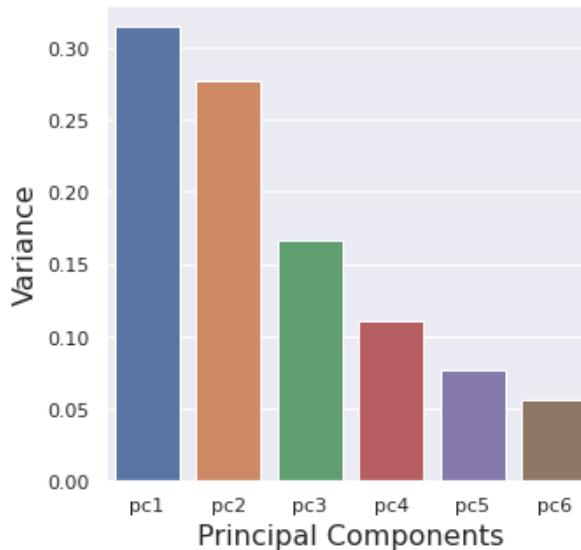


Fig. 2.8: Este diagrama de barras describe el grado de variabilidad o varianza explicada para cada componente principal resultado de aplicar el algoritmo *PCA* [10] sobre el conjunto de variables numéricas originales.

Varianza

- pc1: 31 %
- pc2: 28 %
- pc3: 16 %
- pc4: 11 %
- pc5: 7 %
- pc6: 5 %

En la figura 2.8, inicialmente podemos apreciar que todas las componentes tienen niveles de variabilidad o varianza explicada muy bajos, donde la primera componente llega solamente al 31 %. Esto indica que el grado de correlación de las variables originales es muy bajo. Utilizando el criterio del bastón roto podríamos seleccionar las 3 primeras componentes principales, ya que son las que acumulan mayor varianza.

Por otro lado, debemos tener en cuenta que el análisis por componentes principales es un análisis lineal. Es decir, tiene encuentra únicamente correlaciones lineales entre las variables originales. De esta forma, el método puede estar perdiendo de vista correlaciones no lineales mas complejas, donde podríamos encontrar un mayor grado de correlación. Las 3 primeras componentes acumulan un grado de variabilidad del 85 %.

Cargas o Loadings

Las componentes principales son combinaciones lineales de las variables originales. Luego, las cargas o *loadings* son los coeficientes utilizados para transformar las variables originales en las componentes principales mediante combinaciones lineales.

De esta forma, los coeficientes definen una medida de correlación o grado de aporte de cada variable original a una componente principal.

A continuación se pueden visualizar las cargas o *loadings*:

Variable	PC1	PC2	PC3
Popularity	0.79	-0.09	-0.003
Vote Mean	0.55	-0.55	-0.03
Vote Count	0.88	-0.11	-0.0002
Release Year	0.33	0.8	0.045
User ID	-0.006	-0.08	0.99
Movie ID	0.24	0.8	0.04

Tab. 2.8: Coeficientes de componentes principales vs variables originales. Cada uno de estos valores representan el grado de correlación o aporte de cada variable original a cada componente principal.

En la tabla 2.8 vemos que *Vote Count* (88 %) (Cantidad de votos) y *Popularity* (80 %) (Popularidad) tiene una correlación positiva muy alta sobre la componente *PC1*. Lo mismo sucede con *Vote Mean* (55 %) (Media de la cantidad de votos) en menor medida. Entre dos observaciones con distintos valores de popularidad, la que tenga un valor mas alto, aportara mas a la componente *PC1* que a las otras dos (*PC2* y *PC3*). También vemos que las variables *Release Year* (Año de estreno) y *Movie ID* tienen un aporte, considerable, pero mas bajo del 33 % y 24 % respectivamente, sobre la componente *PC1*. La variable *User ID* no tiene aporte alguno sobre la componente *PC1*. Las variables que mas aportan a la componente *PC2* son *Vote Mean* (55 %) y *Vote Count* (11 %) respectivamente. Este aporte es negativo, esto indica que un aumento en los niveles de esta variable significa una disminución en la componente *PC2*. La variable *Release Year* (Año de estreno) tiene el aporte positivo mas alto sobre la componente *PC2* siendo del 80 %. Para la componente *PC3*, las variables que mas aportan son *User ID* (99 %) y *Release Year* (45 %) respectivamente, ambas positivas. Vemos que un aumento en la variable *User ID* produce un aumento casi en una unidad sobre el coeficiente, pero *Release Year* es la mitad en relación. De este forma, la componente *PC1* podríamos nombrarla como Nivel de popularidad o conocimiento general de una película. La componente *PC2*, en algún sentido mide lo contrario a la popularidad, es un indicador de cuan *underground* es un nuevo estrenos. La componente *PC3* es mas difícil de nombrar, pero podría llamarse: Grado *newbie* de un usuario, indicando cuan nuevo es un usuario.

Biplot

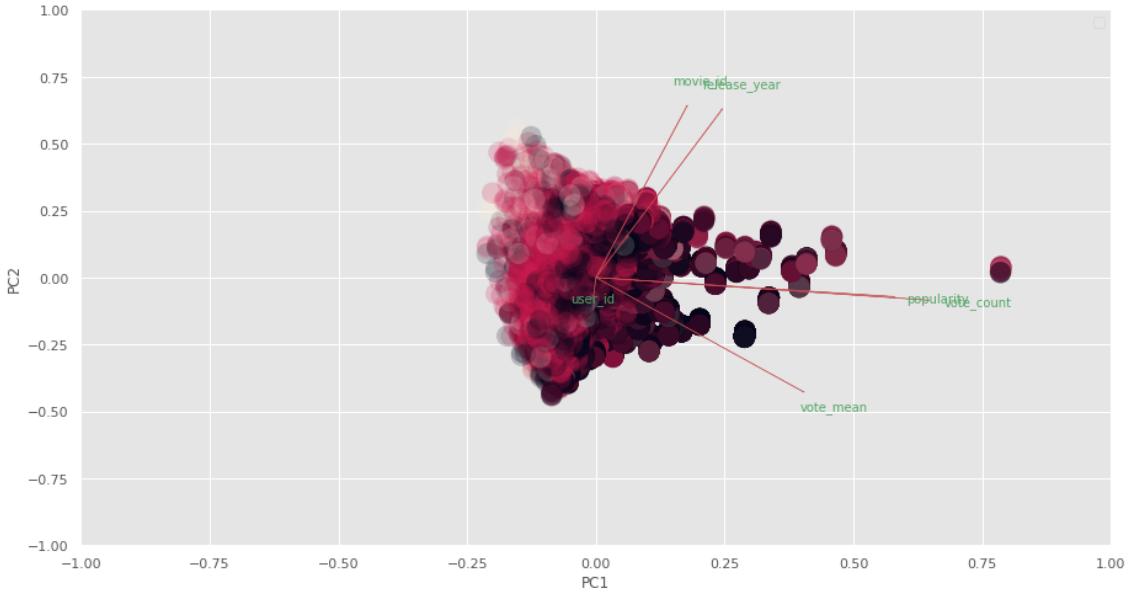


Fig. 2.9: Diagrama [11] Biplot. Este diagrama representa los valores de las variables originales coloreados en color rojo, negro y gris. Estos colores corresponden a tres segmentos de calificaciones: > 2 , entre 2 y $3,5$ y > 4 . También se pueden apreciar los vectores correspondientes a las variables originales.

En la figura 2.9 se expone un diagrama Biplot [11]. Este representa los valores de las variables originales coloreados en color rojo, negro y gris correspondientes a tres segmentos de calificaciones: > 2 , entre 2 y $3,5$ y > 4 . También se representan los vectores correspondientes a las variables originales. A primera vista observamos que las variables *Popularity* (Popularidad) y *Vote Count* (Cantidad de votos) tienen una correlación muy alta, ya que el ángulo entre sus vectores es prácticamente cero. Esto tiene sentido, ya que ambas son medidas de popularidad de alta co-linealidad. Las variables *Popularity* (Popularidad), *Vote Count* (Cantidad de votos) y *Vote Mean* (Media de la cantidad de votos) tiene un aporte positivo sobre la componente *PC1* (en menor medida). Esto último se corresponde con los coeficientes de las cargas analizados anteriormente. Las variables *Movie ID* y *Release Year* (Año de estreno) aportan en menor medida sobre la componente *PC1*. De esta forma, se constata lo visto anteriormente en análisis de carga, donde la componente *PC1* representa el grado de popularidad de una película. La variable *Vote Mean* (Media de la cantidad de votos) aporta en forma negativa y las variables *Movie ID* y *Release Year* (Año de estreno), en forma positiva sobre la componente *PC2*. Las variables *Popularity* (Popularidad), *Vote Count* (Cantidad de votos) tienen casi aporte nulo a la componente *PC2* en correspondencia con el análisis de cargas anteriormente expuesto. Si visualizamos los puntos que representan a las observaciones originales en el espacio latente generado por PCA [10], vemos que las observaciones de color negro (> 4 puntos) y gris (entre 2 y $3,5$ puntos) se encuentran mas a la derecha que aquellas coloreadas en rojo (> 2 puntos). Esto indica que hay un crecimiento del nivel de popularidad cuanto mas a derecha se encuentre un punto en la componente *PC1*, validando los análisis anteriores. Si visualizamos los puntos correspondientes a las observaciones en las direcciones de la componente *PC2*, vemos

que a mayor valor en la componente, menor es el grado de popularidad de las películas, ya que los puntos rojos tienden a estar en el extremo positivo de la componente. Esto valida la hipótesis de que la componente $PC2$ indica cuan *underground* es un película.

3. ARQUITECTURA GENERAL DE *REC CHATBOT API*

Para llevar a cabo este trabajo de investigación, se tomó la decisión de modificar el enfoque tradicional que comúnmente se utiliza para investigar en este ámbito. Por lo general, la metodología típica implica el desarrollo de una librería o paquete en *python*, que resulta ser accesible tanto desde un entorno de *Jupyter* como desde cualquier aplicación que haya sido creada utilizando dicho lenguaje de programación. Sin embargo, en este caso particular, se optó por una alternativa diferente: se decidió crear una *API* que opere bajo un contrato *REST*. Esta elección presenta diversas ventajas significativas en comparación con el desarrollo convencional de una librería o paquete:

- **Independencia del lenguaje:** Una *API REST* puede ser consumida por cualquier lenguaje de programación que soporte *HTTP*, no solo *Python* o el lenguaje con el que este escrito el paquete.
- **Escalabilidad:** Las *APIs REST* son más fáciles de escalar horizontalmente, distribuyendo la carga entre múltiples servidores.
- **Separación cliente-servidor:** Permite una clara separación entre el *frontend* y el *backend*, facilitando el desarrollo y mantenimiento de forma independiente.
- **Integración con sistemas distribuidos:** Facilita la integración con otros servicios y sistemas distribuidos.
- **Caching:** Se puede implementar caches fácilmente a nivel *HTTP* para mejorar el rendimiento.
- **Seguridad:** Permite implementar mecanismos de autenticación y autorización estándar como *OAuth* entre otros.
- **Documentación:** Existen herramientas para generar documentación automática de *APIs REST* (como *Swagger*, *Redoc*, *OpenAPI*, etc..).

Luego, en la figura 3 se pueden apreciar las piezas que componen la arquitectura general de la *API REST* (desde ahora *API Rec Chatbot*).

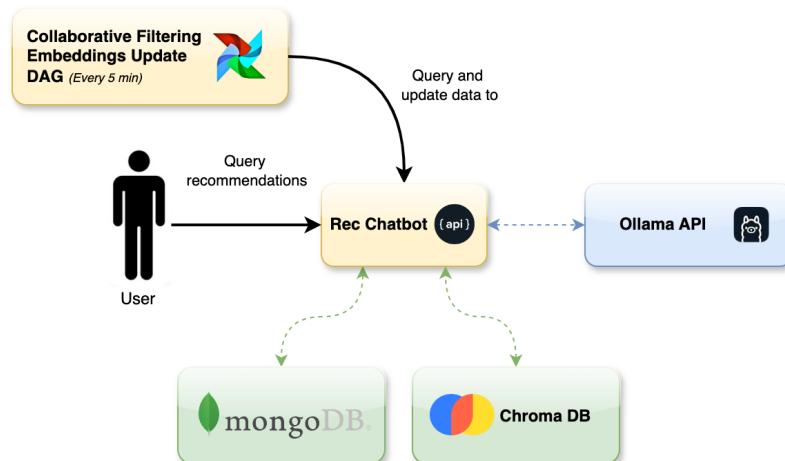


Fig. 3.1: Arquitectura general: Se puede apreciar un diagrama de despliegue donde se especifica cada componente de la arquitectura general y la interacción entre los mismos.

En este contexto, puede apreciarse un diagrama de despliegue que ilustra de manera detallada cada uno de los componentes que conforman la arquitectura general y la interacción que existe entre ellos. A grandes rasgos, al inicio del proceso, el usuario realiza una solicitud de recomendaciones utilizando lenguaje natural, lo que inicia una serie de operaciones en el sistema. En este punto, el componente conocido como *API Rec Chatbot* se encarga de buscar ítems que sean relevantes y cercanos a la petición del usuario en una instancia de *ChromaDB*. Una vez que se han encontrado los ítems pertinentes, el sistema procede a realizar un ordenamiento de estos elementos. Este paso es clave, ya que se utilizan tanto los modelos de lenguaje que son proporcionados por *API Ollama* como también los embeddings y las predicciones que son generadas por los modelos de filtros colaborativos, los cuales han sido específicamente entrenados a través de un proceso que se ejecuta en la instancia de *Apache Airflow* [1]. Este último proceso asegura que los resultados sean eficaces y relevantes para el usuario. Además, es importante destacar que todos los datos que están relacionados con los perfiles de los usuarios, así como las calificaciones que estos otorgan, las predicciones que se generan, entre otros aspectos importantes, se persisten en una instancia de base de datos *MongoDb*. Esta persistencia de datos es crucial para el funcionamiento continuo y la mejora del sistema, ya que permite realizar un seguimiento de las interacciones y preferencias de los usuarios a lo largo del tiempo.

3.1. Componentes

Por un lado, el *API Rec Chatbot* desarrollada en este trabajo, permite gestionar perfiles de usuarios, registrar su historial de calificaciones de películas y ofrecer recomendaciones a través de sus solicitudes en lenguaje natural.

Por otro lado, el *API Rec Chatbot* esta integrada con el *API Rest Ollama* que facilita realizar consultas a una batería de grandes modelos de lenguaje. El *API Rec Chatbot* se enfoca mayormente en la inferencia de recomendaciones mediante un ensamblaje de modelos o algoritmo de inferencia que combina grandes modelos de lenguaje y modelos de filtros colaborativos en diversas partes de su flujo. Mas adelante, se explicará detalladamente el flujo de inferencia y entrenamiento de la solución.

Para la gestión y operación de *LLM* locales, se eligió la plataforma *Ollama*. Esta herramienta permite a los usuarios descargar modelos *LLM* pre-entrenados que son especialmente receptivos a instrucciones en lenguaje natural, lo cual es crucial para desarrollar aplicaciones interactivas. Una característica destacada de *Ollama* es su capacidad para servir localmente una interfaz *Rest*, que brinda a los desarrolladores acceso fácil y eficiente a todos los modelos de lenguaje descargados.

En cuanto al modelo de datos se cuenta separados en dos base de datos:

- La primera de estas es una base de datos *MongoDB*. Esta base de datos se encarga de almacenar información relacionada a los perfiles de los clientes que interactúan con la *API Rec Chatbot*. A su vez, también guarda ítems, que en este contexto se refieren a las películas disponibles en el sistema, así como las calificaciones proporcionadas por los usuarios sobre las mismas. Estas calificaciones pueden abarcar tanto datos reales, aquellos introducidos por los propios usuarios, como predicciones generadas por el sistema. De esta forma, se busca ofrecer un servicio personalizado y adaptado a las preferencias de cada usuario.
- La segunda es un base de datos *ChromaDB*. Esta base de datos tiene una función específica en el almacenamiento de *embeddings*, que son representaciones vectoriales tanto de los usuarios como de los ítems, ambas generadas por un modelo de filtros colaborativos. Además se almacenan *embeddings* que reflejan el contenido de cada ítem, en este caso, las películas, lo que a su vez habilita la utilización de la técnica conocida como *RAG(Retrieval Augmented Generation o Generación Aumentada por Recuperación)* [7]. Esta técnica permite realizar búsquedas de ítems que son similares a las consultas realizadas por los usuarios.

Por último, se utilizo una instancia de *Apache Airflow*. *Apache Airflow* permite la orquestación de flujos de trabajo para programar, ejecutar y monitorear secuencias complejas de tareas. Estos procesos se representan como grafos dirigidos acíclicos, lo que facilita dividir un proceso en tareas más simples y re-utilizables. Esto es fundamental para re-entrenar el ensamblaje de modelos del *API Rec Chatbot* y generar las predicciones y embeddings necesarios para su funcionamiento. Además, debido a la significativa cantidad de usuarios, ítems e interacciones en el conjunto de datos, esta arquitectura es esencial para garantizar el correcto funcionamiento del *API Rec Chatbot* y evitar problemas de escalabilidad. En resumen, se trata de una forma de desarrollar aplicaciones que es comúnmente utilizada por empresas de servicios en Internet.

4. FLUJO DE INFERENCIA DE *REC CHATBOT API*

En la sección anterior se presentó la arquitectura general de la solución, detallando cada componente mediante un diagrama de despliegue, lo que ayuda a entender de forma más simple la solución. En esta sección, se profundizará en cada uno de estos componentes y explicaremos el funcionamiento de la solución. Para ello, dividiremos la funcionalidad de *API Rec Chatbot* en dos flujos:

- Flujo de entrenamiento de modelos y generación de datos: Necesarios para la inferencia de *API Rec Chatbot*.
- Flujo de inferencia sobre las consultas del usuario: En este flujo se explica el paso a paso, desde que el usuario realiza una consulta, hasta que obtiene una recomendación como respuesta.

Comencemos con el flujo de inferencia. En la siguiente figura 4, se puede apreciar un diagrama donde se expone el flujo de inferencia. Es decir, cada vez que un usuario realizar una consulta a *API Rec Chatbot* se realizan todas las acciones especificadas en este flujo.

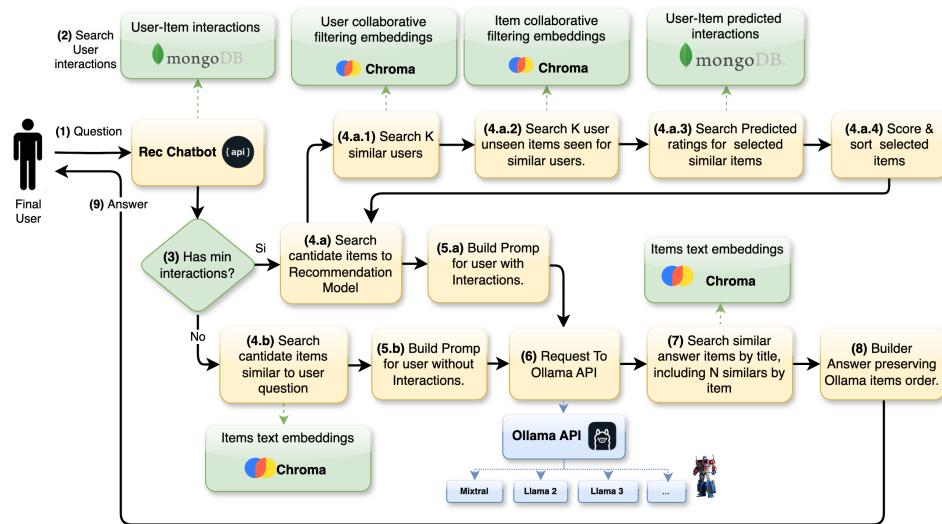


Fig. 4.1: Flujo de inferencia: Este flujo especifica detalladamente todos los pasos secuenciales que lleva a cabo la *API Rec Chatbot* para resolver una recomendación planteada por un usuario utilizando lenguaje natural. Cada etapa en este proceso es crucial para garantizar que la respuesta proporcionada sea adecuada y esté alineada con las necesidades del usuario.

Antes de comenzar, aclaramos que al referirnos al usuario final hablamos de aquel que interactúa con una aplicación, ya sea móvil o de escritorio. En este caso, no contamos con una interfaz gráfica, ya que esto va más allá de los objetivos de este trabajo. Por tanto, el término usuario final se refiere a quienes podrían usar una aplicación móvil o de escritorio que consume o integre la *API Rec Chatbot* como parte de su funcionalidad. También existen usuarios más avanzados que interactúan con la *API* a través de un cliente *HTTP*, como *curl*, *Postman* o *Insomnia*. Estos últimos son usuarios con conocimientos

sobre cómo interactuar directamente con una *API REST*. En este trabajo, se decidió interactuar directamente con la *API* para simplificar el desarrollo.

Aclarado esto, e comenzara a explicar el flujo de inferencia.

4.1. Solicitud de recomendaciones

Al comienzo del proceso (Paso 1 de la figura 4), el usuario tiene la opción de utilizar una aplicación móvil, una aplicación de escritorio o incluso acceder directamente a través de un cliente http. Mediante cualquiera de estos métodos, el usuario puede formular una pregunta en lenguaje natural relacionada con algún contenido que desea visualizar. En este caso específico, se trata de una película, pero también puede referirse a contenido sobre un género particular, una saga específica, o una historia que cuente con cierto tipo de trama, ya sea de acción, drama, comedia, terror, entre otros. Así, el usuario busca obtener información detallada que le ayude a decidir qué ver o comprender mejor el material que le interesa.

4.2. Consulta de interacciones del usuario

En el paso 2 de la figura 4, se llevan a cabo la consulta de las interacciones del usuario que formuló la pregunta, accediendo a la base de datos *MongoDB*. Para identificar de manera única al usuario en cuestión, se utiliza su *email* como identificador exclusivo, lo que permite obtener información detallada sobre sus interacciones previas.

4.3. ¿El usuario realizó el mínimo de interacciones?

En el paso 2 de la figura 4, se cuenta el número de interacciones previas del usuario con la aplicación. Si alcanza el mínimo requerido, se pueden iniciar recomendaciones personalizadas; de lo contrario, se utilizará un esquema de recomendación que puede sugerir contenido sin conocimiento profundo del usuario, ya que no conoce su comportamiento al momento de calificar contenido. Es importante aclarar que las interacciones se refieren a las calificaciones que el usuario otorga a los contenidos. Cada vez que el usuario puntúa una película (por ejemplo, con 5 estrellas), el servicio registra esta interacción y la utiliza para entrenar un modelo de recomendación que personaliza el contenido. Para lograr esto, es necesario contar con un mínimo de interacciones del usuario, lo cual mejora la precisión de las recomendaciones. Este mínimo es un hyper-parámetro más del modelo ensamblé o servicio desarrollado. Este paso permite manejar el problema de arranque en frio (*Cold Start*) que sufren los sistemas de recomendación basados en filtros colaborativos. Esto refiere a que estos sistemas de recomendación no pueden entrenarse sin un número mínimo de puntuaciones del usuario. Cuando no se cuenta con esta información es necesario utilizar otros sistemas de recomendación que permitan sugerir contenido hasta que el usuario llegue a este numero mínimo de puntuaciones o calificaciones.

4.4. El usuario no cumple con el número mínimo de interacciones

Cuando el usuario no cumpla con el número mínimo de interacciones (Paso 4.b de la figura 4), se toma su consulta en lenguaje natural y se lleva a cabo la generación de un *Sentence Embedding*, es decir se genera un vector de números que representa al texto de

la consulta del usuario. A continuación, con la consulta del usuario ya codificada como un *embedding*, se procede a buscar los *embeddings* más cercanos en la base de datos *ChromaDb*. Este motor de base de datos presenta algunas similitudes con el motor de base de datos *MongoDb*, ya que ambos sistemas organizan los datos en colecciones de documentos. Para proseguir, se realiza la búsqueda dentro de una colección particular llamada *items_content*, esta colección alberga los *embeddings* que corresponden a todas películas disponibles. Es fundamental aclarar que los *embeddings* se generan a partir de información relativa al título de la película, la sinopsis que la describe y también la lista de géneros cinematográficos a los que pertenece. Estos *embeddings* son producidos mediante modelos de tipo *transformer*; en este caso particular se utiliza el modelo *all-mpnet-base-v2*. Es relevante destacar que estos modelos han sido pre-entrenados específicamente para realizar la tarea de *clustering*, y es importante tener en cuenta que no todos los modelos de lenguaje ofrecen buenos resultados para esta tarea, a menos que hayan sido entrenados específicamente con este propósito.

4.5. Construir *prompt* para consulta al modelo de lenguaje

Después de generar una lista de ítems similares a la consulta del usuario en el paso anterior, es crucial en este punto (Paso 5.b de la figura 4) ordenar estos ítems según un *score* específico para identificar los más relevantes para el usuario. Luego, se seleccionarán los primeros n ítems de esta lista ordenada. Para esta tarea, se pueden usar diversos métodos, ya sean basados en contenido o en filtros colaborativos. Sin embargo, dado que no hay interacciones previas del usuario, se optó por un enfoque basado exclusivamente en el contenido disponible.

Este enfoque es, en cierto modo, poco convencional, ya que inicia con la aplicación de técnicas de *clustering* para identificar contenido que es similar a la consulta realizada, lo cual nos ayuda a reducir el espacio de búsqueda y a centrar nuestra atención en las opciones más relevantes. Después de haber agrupado el contenido similar, se recurre a un modelo de lenguaje grande que tiene la tarea de ordenar los ítems que se consideran más cercanos a las preferencias del usuario en función de los datos disponibles. Para que el modelo pueda cumplir con esta función de manera efectiva, es necesario crear un *prompt* que contenga información sobre el usuario conocida previamente a su consulta, así como una lista de ítems candidatos que ya fueron identificados en el paso anterior (Paso 4.b de la figura 4).

Entonces, se le solicita a un modelo de lenguaje grande que realice la tarea de ordenar los ítems en función de las preferencias del usuario. Es importante resaltar que durante el proceso de experimentación se probaron múltiples tipos de *prompts* con el objetivo de encontrar la estrategia más efectiva. Sin embargo, se observó que los modelos locales mostraron una menor precisión al intentar seguir instrucciones complejas proporcionadas en estos *prompts*. Esta situación llevó a tomar la decisión de utilizar *prompts* más simples, lo cual resultó en una mejora notable en la coherencia y rendimiento del modelo de lenguaje grande al momento de realizar el ordenamiento solicitado.

4.5.1. *Prompt* para usuarios que no cumple el mínimo de interacciones

A continuación se puede apreciar la plantilla del *prompt* para tener un idea de como se estructura:

```

1 PROMPT_LOW_INTERACTIONS = """
2 {user_profile}
3 {user_history}
4 {request}
5 {candidates}
6
7 Recommend {limit} movies from candidate movies list, based on 'question', 'user profile'
8 and 'seen movies'. Return only a list of recommendations with next format:
9
10 Number. Title (release year): Synopsis.
11
12 Response only has specified content and must have all specified items.
13 """

```

Para comenzar este es un fragmento de código fuente en lenguaje *Python*. Este código fuente define el *template* o plantilla utilizada para construir dinámicamente el *prompt* para usuarios que no tienen una cantidad mínima de interacciones. Cabe aclarar que existe un *prompt* para cada tipo de usuario: Aquellos que no cumplen con el número mínimo de interacciones y aquellos si lo hacen.

Este *prompt* esta compuesto principalmente por 5 secciones, definidas por un texto dinámico que se reemplaza en cada variable especificada entre corchetes.

4.5.2. Perfil del usuario

La variable *user_profile* contiene el perfil de usuario. Antes que nada se debe registrar el perfil del usuario, sin este no es posible comenzar a realizar consultas a la *API Rec Chatbot*. A continuación se puede apreciar una solicitud de alta de un perfil de usuario a *API Rec Chatbot*:

```

1 curl --location 'http://nonosoft.ddns.net:8080/api/v1/profiles' \
2 --header 'Content-Type: application/json' \
3 --data-raw '{
4   "name"      : "Adrian",
5   "email"     : "adrianmarino@gmail.com",
6   "metadata": {
7     "studies"    : "Engineering",
8     "age"        : 42,
9     "genre"       : "Male",
10    "nationality" : "Argentina",
11    "work"        : "Software Engineer",
12    "content_preferences": {
13      "release": { "from" : "1970" },
14      "genres" : [ "thriller", "suspense", "science fiction", "love", "comedy" ]
15    }
16  }
17}'

```

```

15 }
16 }
17 }'
```

Mediante el comando *curl* se realiza una solicitud POST para dar de alta el usuario en *API Rec Chatbot*, donde el *request body* contiene los siguientes datos:

- Nombre del usuario.
- *Email* del usuario: Este es utilizado como identificador único del mismo en el sistema.
- Edad, género, nacionalidad y una breve descripción laboral del mismo.
- En la sección *content_preferences* se puede establecer el año mínimo de lanzamiento de una película (*release_from*), determinando así la antigüedad a partir de la cual se prefieren recomendaciones. Además se puede definir una lista de géneros preferidos en la propiedad *genres*.

Con esta información se puede generar el texto que luego se reemplazara en la variable *user_profile* en el *prompt* principal.

4.5.3. Historial de comportamiento del usuario

La variable *user_history* presenta una lista detallada de los ítems que han sido calificados por el usuario junto a su calificación. Esto se refiere específicamente a aquellos elementos que el usuario ha llegado a conocer y ha evaluado en función de su experiencia personal. Los ítems están organizados de acuerdo con el *timestamp* de calificación, lo que significa que se presentan en un orden ascendente, comenzando desde el ítem más antiguo que fue calificado hasta llegar al más reciente. Esta disposición permite al lector observar la evolución de las calificaciones a lo largo del tiempo, ofreciendo una perspectiva clara sobre cómo han cambiado las opiniones del usuario sobre cada uno de los ítems evaluados.

4.5.4. Solicitud del usuario

La variable *request* contiene la solicitud del usuario en lenguaje natural. Ejemplo de esas podrían ser:

- Quisiera ver una película de Pixar. Cuales me recomendarías?
- Quiero ver una película de suspenso ambientada en un futuro disto-pico.

4.5.5. Items candidatos

La variable *candidates* contiene la lista de items candidatos resuelta en el paso 4.b ordenadas por su similitud con la variable *request*.

4.5.6. Instrucciones y formato de la respuesta

Finalmente se especifica las instrucciones que debe seguir el modelo de lenguaje:

- Debe seleccionar película a recomendar de la lista de items candidatos.

- Para realizar la recomendación, el *LLM* debe basarse en la solicitud del usuario, los datos referentes a su perfil y las películas que ya ha calificado. Con este paso se intenta que el modelo de lenguaje realiza una selecciones los títulos candidatos que mejor correlaciones con esta información.
- Se define explícitamente el formato de la respuesta, que consiste en una lista ordenada por un índice numérico que indica relevancia con el título, año de estreno y una breve sinopsis. Debido a la limitada capacidad de comprensión de las instrucciones, se optó por este formato óptimo. Aunque el formato más adecuado habría sido una lista de identificadores, el modelo a menudo no seguía estas indicaciones y devolvía una lista en el formato establecido, lo que aumentó la probabilidad de que el modelo lo respetara.
- Finalmente, se enfatiza al modelo de lenguaje la importancia de respetar el ítem de la lista de candidatos, ya que estos modelos tienden a incluir títulos que no son candidatos, nuevamente debido a su baja capacidad para seguir instrucciones.

4.6. Consulta a modelo de lenguaje

En el paso 6 de la figura 4, se lleva a cabo una solicitud *http* a la *API REST de Ollama*. Durante este paso, los usuarios tienen la opción de seleccionar la distribución del modelo de lenguaje que pueden utilizar. Esta selección es significativa, ya que cada distribución presenta diferentes hiperparámetros, así como variaciones en su tamaño e implementación, lo que puede influir en el rendimiento del modelo general que define *API Rec Chatbot*. Se llevaron a cabo pruebas exhaustivas con varios modelos, y entre ellos, Llama2 y Llama3 demostraron ser los que proporcionan resultados más coherentes y más estables , destacándose en comparación con otras alternativas evaluadas.

4.7. Búsqueda de ítems similares en base de datos *ChromaDb* y construcción de respuesta final

En el paso 7 y 8 de la figura 4, se toma la respuesta del *LLM* mencionado anteriormente. Esta respuesta es una lista ordenada de ítems, pero no se puede co-relacionar directamente con los ítems en la base de datos *MongoDb* porque no se proporciona un identificador de ítem. Para resolver esto, se buscan títulos por proximidad en la base de datos *ChromaDb*. Cada título se codifica en un *Sentence embedding* y se busca en la base de datos, generando una lista de N ítems para cada título de la respuesta original. Esto también permite recomendaciones por contenido al ofrecer ítems similares, enriqueciendo la respuesta cuando el modelo de lenguaje no es coherente o tiene menos ítems de los específicos. Este es un hiperparámetro que define cuántos ítems seleccionar. En la evaluación del modelo, se tomó solo un ítem similar, el más cercano a cada ítem de la respuesta original. Es importante señalar que el ítem seleccionado podría no ser el original, sino uno parecido. Esto se relaciona con el problema mencionado anteriormente, donde el modelo podría omitir instrucciones y devolver ítems que no estén en la lista de candidatos ni en la base de datos de *API Rec Chatbot*.

En resumen, este paso transforma la respuesta del *LLM* en una lista de ítems de las bases de datos de la *API Rec Chatbot*. Esta transformación es no determinista, ya que no siempre se obtienen el mismo número de ítems ni los mismos ítems, aunque sucede así la mayoría de las veces. Todo esto sucede dado que los modelos de lenguaje no puede cumplir

al pie de la letra con las instrucciones que le son dadas.

Finalmente, se consulta en la base de datos *MongoDB* los documentos correspondientes a cada ítem de la lista resultante. Se agrega el título, la *URL* de su póster (imagen) y se mantiene la descripción generada por el *LLM*, que es más resumida que la sinopsis original. También se toman los géneros del documento y se genera una lista de *URLs* que permite al usuario votar por los ítems recomendados. La lista de ítems se devuelve ordenada al usuario, quien puede votar por cada uno de ellos si lo desea. Estas votaciones representan la interacción del usuario con el sistema. Las votaciones se almacenan en la base de datos *MongoDB* en la colección *interactions*, donde cada documento contiene el identificador del usuario, el identificador del ítem, la votación en sí, y la fecha y hora de la votación. Posteriormente, estas interacciones serán consultadas por un proceso que entrena un modelo de recomendación basado en filtrado colaborativo, el cual se utilizará cuando el usuario alcance un número mínimo de interacciones. Esto se desarrollará mejor en los siguientes apartados, donde se detallará el perfil para usuarios con interacciones.

4.8. Usuarios de cumplen con el mínimo de interacciones

Hasta ahora, se ha hablado sobre el flujo para tratar a usuarios con pocas interacciones. ¿Qué sucede si ya tenemos un usuario que cumple con el mínimo? Primero, deberíamos contar con un modelo de filtros colaborativos entrenado que incluya a este usuario y sus interacciones hasta ese momento. Esto nos permite utilizar no solo el modelo de lenguaje seleccionado para elegir y ordenar los ítems candidatos, sino también el modelo de filtros colaborativos. Antes de comenzar, a describir el flujo se debe aclarar que el paso *4.a* en la figura 4 es igual al paso *4.b*, luego los pasos que si se diferencian serían *4.a.1* en adelante y el *5.a*. por otro lado, es importante entender que se seleccionaron modelos de filtros colaborativos analizados en el trabajo de especialización previo [8]. Entre ellos, se pueden utilizar aquellos que, tras su entrenamiento, no solo generan un modelo para predecir calificaciones a partir del ID de usuario e ítem, sino que también producen dos colecciones de *embeddings*, una para usuarios y otra para ítems. A medida que los modelos se entrena, identifican correlaciones entre los usuarios y entre los ítems, lo que se refleja en el ajuste de sus *embeddings* en cada iteración de entrenamiento. Esto permite no solo predecir calificaciones, sino también encontrar usuarios e ítems similares. En este trabajo de hará uso de estas capacidades al momento de la inferencia.

Para los pasos *4.a.1* y *4.a.2* de la figura 4 es necesario consultar dos colecciones extra de *embeddings*:

- ***items_cf***: Contiene los *embeddings* de todos los ítems del sistema. Estos *embeddings* son generados a partir de entrenamiento del modelo *DeepFM* [6, 12]. Este modelo aprende dos *embeddings* de usuarios e ítems en su entrenamiento.
- ***users_cf***: Contiene los *embeddings* de todos los usuarios del sistema aprendidos en el entrenamiento del modelo *DeepFM* [6, 12].

Cada colección de *embeddings* contiene metadatos que incluye el identificador único de cada entidad, ya sean usuarios o ítems, lo que facilita la búsqueda de entidades que sean similares o cercanas. A partir del identificador del usuario actual, se procede a buscar otros usuarios que sean similares, lo cual es definido por el hiperparámetro *k_sim_users*. Este proceso es crucial para encontrar recomendaciones personalizadas. Una vez obtenidos los usuarios similares, se lleva a cabo la selección de hasta *max_items_by_user* ítems que hayan sido vistos por cada uno de esos usuarios similares.

Durante esta selección, también se aplica dos filtro adicionales:

- **Items con puntuación mínima**: Es la puntuación mínima que debe cumplir cada ítem, la cual es establecida por el hiperparámetro *min_rating_by_user*. Por defecto, esta puntuación mínima se encuentra en 3.5 puntos, asegurando que solo aquellos ítems que han recibido una valoración aceptable sean considerados en la recomendación, lo que ayuda a mejorar la calidad de las sugerencias para el usuario actual.
- **Item no vistos**: Además se filtran los ítems no vistos por el usuario actual, definido por el hiperparámetro *not_seen*.

Cabe aclarar que la selección de ítems se realiza al azar y esta definida por el hiperparámetro *random_selection_items_by_user*, cuyo valor predeterminado es 0.5. Es decir, el 50 % de las veces se selecciona un ítem al iterar la lista de ítems de cada usuario que cumple con la restricción anteriormente definidas.

A este punto resta aplicar algún tipo de ordenamiento a la lista de ítems candidatos, a la cual finalmente se le aplicara un segundo ordenamiento mediante el modelo de lenguaje (Paso 5.a de la figura 4 en adelante).

Para ordenar los candidatos se probaron distintos enfoques.

pred_user_rating: Se ordena según las calificaciones predichas por el modelo de filtros colaborativos(*DeepFM* [6, 12]), que indica la calificación que el modelo estima que el usuario otorgaría al ítem. Esta es la medida mas personalizada pero sola no es muy efectiva.

user_item_sim: Representa la similitud entre el usuario que solicita la recomendación y el usuario que calificó el ítem seleccionado. Esto puede causar que ítems con el mismo *score* se repitan, especialmente en el caso de *fun users* que suelen calificar muchas películas de un mismo género, lo que puede hacer que sus ítems aparezcan primero. Es importante tener en cuenta que los ítems pueden repetirse entre diferentes usuarios, lo cual es normal. Por ello, se debe elegir una estrategia para aplicar una media o mediana al *score* en casos de repetición. Esto influye en cierta medida en la personalización del orden, por lo que es preferible adoptar las siguientes estrategias.

user_sim_weighted_rating_score: Se asigna un *score* a cada ítem según la similitud entre el usuario que solicita la recomendación y el que lo calificó, ponderado por la calificación media de los usuarios más cercanos. Esta estrategia soluciona el problema de tener bloques de ítems por usuario cercano, pero pierde personalización al utilizar la calificación media en lugar de la que otorgaría el usuario que realizó la consulta. Esto se aborda en la siguiente estrategia.

$$UserSimWeightedByRating = UserSim * \frac{ItemUserPredictedRating}{MaxItemsUserPredictedRating} \quad (4.1)$$

user_sim_weighted_pred_rating_score: Es similar al *score* anterior, pero más personalizado, ya que se ajusta según la calificación que se estima que el usuario otorgaría al ítem. La métrica anterior utiliza el promedio o mediana del *rating* general del ítem para usuarios cercanos, proporcionando cierto grado de personalización, pero aquí se pondra por la calificación específica predicha para el usuario que solicitó la recomendación, lo que la hace más personalizada.

$$UserSimWeightedByPredRating = UserSim * \frac{ItemUserPredRating}{MaxItemsUserPredRating} \quad (4.2)$$

Para continuar con nuestro flujo, solo queda aplicar alguno de los métodos de *scoring* a los ítems candidatos definido mas arriba. A diferencia del flujo para usuarios sin interacciones, donde la similitud de los ítems con la consulta del usuario define el orden de los ítems candidatos, aquí se utiliza un *score* basado en el aprendizaje de un modelo de filtrado colaborativo. Esta es la sutil diferencia. Posteriormente, el modelo de lenguaje ajustará el ordenamiento en un segundo filtro. Así, obtenemos un *score* personalizado que considera las correlaciones en el comportamiento del usuario con otros usuarios, así como la relación de su historia y perfil con los ítems candidatos.

5. FLUJO DE ENTRENAMIENTO DE *REC CHATBOT API*

En la sección anterior se presentó el flujo de inferencia del modelo de recomendación *Rec Chatbot API*. En ese flujo, contábamos con *embeddings* disponibles en una base de datos lista para utilizar. También disponíamos de modelos pre-entrenados. Sin embargo, ¿cómo se generan estas herramientas a partir de los datos que ingresan *Rec Chatbot API*? En este apartado, explicaremos el proceso de entrenamiento, que se encargará de generar y mantener actualizados estos modelos y *embeddings*, basándose en las actualizaciones de información en la base de datos de *Rec Chatbot API*, incluyendo altas y bajas de perfiles de usuarios y sus calificaciones. Para comenzar, se desarrollaron dos procesos o flujos que permiten esto:

- El primer flujo realiza el entrenamiento de un modelo de recomendación basado en filtros colaborativos, que puede ser sustituido por cualquier modelo que prediga calificaciones entre 0 y 5 y que ajuste vectores de *embedding* para aprender características entre usuarios e ítems. En este trabajo, se eligió un modelo con arquitectura *DeepFM* [6, 12], aunque también se podrían emplear otros como *GMF*, *Biased-GMF* o *NN-FM*. Para más detalles, consulte [8], donde se explican estos modelos en detalle y se comparan sus resultados.
- El segundo flujo se encarga de actualizar los *embeddings* necesarios para la búsqueda de ítems similares o mas cercanos a la consulta del usuario (Técnica *RAG(Retrieval Augmented Generation* o Generación Aumentada por Recuperación) [7]).

5.1. Flujo de Entrenamiento

Para gestionar de manera efectiva y ejecutar este flujo de trabajo, se eligió la herramienta conocida como *Apache Airflow*. Esta plataforma es altamente eficiente y permite programar, ejecutar y monitorizar tareas complejas de forma organizada. En *Apache Airflow*, los procesos se definen utilizando estructuras denominadas grafos dirigidos acíclicos (*DAG*), donde cada uno de los nodos dentro del grafo representa una tarea específica a llevar a cabo. Cabe señalar que estas tareas están diseñadas para ser reutilizadas según se necesite, lo que facilita el mantenimiento de los flujos de trabajo.

En el marco de este proyecto, la utilización de *Apache Airflow* se torna fundamental, ya que garantiza la automatización y orquestación de múltiples procesos importantes que están relacionados directamente con el re entrenamiento de modelos, así como con la gestión de datos necesarios para asegurar un funcionamiento efectivo de *Rec Chatbot API*. Esta herramienta no solo permite llevar a cabo los procesos de manera eficiente, sino que también ofrece una forma clara de visualizar el flujo de trabajo en su totalidad. A continuación, en la figura 5.1, se puede apreciar un diagrama de flujo ilustrativo que describe en detalle cada uno de los pasos que componen el proceso de entrenamiento (*DAG*).

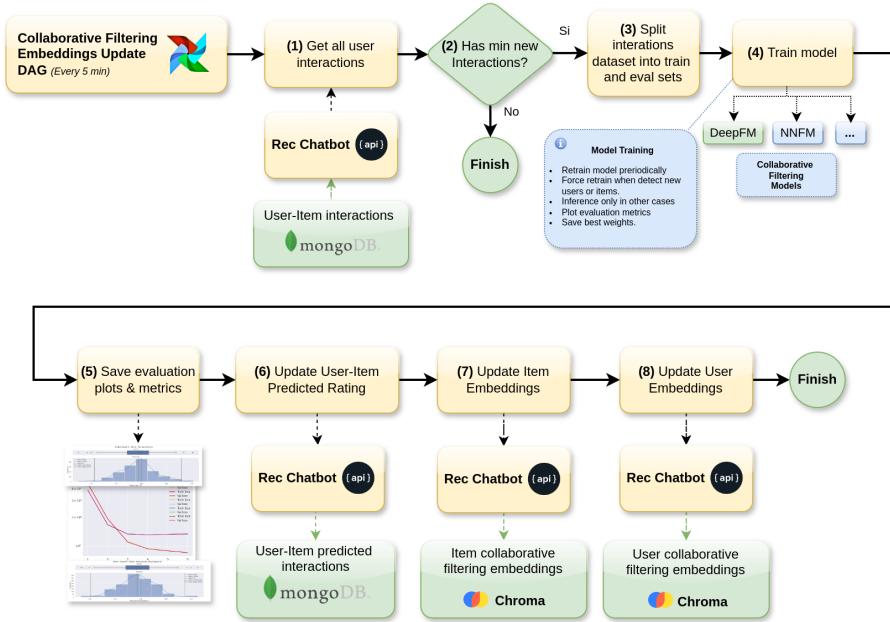


Fig. 5.1: Flujo de entrenamiento: Es un proceso que correr en segundo plano, encargado de entrenar el modelo de filtros colaborativos y actualizar los *embeddings* de usuarios e ítems.

En el paso 1 del flujo de la figura 5.1, se solicita a *Rec Chatbot API* las interacciones de los usuarios con los ítems disponibles, representadas como tuplas (*user_id*, *item_id*, *rating*). El proceso *DAG* registra el número de interacciones procesadas en ejecuciones anteriores, permitiendo determinar cuántas nuevas interacciones han ocurrido desde la última ejecución. Si la diferencia supera 50 (un valor configurable), se activa el entrenamiento; de lo contrario, se finaliza el *DAG* (Paso 2). Esta estrategia es útil para gestionar un gran número de usuarios e interacciones en aplicaciones escalables. Cuando el número de usuarios varía poco, se podría evitar el re-entrenamiento completo del modelo, ajustándose solo a las nuevas interacciones. Sin embargo, esto debe manejarse con precaución para evitar el sobre ajuste a las nuevas interacciones y descuidar las anteriores. Dado que en aplicaciones de gran escala es común un aumento constante de perfiles de usuario, esta estrategia puede no ser óptima, siendo más eficiente re-entrenar el modelo completo solo cuando sea necesario. Lo mismo ocurre con los ítems (películas), ya que es habitual que se añadan nuevos ítems y se eliminen otros en plataformas de contenido, lo que requiere re-entrenamiento.

Al llegar al paso 3, el proceso identifica interacciones nuevas y potencialmente nuevos perfiles de usuario. Dado que la *API Rec Chatbot* es escalable y puede tener millones de usuarios, no es óptimo re-entrenar el modelo de forma constante ya que se deben economizar recursos. Las instancias de *Apache Airflow* suelen ser ejecutadas en contenedores de *Docker* en una nube, como *Google Cloud Services*, lo que genera costos dependiendo del hardware seleccionado, uso de *CPU*, *GPU*, almacenamiento, etc. Por ello, se debe re-entrenar el modelo lo menos posible, solo lo justo y necesario. Para lograrlo, se implementó una abstracción llamada *ModelLoader* en *Python*, que permite cargar modelos desde un archivo de pesos y prepararlos para la inferencia. *ModelLoader* permite establecer los criterios bajo los cuales se debe re-entrenar el modelo:

- El modelo se re-entrena periódicamente, lo que significa que hay un período durante el cual solo se utiliza para inferencia.
- Si las nuevas interacciones son de usuarios e ítems existentes, no se requiere re-entrenamiento, y se espera hasta el siguiente ciclo de re-entrenamiento. Sin embargo, si hay interacciones de nuevos usuarios o ítems, es necesario forzar el re-entrenamiento del modelo, ya que de lo contrario no se podrían realizar inferencias sobre los mismos, dado que el modelo necesita generar *embeddings* para estos usuarios o ítems necesarios para la inferencia.

El *ModelLoader* determina cuándo re-entrenar el modelo. Si decide no re-entrenar, carga el modelo con los pesos del último entrenamiento; de lo contrario, se re-entrena, guarda los mejores pesos y luego carga el modelo para su inferencia. Durante el entrenamiento, *ModelLoader* genera gráficas para monitorear el proceso y verificar el ajuste del modelo, permitiendo ajustes en los hiperparámetros si es necesario. Cada ejecución produce un nuevo conjunto de gráficas para rastrear el historial de entrenamiento (Paso 5).

En este punto del proceso *DAG*, contamos con un modelo de filtro colaborativo entrenado y listo para la inferencia. En el paso 6, se realiza la inferencia de las calificaciones de ítems para los cuales los usuarios aún no han hecho una valoración. Luego, el *DAG* envía una solicitud HTTP a la *Rec Chatbot API* para guardar estas predicciones, que se utilizarán en el proceso de inferencia del modelo de recomendación de *Rec Chatbot API* para ordenar los ítems recomendados a cada usuario.

Para llevar a cabo los pasos 7 y 8, es necesario extraer del modelo de filtros colaborativos dos listas de *embeddings*: una correspondiente a los usuarios y otra a los ítems. Se conoce la asignación de cada *embedding* a su usuario e ítem asociado. De este modo, se efectúan dos llamadas HTTP a la API de Rec Chatbot para persistir las nuevas versiones de los *embeddings* de usuarios e ítems generadas durante el entrenamiento del modelo. Posteriormente, estos *embeddings* son utilizados por el modelo de recomendación de la Rec Chatbot API para buscar ítems calificados por usuarios similares y también para encontrar ítems que sean similares a los resultados de una consulta del usuario.

5.1.1. Flujo de actualización de búsquedas (Técnica *RAG(Retrieval Augmented Generation* o Generación Aumentada por Recuperación))

Este flujo se ejecuta manualmente y genera *embeddings* para cada ítem, utilizando el título, la sinopsis y los géneros de cada película. Para ello, emplea un modelo conocido como *Sentence Transformers*, que permite crear *embeddings* a partir de oraciones o párrafos. Estos *embeddings* son utilizados posteriormente para identificar los ítems más cercanos a una consulta de usuario.

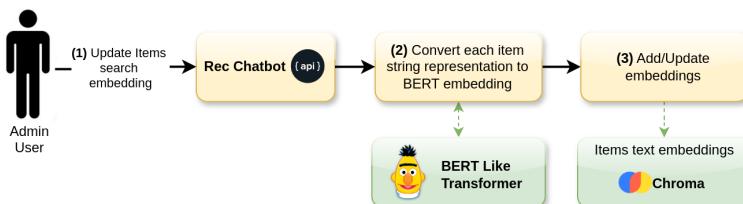


Fig. 5.2: Flujo de generación y actualización *embeddings* utilizados para la búsqueda de ítems cercanos a una consulta del usuario (Técnica *RAG(Retrieval Augmented Generation* o Generación Aumentada por Recuperación) [7]).

6. MODELO INTERNO BASADO EN FILTROS COLABORATIVOS

En un trabajo previo de especialización [8] de la maestría, se exploraron diversos enfoques para implementar sistemas de recomendación, incluyendo *KNN*. Sin embargo los modelos basados en *KNN* presentan varias limitaciones significativas, siendo el problema de escala uno de los más importantes. El tamaño de los datos a procesar depende casi linealmente de los recursos de memoria, *CPU* y/o *GPU* disponibles. Por lo tanto, al necesitar procesar grandes volúmenes de datos para hacer predicciones, se suelen elegir modelos que aplican algún tipo de reducción de dimensionalidad para construir su representación interna, la cual se utiliza posteriormente para realizar predicciones. Esta representación interna se denomina a menudo "modelo", ya que lo que define el modelo no es el algoritmo en sí, sino el estado interno alcanzado tras el entrenamiento.

6.1. Codificación *One-Hot* vs *Embeddings*

Particularmente en el ámbito de recomendaciones, se cuenta con variables categóricas de alta dimensionalidad. Para este trabajo, tenemos dos variables con esta característica: los identificadores secuenciales de usuarios e ítems. Cuando trabajamos con modelos de *Machine Learning*, particularmente con redes neuronales, es necesario convertir las variables categóricas en una representación numérica. El enfoque más simple o *naive* consiste en realizar una codificación *one-hot* de la variable categórica, la cual consta de codificar cada posible valor de la variable como un vector que contiene tantas posiciones como valores tenga la variable. De esta forma, cada vector tiene un valor 1 en la posición que concuerde con el valor representado y un valor 0 (cero) en las demás posiciones. Por ejemplo, suponemos que tenemos la siguiente variable:

- Variable Categórica: Estado del Tiempo.
- Posibles valores: Nublado, Despegado y Lluvioso.

Si codificamos sus valores usando una codificación *one-hot*, obtendremos los siguientes vectores:

- *Nublado* = [1, 0, 0]
- *Despegado* = [0, 1, 0]
- *Lluvioso* = [0, 0, 1]

Entonces, el valor *Nublado* se convierte en 3 entradas para una red neuronal a las cuales se le pasa los números 1, 0 y 0 respectivamente. Ahora pensemos en la cantidad de usuarios que tiene *Google* o *Amazon*. ¿Qué tamaño tendría el vector que representa a un solo usuario? ¿Por qué usar un vector 99% vacío para representar un valor? ¿No hay una forma más compacta de realizar esta codificación?

La respuesta corta es sí, en estos casos se utilizan *Embeddings*. ¿Pero qué son los *Embeddings* y en qué se diferencian de la codificación *one-hot*?

Un *Embedding* no es más que una forma de codificar valores de una variable categórica usando vectores de menor tamaño. Es decir, si tenemos una variable categórica que tiene

10.000 posible valores, dependiendo del caso, podríamos elegir un tamaño de 100 posiciones. Este tamaño debe ser elegido de forma tal que no se produzca pérdida de información. Por esta cuestión, el tamaño de estos vectores se transforma en un hiperparámetro mas a ajustar al momento de entrenar los modelos que utilicen esta técnica de codificación.

Otro punto importante que diferencia ambas codificaciones, reside en la distancia entre vectores. Si tomamos dos vectores con codificación *one-hot* y los graficas en un espacio tridimensional o bidimensional, se aprecia que el ángulo entre estos siempre es el mismo, 90 grados. Supongamos el caso anterior de la variable Estado del Tiempo, si representamos en el espacio todos sus valores, podemos ver que la distancia es la misma entre cualquier par de vectores. Si ahora codificamos la misma variable usando *Embeddings* esto cambia, ya que los vectores que representan a los valores Nublado y Lluvioso tiene un ángulo menor a 90 grados. Por otro lado, ambos vectores están alejados del vector Despejado. De esta forma, un *Embedding* permite captar mas información, ya que realiza una clusterización o agrupación de los valores que son mas cercanos en términos de significado. Los días nublados y lluviosos son muy parecido entre sí y muy distintos a un dia despejado.

De esta forma los *Embeddings* tiene una doble ganancia sobre la codificación *One-Hot*: comprimen la información y además captan información útil para la clusterización o agrupación de sus valores. Algo interesante a destacar, es que los modelos que entrena *Embeddings* captan esta información de forma automática en base a las observaciones usadas en el entrenamiento, generando estos espacios latentes llamados *Embeddings*.

6.2. Capa o módulo *Embedding*

En el ámbito del *Deep Learning* o *Machine Learning* se cuenta con la abstracción de capas (en *frameworks* como *Keras*) o módulos (en *PyTorch*), las cuales encapsulan el comportamiento esencial en un conjunto de bloques básicos utilizados para construir cualquier modelo. Los bloques que permiten que un modelo infiera o construya un *embedding* durante el entrenamiento, son los bloques *Embedding* y *EmbeddingBag* en *PyTorch* o simplemente *Embedding* en *Keras*.

Por un lado, podemos elegir el tamaño de los vectores *embedding*, el cuál, como ya adelantamos, es un hiperparámetro mas a optimizar. Por otro lado, debemos definir la cantidad de vectores de *embedding* que debe contener la capa. Ésta es siempre igual al número total de valores que puede tomar la variable categórica a codificar.

De esta forma, si deseamos crear una capa o módulo *Embedding* para la variable categórica Estado del Tiempo, deberíamos crear una capa de tamaño 3, ya que cuenta con 3 posible valores, con un tamaño de vector menos a 3, ya que de lo contrario, tendríamos la misma dimensionalidad que tenemos al usar la codificación *one-hot*, con la diferencia de que una capa *Embedding* capta la similitud entre los valores de la variable categórica a diferencia de la codificación *one-hot*.

El modo de funcionamiento de la capa es muy simple. Esta se puede pensar como una tabla de *Hash*, donde cada clave es un valor de la variable categórica. Es decir, que tendremos tantas claves como valores pueda tomar la variable categórica. Estas claves son codificados a números y los valores asociados a cada clave son vectores *embedding*. Cave aclarar que en general, estos vectores son inicializados con valores aleatorios. Finalmente, en la etapa de entrenamiento, el modelo irá ajustando los valores de cada vector *embedding*.

En la etapa de *forward pass*, se pasa como entrada un valor de la variable categórica codificado como numérico. Para nuestra variable Estado del Tiempo podríamos codificar sus valores como sigue:

- Nublado => 0
- Despegado => 1
- Lluvioso => 2

Entonces, si pasamos el valor Nublado como entrada a la capa, en realidad estamos pasando la número o clave *hash* 0. Luego de esto, la capa resuelve el vector *embedding* asociado a esa clave y lo devuelve a su salida.

Finalmente, debemos tener en cuenta que el proceso de *back-propagation* sera el encargado de ir ajustando los valores, también llamados pesos de los vectores *embedding*, de acuerdo a lo que se requiera en la salida del modelo durante el proceso de optimización.



Fig. 6.1: Esquema de una capa o módulo *Embedding*.

6.3. Máquinas de Factorización (*FM*)

Antes de introducir el modelo de máquinas de factorización profundas (*DeepFM* [6, 12]) se comenzara explicando uno de sus componentes mas importantes: las máquinas de factorización [13, 14].

Las máquina de factorización propuestas por *Steffen Rendle* en 2010 [9], son algoritmos supervisados que puede ser utilizados para tareas de clasificación, regresión y tareas de ranking como sucede en el ámbito de los sistemas de recomendación. Rápidamente se convirtieron en un método popular para hacer predicciones y recomendaciones. La máquina de factorización es una generalización de un modelo lineal y un model de factorización de matrices, mas aun, recuerdan mucho a un máquina de soporte vectorial (*SVM*) que utiliza un kernel polinomial.

A continuación, se define el modelo formalmente:

- $x \in \mathbb{R}^d$ es un vector de características donde cada una de sus componentes representa a una variable del conjunto de datos, siendo d la cantidad de variables. En nuestro caso, $x \in \mathbb{R}^2$ ya que tenemos dos variables, usuarios e ítems.
- $y \in \mathbb{R}$ es la variable *target* o resultado a predecir. Dado el dominio del conjunto de datos seleccionado, seria la calificación del usuario.

Luego, podemos definir el modelo para una máquina de factorización de grado 2 de la siguiente forma:

$$\hat{y}(x) = \mathbf{w}_0 + \sum_{i=1}^d \mathbf{w}_i x_i + \sum_{i=1}^d \sum_{j=i+1}^d \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \quad (6.1)$$

Donde:

- d es la cantidad de características o variables a utilizar. Para el caso de estudio en este trabajo $d = 2$ (usuarios e ítems).
- $\mathbf{w}_0 \in \mathbb{R}$ es el bias o intercepto del modelo.
- $\sum_{i=1}^d \mathbf{w}_i x_i$: Esta es la parte lineal del modelo. Aquí, x_i representa el valor del *feature* i -ésimo. \mathbf{w}_i es el peso asociado al *feature* i -ésimo, que determina la influencia de ese *feature* en la predicción. Multiplicamos el valor de cada *feature* x_i por su peso correspondiente \mathbf{w}_i y sumamos todas estas contribuciones para obtener la parte lineal de la predicción.
- $\sum_{i=1}^d \sum_{j=i+1}^d \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$: Esta es la parte de factorización del modelo. Donde, $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ es el producto interno (o producto escalar) entre los vectores \mathbf{v}_i y \mathbf{v}_j . Cada vector \mathbf{v}_i representa una factor latente o *Embedding* del *feature* i -ésimo. Estos vectores capturan interacciones no lineales entre características y se utilizan para modelar relaciones más complejas entre ellas. Similar a la parte lineal, multiplicamos los valores de las características x_i y x_j por el producto interno $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ y sumamos todas las contribuciones para obtener la parte de factorización de la predicción.

En resumen, la expresión completa $\hat{y}(x)$ representa la estimación de la variable dependiente \hat{y} basada en la entrada x , utilizando una combinación lineal de los pesos de las características y una suma de productos internos entre los vectores de características latentes para capturar interacciones no lineales entre las características. Este modelo es

comúnmente utilizado en el campo del aprendizaje automático para problemas de regresión y clasificación.

De esta forma los dos primeros términos corresponden al modelo de regresión lineal y el último término es una extensión del modelo de factorización matricial. Si la variable i representa un ítem y la variable j a un usuario, el tercer término es el producto escalar entre los vectores *embedding* de usuario u y ítem i . Por otro lado, vale la pena aclarar que este método también puede generalizar en órdenes superiores al grado 2, sin embargo, la estabilidad numérica podría disminuir la generalización del método.

Al aplicar un método de optimización con las máquinas de factorización, como puede ser el método del gradiente descendente, se puede llegar fácilmente a una complejidad del orden $\mathcal{O}(kd^2)$, ya que se deben calcular todas las interacciones de a pares. Para resolver este problema de *performance*, podemos reorganizar el tercer término del método. Esto reduce en gran medida el costo de cálculo, llevándolo a una complejidad de tiempo de orden lineal $\mathcal{O}(kd)$. A continuación se describen los pasos para bajar el nivel de complejidad del método:

$$\begin{aligned}
 &= \sum_{i=1}^d \sum_{j=i+1}^d \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \\
 &= \frac{1}{2} \sum_{i=1}^d \sum_{j=1}^d \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j - \frac{1}{2} \sum_{i=1}^d \langle \mathbf{v}_i, \mathbf{v}_i \rangle x_i x_i \\
 &= \frac{1}{2} \left(\sum_{i=1}^d \sum_{j=1}^d \sum_{l=1}^k \mathbf{v}_{i,l} \mathbf{v}_{j,l} x_i x_j - \sum_{i=1}^d \sum_{l=1}^k \mathbf{v}_{i,l} \mathbf{v}_{i,l} x_i x_i \right) \\
 &= \frac{1}{2} \sum_{l=1}^k \left(\left(\sum_{i=1}^d \mathbf{v}_{i,l} x_i \right) \left(\sum_{j=1}^d \mathbf{v}_{j,l} x_j \right) - \sum_{i=1}^d \mathbf{v}_{i,l}^2 x_i^2 \right) \\
 &= \frac{1}{2} \sum_{l=1}^k \left(\left(\sum_{i=1}^d \mathbf{v}_{i,l} x_i \right)^2 - \sum_{i=1}^d \mathbf{v}_{i,l}^2 x_i^2 \right)
 \end{aligned} \tag{6.2}$$

Con esta re-formulación del último término, la complejidad del método se reduce considerablemente. Además, para las variables ralas, solo se deben computar los valores distintos de cero, para que la complejidad general sea lineal. Finalmente, la expresión del método aplicando esta re-formulación queda como sigue:

$$\hat{y}(x) = \mathbf{w}_0 + \sum_{i=1}^d \mathbf{w}_i x_i + \frac{1}{2} \sum_{l=1}^k \left(\left(\sum_{i=1}^d \mathbf{v}_{i,l} x_i \right)^2 - \sum_{i=1}^d \mathbf{v}_{i,l}^2 x_i^2 \right) \tag{6.3}$$

6.4. Máquinas de factorización profundas (*DeepFM*)

Hasta aquí, a grandes rasgos, todos los modelos expuestos tratan de captar el comportamiento de las interacciones o correlación usuario-ítems, ya sean implícitas o explícitas. A pesar de este gran progreso, los métodos expuestos anteriormente (exceptuando las máquinas de factorización) parecen tener un fuerte sesgo al predecir las interacciones o correlaciones de bajo y alto orden, requiriendo en algunos casos realizar ingeniería de características para disminuir estos sesgos.

El modelo de máquinas de factorización profundas (*DeepFM*) [6, 12] o maquina de factorización basada en *Deep Learning*, mejora el aprendizaje de las interacciones o correlaciones de bajo y alto orden. Este modelo combina máquinas de factorización y *Deep Learning* en una nueva arquitectura de red neuronal, la cual captura estas correlaciones. Por otro lado, es una evolución del modelo *Wide and Deep* [5] de *Google*, el cual es un ensamble de dos modelos: uno lineal, que captura las interacciones o correlaciones de alto orden y una red neuronal perceptron multi-capa o *Multi-Layer Perceptron (MLP)*, la cual captura correlación de mas bajo orden (aquellas mas complejas).

A continuación de puede visualizar un diagrama de bloques de alto nivel del modelo:

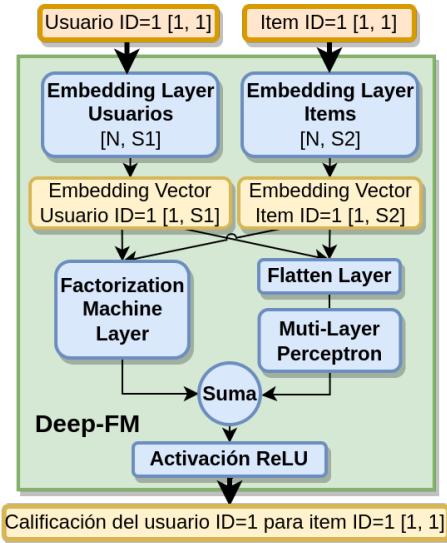


Fig. 6.2: Esquema de un modelo *Deep Factorization Machine (DeepFM)* [6, 12] o maquina de factorización basada en *Deep Learning*.

Donde se puede apreciar que las entradas del modelo son las variables categóricas correspondiente a usuarios e ítems, como en los modelos previamente visto. Dado un identificador de usuario e ítem, se resuelve sus correspondientes vectores *Embedding*, los cuales se convierten en entradas para los siguientes dos bloques. Uno de los bloques intermedios, no es mas que una red neuronal multi capa con capas densas o *fully connected*. Por el otro lado, ambos vectores se toman como entrada a la máquina de factorización. Las salidas de ambos bloques intermedios, de valores escalares, se suman y se pasan por una activación *ReLU*. Para el caso de estudio de este trabajo, las salidas o calificaciones toman valores mayores a cero, por esta cuestión es mas adecuado usar una activación *ReLU* frente a una lineal.

7. PRUEBAS DE EVALUACIÓN

En este apartado se describirá detalladamente el tipo de pruebas realizadas y cómo se llevaron a cabo. Para la evaluación, se propone un escenario con una instalación real de la *API Rec Chatbot*, la cual ya se explica en detalle en el apartado 3. De esta forma, contamos con dicha instalación y su respectivo proceso de entrenamiento (*DAG*), el cual se ejecuta cada *5 min* si se encuentran mas de *50* nuevas iteraciones(calificaciones de los usuarios). El objetivo de la prueba es simular un ambiente real de uso de la *API Rec Chatbot* y evaluar la evolución de las recomendaciones para cada usuario de prueba.

7.1. Preparación del ambiente de evaluación

Se dispone de dos conjuntos de datos: uno para entrenamiento y otro para prueba. Tras realizar el flujo de preprocesamiento descrito en la sección 2.3, se excluyen los usuarios que tienen menos de *20* interacciones. Después de este proceso, obtenemos un conjunto de datos con la siguiente distribución de interacciones máximas por usuario (figura 7.1):



Fig. 7.1: Este gráfico es un histograma que muestra la distribución de interacciones máximas de usuarios. Cada rango representa un rango de interacciones, excluyendo el primer rango de *0* a *19* interacciones. El eje *X* (horizontal) muestra diferentes rangos de interacciones de usuarios, desde *0-19* hasta *200+*. El eje *Y* (vertical) muestra el conteo de usuarios, llegando aproximadamente hasta *450* como máximo por rango.

Observaciones

- El pico más alto está en el rango de *20-29* interacciones, con aproximadamente *450* usuarios.
- Hay una tendencia general descendente desde ese pico hasta el rango *90-99*.
- Existe un segundo pico menor en el rango *100-199*, seguido de una disminución en *200+*.
- El rango *0-19* fue filtrado del conjunto de datos, ya que estas interacciones no son buenas para entrenar al modelo.
- La mayoría de los usuarios tienen entre *20-29* interacciones.

- Es relativamente poco común tener usuarios con más de 100 interacciones.
- Hay un grupo interesante de usuarios muy activos en el rango 100-199.

Para construir el conjunto de evaluación se seleccionaron 15 usuarios de cada uno de los siguientes grupos 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-89, 90-99 y 100-199. Excluyendo los grupos 0-19 y 200+ para tratar de representar el comportamiento del usuario promedio, excluyendo comportamientos atípicos o aquellos que no van a permitir realizar una prueba completa del modelo (0-19). De esta forma se construye un conjunto de evaluación que represente el comportamiento del usuario promedio, quedando dos conjuntos de entrenamiento y evaluación con la siguiente cantidad de interacciones:

- **Entrenamiento:** 185.096 interacciones (95 % el conjunto de datos).
- **Evaluación:** 9.177 (0.04 % el conjunto de datos). Este número de interacciones resulta suficiente para realizar la prueba de evaluación y permite utilizar el mayor número posible de interacciones para el entrenamiento del modelo. Además debe tenerse en cuenta el tiempo de ejecución de cada inferencia del modelo. Este ronda entre los 8 y 20 segundos. Debido a esto último, tampoco sería posible realizar un particionamiento tradicional 70-30 ya que tendríamos un número muy grande de interacciones a evaluar.

Por otro lado, se agregaron interacciones de usuarios falsos, conocidos como usuarios *fun*. Estos usuarios son entusiastas de ciertos géneros, sagas de películas, etc. Esto usuarios se crean de manera artificial para mejorar las recomendaciones de los modelos de recomendación de filtrado colaborativo, especialmente en casos donde hay grupos de usuarios aislados que no se relacionan entre sí. Para llevar a cabo este proceso, se buscaron ítems a partir de una frase y luego se crea un nuevo usuario al que se asocia una interacción por cada ítem encontrado. A continuación, se muestra el mapeo de frases y la cantidad de ítems obtenidos en cada búsqueda. Cada frase corresponde a un usuario *fun*.

- "pixar animated movie for children" (Máximo de 200 ítems).
- "science fiction, action" (Máximo de 100 ítems).
- "war weapons" (Máximo de 70 ítems).
- "iron man, x-men, spider man, bat man, flash, avengers, ant-man, hulk, guardians of the galaxy, marvel, green lantern, superman, watchmen, thor, deadpool, wonder woman, strange, justice League, captain america, logans, kick ass, John Wick" (Máximo de 150 ítems).
- "sci-fi, action, future" (Máximo de 130 ítems).
- "Mission Impossible, spies, bourne identity, 007" (Máximo de 70 ítems).
- "comedy movies" (Máximo de 200 ítems).
- "horror" (Máximo de 100 ítems).
- "thriller, suspense" (Máximo de 150 ítems).
- "love, romance" (Máximo de 150 ítems).
- "time travel" (Máximo de 100 ítems).
- "dead, zombies, post apocalyptic" (Máximo de 100 ítems).
- "software" (Máximo de 100 ítems).
- "hackers" (Máximo de 100 ítems).

De esta forma, se agregan 14 nuevos usuarios artificiales o usuarios *fun*, junto con sus interacciones, al conjunto de datos de entrenamiento.

Para realizar el proceso de evaluación, en primer lugar se deben crear los perfiles de los usuarios encontrados en el conjunto de evaluación. Dado que no se cuenta con información de los mismos. Para realizar esto se generan datos falsos, de forma aleatoria, para el nombre y *email* del usuario. Por otro lado se definen los campos *preferred_from* y *genres* que también son parte del perfil de usuario:

- *preferred_from*: Este filtro establece la fecha mínima de lanzamiento de las películas recomendadas. Basándose en las iteraciones de evaluación de cada usuario, se selecciona la fecha de lanzamiento más temprana entre todas las películas que el usuario calificó.
- *géneros*: Es una colección de los géneros preferidos por el usuario, al menos inicialmente, similar al campo anterior. Se utiliza un conjunto de todos los géneros.ros de las películas calificadas pro el usuario en el conjunto de evaluación.

una vez creados todo los perfiles se dan de alta en *API Rec Chatbot* mediante el *endpoint POST /api/v1/profiles*.

Cabe aclarar que estos usuarios no están en el conjunto de entrenamiento; son nuevos para la *API Rec Chatbot*. Además, los usuarios e interacciones presentes en la base de datos de *API Rec Chatbot* pertenecen en su totalidad al conjunto de entrenamiento. A este momento tenemos la base de datos con el conjunto completo de entrenamiento y cargados los perfiles de usuarios del conjunto de evaluación.

7.2. Proceso de evaluación

El proceso de evaluación simula las interacciones de los usuarios con el *API Rec Chatbot*. Se selecciona aleatoriamente y sin reposición cada usuario del conjunto de evaluación, y se itera sobre sus interacciones en orden cronológico, creando consultas a el *API Rec Chatbot* en grupos de 5 interacciones una a la vez. Las consultas o *prompts* tienen el siguiente patrón:

I want to see: title1..title5

Aquí se aprecia que se concatena la cadena de caracteres I want to see: con los títulos de las 5 películas correspondientes a las interacciones del usuario a evaluar. Luego, *API Rec Chatbot* buscará y recomendará al menos 5 películas, incluyendo la opción de calificar cada una con una puntuación de 1 a 5. Las puntuaciones reales del conjunto de evaluación se utilizarán para calificar las recomendaciones. Este proceso se repetirá con las siguientes 5 películas, en el orden cronológico de las interacciones, hasta que se agoten las interacciones del usuario o se alcance un límite de re-intentos en caso de errores con la *API Rec Chatbot*.

Mientras se realiza la prueba de evaluación durante el entrenamiento de la solución, se identifican nuevas interacciones y se decide si es necesario re-entrenar el modelo. El modelo se re-entrena cuando se detectan al menos 25 nuevas interacciones, garantizando así que no haya un desfase significativo. En aplicaciones reales, re-entrenar el modelo cada vez que un usuario califica no es viable debido a los altos requerimientos computacionales, lo que resulta insostenible en términos de *hardware* y costos. Por ello, se opta por un enfoque de entrenamiento en paralelo. El proceso de evaluación continuará hasta alcanzar un mínimo de 50 interacciones por usuario.

El *API Rec Chatbot* consiste en un ensamble de modelos que utiliza modelos de lenguaje para la inferencia. Por ello, se decidió evaluar el sistema dos veces, utilizando dos

versiones diferentes del modelo *Llama*: *Llama2* y *Llama3*. Esta selección se realizó tras una prueba temprana donde se compararon distintas implementaciones de modelos de lenguaje, eligiendo estos dos por su consistencia en las respuestas. No todos los modelos de lenguaje ofrecen respuestas consistentes, lo que puede provocar errores en el *parsing* y hacer que el *API Rec Chatbot* no retorne recomendaciones el 100 % de las veces.

7.3. Métricas

El proceso de evaluación descrito anteriormente registrará los resultados de cada interacción de los usuarios del conjunto de evaluación con *API Rec Chatbot*, generando un archivo que contendrá los datos necesarios para calcular métricas que permitan evaluar el rendimiento del sistema de recomendación. A continuación se definen todas las métricas utilizadas para facilitar una comprensión inicial. Esto permitirá entender mejor los resultados y las conclusiones de este trabajo de tesis.

7.3.1. NDCG (Normalized Discounted Cumulative Gain)

El *Normalized Discounted Cumulative Gain* (*NDCG*) es una métrica de evaluación común en el aprendizaje automático, especialmente en tareas de *ranking* o sistemas de recomendación. Esta métrica mide la calidad de una lista ordenada de resultados, teniendo en cuenta no solo si los elementos relevantes están presentes en la lista, sino también su posición. Se basa en dos principios fundamentales:

- Los ítems altamente relevantes son más útiles cuando aparecen primeros en la lista de recomendaciones.
- Los ítems altamente relevantes son más útiles que los marginalmente relevantes.

Funcionamiento

A continuación definimos *NDCG* como:

$$\begin{aligned} DCG &= \sum_{i=1}^n \frac{rating_i}{\log_2(i+1)} \\ IDCG &= \sum_{i=1}^n \frac{rating_i^{ideal}}{\log_2(i+1)} \\ NDCG &= \frac{DCG}{IDCG} \end{aligned} \tag{7.1}$$

Donde:

- n es la cantidad de ítems en la lista de recomendaciones.
- $rating_i$ es la calificación del ítem en la posición i de dicha lista.
- $rating_i^{ideal}$ también representa una calificación, pero se basa en una lista ordenada de mayor a menor calificación, lo que significa que el ítem en la posición i podría ser diferente al ítem de la lista original.

Primero, se calcula el *DCG* (*Discounted Cumulative Gain*):

- Para cada ítem i de la lista de recomendada, se divide su *rating* o calificación por el logaritmo de su posición en la lista de recomendaciones ($\log_2(i + 1)$). El logaritmo en base 2 de la posición actúa como un factor de descuento.
- Los ítems en las primeras posiciones de la lista están menos penalizadas por el factor de descuento. Cuanto mayor es el valor de esta división, más relevante es el ítem para la métrica.
- Finalmente se suman todos los valores calculados para cada ítem en la lista, siendo este valor final una medida del *Discounted Cumulative Gain*.

Luego, es necesario calcular el *IDCG* (*Discounted Cumulative Gain ideal*):

- Se ordenan los ítems de la lista por *rating* o calificación de mayor a menor.
- Se calcula el *DCG* (*Discounted Cumulative Gain*) con este orden ideal.

Como paso final, se normaliza el *DCG* dividiendo por el *IDCG*. Esto escala *DCG* a un valor entre 0 y 1, donde:

- 1 indica que la posición de los ítems en la lista(*ranking*) es igual al ideal. Es decir, los ítems están ordenados descendente-mente por la calificación que el usuario le ha dado.
- 0 indica el peor *ranking* posible, es decir sería el orden inverso al *ranking* ideal.

Ejemplo práctico

Supongamos que contamos con dos sistemas de recomendación de películas, donde ambos miden la relevancia con una puntuación entre 0 a 3 puntos. Luego obtenemos las recomendaciones de ambos sistemas como sigue:

- Sistema *A* recomienda:
 - Película muy relevante (3 puntos)
 - Película poco relevante (1 punto)
 - Película relevante (2 puntos)
- Sistema *B* recomienda:
 - Película poco relevante (1 punto)
 - Película relevante (2 puntos)
 - Película muy relevante (3 puntos)

Aunque ambos sistemas recomiendan las mismas películas, el Sistema *A* tendrá un *NDCG* mayor, porque ubicó la película más relevante o mayor puntuada al principio. Esto refleja una mejor experiencia de usuario, ya que encontrará contenido más relevante.

7.3.2. Exhaustividad (*Recall*)

Mide la capacidad del sistema para encontrar todos los ítems relevantes para el usuario. Se enfoca en evaluar cuántos de los ítems que realmente le interesan al usuario fueron recomendados. No considera el orden de las recomendaciones ni la precisión con ítems no relevantes.

Funcionamiento

A continuación definimos (*Recall*) como:

$$\text{True Positive} = \text{Ítems Recomendados} \cap \text{Ítems Relevantes} \quad (7.2)$$

$$\text{True Positive} + \text{False Negative} = \text{Ítems Relevantes} \quad (7.3)$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \quad (7.4)$$

Entonces, los pasos para calcular el (*Recall*) son los siguientes:

- Se identifica el conjunto de ítems relevantes para el usuario. En el caso de la prueba de evaluación, son todos los ítems utilizados para construir el *prompt* de consulta.
- Se cuenta cuántos de estos ítems relevantes aparecen en la lista de ítems recomendados resultado de la consulta 7.2.
- Se divide por el número total de ítems relevantes 7.4.
- El resultado es un valor entre 0 y 1 donde:
 - El valor 1 indica que todos los ítems recomendados eran relevantes para el usuario, lo que en la prueba de evaluación significa que todos los ítems coincidían con los utilizados para generar el *prompt* de la consulta.
 - El valor 0 significa que no se recomendó ningún ítem relevante. Lo que significa que ningún de los ítems recomendados se encontraban en el *prompt* de la consulta.

Ejemplo práctico

En una aplicación de venta de libros *online*, un usuario está interesado en 5 libros de ciencia ficción. El sistema le recomienda 5 libros, de los cuales 3 son de los que le interesan.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} = \frac{3}{5} = 0,6 = 60\% \quad (7.5)$$

Esto indica que el sistema encontró más de la mitad de los libros que le interesaban al usuario, pero se perdió dos títulos relevantes.

7.3.3. Mean Reciprocal Rank (MRR)

El *MRR* se usa para evaluar la eficacia de un sistema de recomendación o motor de búsqueda en términos de cuán cerca del inicio en los resultados se encuentran los ítems relevantes para los usuarios. Un *MRR* alto sugiere que el sistema está bien optimizado para mostrar los resultados más relevantes en las primeras posiciones, mientras que un valor más bajo indica que los usuarios deben revisar más resultados para encontrar lo que buscan.

Funcionamiento

La métrica se define como sigue:

$$MRR = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{1}{rank_u} \quad (7.6)$$

Donde:

- $|U|$ es el número total de usuarios registrados en el sistema de recomendación.
- $rank_u$ es la posición del primer ítem relevante para el usuario u dentro de una lista de ítems recomendada por el sistema de recomendación.

Entonces, para cada usuario:

- Se encuentra la posición del primer ítem relevante ($rank_u$) en la lista recomendada.
- Se calcula el recíproco de esta posición ($1/rank_u$).

Luego, se promedian todos los valores calculados para cada usuario de la siguiente manera:

- Se suman todos los recíprocos.
- Se divide el total por el número de usuarios ($|U|$).

El resultado será un valor entre 0 y 1, donde:

- 1 significa que el primer ítem fue relevante para todos los usuarios.
- Valores cercanos a 0 indican que los ítems relevantes aparecen en las últimas posiciones en la lista de ítems recomendados a cada usuario.

Ejemplo práctico

En un buscador de restaurantes:

- El usuario 1 encuentra el primer restaurante relevante en la 2da posición (1/2).
- El usuario 2 encuentra el primer restaurante relevante en la 1ra posición (1/1).
- El usuario 3 encuentra el primer restaurante relevante en la 4ta posición (1/4)

$$MRR = \frac{1/2 + 1/1 + 1/4}{3} = 0,58 \quad (7.7)$$

Esto indica que, en promedio, los usuarios encuentran un restaurante relevante en la segunda posición, ya que $0,58 \approx 1/2$, donde 2 es la posición.

7.3.4. Mean Average Precision (MAP)

Es una métrica comúnmente utilizada para evaluar la precisión de un sistema de recomendación, un motor de búsqueda, o un modelo de clasificación. En términos simples, *MAP* mide la precisión de un modelo al identificar elementos relevantes entre los resultados y lo hace promediando la precisión en diferentes niveles de relevancia a lo largo de las búsquedas de múltiples usuarios o consultas. Es más rigurosa que el *Recall* ya que considera la posición en que aparecen los ítems relevantes.

Cálculo del *MAP*

1. *Average Precision (AP)* para una consulta individual: Para cada consulta o búsqueda, se calcula el *Average Precision (AP)*, que es el promedio de la precisión en cada posición donde aparece un ítem relevante. La precisión en una posición k es la fracción de ítems relevantes entre los primeros k resultados.

La fórmula para el *AP* de una consulta es la siguiente:

$$\text{Precisión}@k = \frac{\text{Cantidad de ítem relevantes encontrados hasta } k}{k}$$

$$AP = \frac{1}{\text{Cantidad total de ítems relevantes}} \sum_{k=1}^n \text{Precisión}@k \times \text{Relevancia de ítem } k$$

Donde:

- n : Es el número total de ítems en la lista recomendada al usuario.
 - Precisión@ k : Se define como el número de ítems relevantes encontrados en los primeros k ítems, dividido por el total de ítems analizados hasta el momento (k). Por ejemplo, si se analizan los primeros $k = 3$ ítems y se encuentra un ítem relevante entre estos, la precisión@ k es igual a $1/3$.
 - Relevancia en k : Toma el valor 1 si el ítem en la posición k es relevante y 0 si no lo es. Esto permite filtrar los ítems irrelevantes, enfocándose únicamente en las precisiones de los ítems relevantes.
 - Finalmente, se suman las precisiones en las posiciones donde el ítem es relevante, y luego se divide por el número total de ítems relevantes para esa lista recomendada.
2. *MAP* es el promedio de los *AP's*: Tras calcular el *AP* de cada lista recomendada, se promedian estos valores para obtener el *MAP*. Esto refleja el promedio de los *AP's* para todos los usuarios del sistema de recomendación, donde se consulta una recomendación para cada uno de ellos.

La fórmula es la siguiente:

$$MAP = \frac{1}{Q} \sum_{q=1}^Q AP_q$$

Donde:

- Q es el número total de consultas de recomendaciones o usuarios. Si tenemos n usuarios se debe consultar una recomendación de ítems por cada uno.
- AP_q es el *Average Precision* de la consulta q .

Ejemplo de Cálculo de *MAP*

Imaginemos que tenemos 3 usuarios que buscan restaurantes, y los resultados relevantes aparecen en las siguientes posiciones:

- El usuario 1 encuentra ítems relevantes en las posiciones 1 y 3.
- El usuario 2 encuentra ítems relevantes en las posiciones 2 y 4.
- El usuario 3 encuentra un ítem relevante en la posición 1.

Cálculo de *AP* para cada usuario:

- Usuario 1:

$$\begin{aligned}\text{Precisión en posición 1} &= \frac{1}{1} = 1 \\ \text{Precisión en posición 3} &= \frac{2}{3} \approx 0,67 \\ AP_1 &= \frac{1 + 0,67}{2} = 0,835\end{aligned}$$

- Usuario 2:

$$\begin{aligned}\text{Precisión en posición 2} &= \frac{1}{2} = 0,5 \\ \text{Precisión en posición 4} &= \frac{2}{4} = 0,5 \\ AP_2 &= \frac{0,5 + 0,5}{2} = 0,5\end{aligned}$$

- Usuario 3:

$$\begin{aligned}\text{Precisión en posición 1} &= \frac{1}{1} = 1 \\ AP_3 &= 1\end{aligned}$$

Cálculo del *MAP*:

$$MAP = \frac{AP_1 + AP_2 + AP_3}{3} = \frac{0,835 + 0,5 + 1}{3} \approx 0,778$$

Interpretación del *MAP*:

- Un *MAP* cercano a 1 significa que los ítems relevantes suelen aparecer en las primeras posiciones para todos los usuarios, indicando alta precisión del sistema.
- Un *MAP* cercano a 0 sugiere que los ítems relevantes están, en promedio, en posiciones bajas, indicando que el sistema podría mejorar en términos de precisión y relevancia en los primeros resultados.

7.3.5. Catalog Coverage

La métrica *Catalog Coverage* (Cobertura del Catálogo) mide qué tan amplio es el rango de ítems o productos que un sistema de recomendación muestra a los usuarios. En pocas palabras, representa la diversidad del catálogo que es recomendado a través de todas las interacciones con los usuarios. Esta métrica es importante para asegurar que el sistema no se enfoque únicamente en un conjunto pequeño de productos populares, sino que exponga a los usuarios a una gama más amplia de opciones disponibles en el catálogo.

¿Por qué es importante Catalog Coverage?

- Diversidad: Un alto *Catalog Coverage* asegura que el sistema de recomendación promueva ítems diversos, en lugar de solo aquellos que ya son populares.
- Satisfacción del usuario: Al ofrecer una variedad más amplia de ítems, es más probable que los usuarios encuentren productos específicos que se ajusten a sus intereses únicos.
- Valor de negocio: Promover más ítems del catálogo puede llevar a aumentar las ventas de productos menos conocidos o de "larga cola", maximizando el uso de todo el inventario.

Cálculo de *Catalog Coverage*

Catalog Coverage se expresa como el porcentaje de ítems únicos que se recomiendan al menos una vez en relación con el total de ítems disponibles en el catálogo. Su fórmula es la siguiente:

$$\text{Catalog Coverage} = \frac{\text{Número de ítems únicos recomendados}}{\text{Número total de ítems en el catálogo}}$$

Donde:

- *Número de ítems únicos recomendados* es el conteo de los ítems que fueron recomendados al menos una vez en todas las interacciones con los usuarios.
- *Número total de ítems en el catálogo* es el total de ítems que el sistema podría recomendar.

Ejemplo de Cálculo

Imaginemos que tenemos un catálogo de 1000 ítems (productos, películas, etc.), y un sistema de recomendación que ha recomendado 200 de estos ítems en total a los usuarios.

$$\text{Catalog Coverage} = \frac{200}{1000} = 0,2 \quad \text{o bien, } 20\%$$

Esto indica que el sistema de recomendación utiliza solo el 20 % del catálogo en sus recomendaciones. Un Catalog Coverage más alto (por ejemplo, 80 %) reflejaría que el sistema está recomendando una mayor variedad de ítems, mientras que un valor bajo puede indicar que el sistema se enfoca en un subconjunto limitado.

Interpretación de Catalog Coverage

- Un alto *Catalog Coverage* (por ejemplo, 70 %-100 %): El sistema cubre una amplia gama del catálogo, lo cual puede ser positivo en términos de diversidad y satisfacción del usuario.
- Un bajo **Catalog Coverage** (por ejemplo, 0 %-30 %): El sistema utiliza solo una pequeña parte del catálogo, lo que puede llevar a una experiencia limitada y potencialmente menos interesante para los usuarios.

Limitaciones

Aunque un alto *Catalog Coverage* puede mejorar la experiencia del usuario, no siempre es ideal si esto implica recomendar productos menos relevantes. Es necesario un balance para mantener tanto la diversidad como la relevancia en las recomendaciones.

7.3.6. Serendipity

La *Serendipity* o serendipia en sistemas de recomendación se refiere a la capacidad de un sistema para proporcionar a los usuarios recomendaciones inesperadas pero valiosas que no buscaban inicialmente. Este concepto está relacionado con la sorpresa positiva y la satisfacción que puede generar el descubrimiento de ítems o productos que son relevantes para el usuario, aunque no fueran explícitamente solicitados.

Importancia de la *Serendipity*

- Experiencia del usuario: Las recomendaciones serendipitosas pueden enriquecer la experiencia del usuario al exponerlo a opciones que no había considerado, aumentando su satisfacción y *engagement* con el sistema.
- Fidelización: Al sorprender a los usuarios con recomendaciones relevantes, se incrementa la probabilidad de que regresen al sistema, mejorando la lealtad del cliente.
- Diversidad: Promover la serendipia ayuda a diversificar las opciones que se muestran, evitando que el sistema se centre únicamente en ítems populares o en las preferencias ya conocidas del usuario.

Cálculo y Medición de *Serendipity*

$$Serendipity = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{|UNEXP_u \cap REL_u \cap REC_u|}{k} \quad (7.8)$$

Donde:

- $|U|$ es el número de usuarios.
- $UNEXP_u$ es el conjunto de ítems inesperados para el usuario u .
- REL_u es el conjunto de ítems relevantes para el usuario u .
- REC_u es el conjunto de ítems recomendados al usuario u .
- k es el número de recomendaciones consideradas.

Entonces, para cada usuario:

- Se identifican los ítems recomendados(REC_u) en el top-k.
- Se identifican los ítems relevantes(REL_u).
- Se identifican los ítems inesperados($UNEXP_u$), el complemento de los dos primeros.
- Se cuenta cuántos ítems cumplen las tres condiciones.
- Se divide por k .
- Se promedian los valores de todos los usuarios ($|U|$).

El resultado es un valor entre 0 y 1 donde:

- 1 indica que todas las recomendaciones son relevantes e inesperadas.
- 0 indica que no hay recomendaciones serendípicas.

Limitaciones

A pesar de los beneficios, la serendipia debe equilibrarse con la relevancia. Si un sistema ofrece demasiadas recomendaciones inesperadas que no son relevantes, puede frustrar a los usuarios en lugar de sorprenderlos positivamente.

Ejemplo de Cálculo

Supongamos que tenemos un sistema de recomendación con tres usuarios ($|U| = 3$), y cada usuario tiene un conjunto de ítems recomendados, relevantes, e inesperados. Queremos calcular la serendipia promedio de las recomendaciones para estos usuarios considerando $k = 5$ recomendaciones.

Luego para cada usuario tenemos los siguientes ítems:

- Usuario 1:
 - $REC_1 = \{A, B, C, D, E\}$
 - $REL_1 = \{A, C, F\}$
 - $UNEXP_1 = \{A, C\}$
- Usuario 2:
 - $REC_2 = \{G, H, I, J, K\}$
 - $REL_2 = \{G, H, M\}$
 - $UNEXP_2 = \{I, K\}$
- Usuario 3:
 - $REC_3 = \{N, O, P, Q, R\}$
 - $REL_3 = \{N, Q, T\}$
 - $UNEXP_3 = \{N, P\}$

Usamos la fórmula de serendipia para cada usuario:

$$Serendipity = \frac{1}{|U|} \sum_{u=1}^{|U|} \frac{|UNEXP_u \cap REL_u \cap REC_u|}{k}$$

Calculamos para cada usuario:

- Para el usuario 1: $UNEXP_1 \cap REL_1 \cap REC_1 = \{C\}$, entonces:

$$\frac{|UNEXP_1 \cap REL_1 \cap REC_1|}{k} = \frac{2}{5} = 0,4$$

- Para el usuario 2: $UNEXP_2 \cap REL_2 \cap REC_2 = \{H\}$, entonces:

$$\frac{|UNEXP_2 \cap REL_2 \cap REC_2|}{k} = \frac{0}{5} = 0$$

- Para el usuario 3: $UNEXP_3 \cap REL_3 \cap REC_3 = \{Q\}$, entonces:

$$\frac{|UNEXP_3 \cap REL_3 \cap REC_3|}{k} = \frac{1}{5} = 0,2$$

Finalmente, el valor promedio de serendipia para los tres usuarios es:

$$Serendipity = \frac{1}{3}(0,4 + 0 + 0,2) = 0,2$$

8. RESULTADOS

En los siguientes capítulos se presentaran los resultados de las pruebas de evaluación para dos configuraciones distintas del sistema de recomendación. Para esto se reemplaza un componente del sistema de recomendación, encargado de aplicar el *ranking* a los ítems que forman parte del resultado de una consulta. Éste componente es un modelo de lenguaje grande encargado de refinar el ordenamiento de los resultados, el cual se utiliza a lo largo del ciclo de vida del usuario.

Para el sistema de recomendación, el usuario puede tomar dos estados posibles, e ir cambiando entre los mismos dentro de su ciclo de vida. Estos estados definen dos estrategias distintas de recomendación de contenido:

- Estado frio (*Cold*): Para los usuarios en este estado, el sistema de recomendación carece de información sobre su comportamiento, ya que este no ha calificado ningún contenido en el ultimo tiempo. En este estado, solo se dispone de información general del perfil del usuario, limitándose a realizar recomendaciones basadas en su perfil y contenido similar a su consulta (estrategia de basada en contenido). En este estado el modelo utiliza un técnica *RAG(Retrieval Augmented Generation o Generación Aumentada por Recuperación)* [7] para buscar contenido similar a la consulta del usuario, utilizando la distancia Coseno como primer método de *ranking*. Luego se utiliza un modelo de lenguaje grande para refinar el ordenamiento basándose en los resultado de la técnica *RAG(Retrieval Augmented Generation o Generación Aumentada por Recuperación)* [7] e información del perfil del usuario.
- Estado caliente (*Warm*): Para los usuarios en este estado, el sistema de recomendación cuenta con un número mínimo de interacciones, utilizadas para realizar recomendaciones basadas en su comportamiento. Se utiliza un modelo de recomendación basado en filtros colaborativos como primer método de búsqueda y ordenamiento(*ranking*) de ítems, seguido por un modelo de lenguaje grande, que refina este ordenamiento en función de las interacciones(calificaciones) del usuario y el perfil del usuario (similar a la estrategia anterior).

Es importante mencionar que las interacciones del usuario (calificaciones de ítems) tienen una fecha de expiración, lo que implica una ventana de tiempo hacia atrás desde la fecha actual para considerar su comportamiento. Esto garantiza que el sistema de recomendación refleje las preferencias más recientes del usuario. Además, el sistema alternará entre diferentes estrategias de recomendación según la frecuencia de interacciones. Aunque la evaluación del modelo no tiene en cuenta esta ventana temporal, ya que las interacciones ocurren dentro del período de ejecución de la prueba(del orden de días), es importante señalar que el sistema cuenta con esta capacidad.

9. RESULTADOS UTILIZANDO EL COMPONENTE *LLM LLAMA2*

En esta sección se presentan los resultados de la evaluación del sistema de recomendación, donde se utilizó el componente *Llama2* para refinar el *ranking* de los ítems recomendados a partir de una consulta realizada por un usuario perteneciente al conjunto de evaluación.

Para comenzar, en la figura 9.1 se muestra la distribución de la cantidad de pasos por sesión de usuario. Un paso representa una recomendación solicitada por un usuario dentro de su sesión y una sesión puede tener n pasos. Se puede apreciar que la mayoría de los usuarios realizan entre 6 y 13 consultas por sesión (rango inter-cuartíl), siendo 6 consultas el valor más común (moda). Algunos usuarios realizan pocas consultas (< 6), mientras que un pequeño grupo llega a realizar muchas (> 22), considerados *outliers* (4,48 % del total). La asimetría positiva indica que la mayoría de los usuarios tienen un comportamiento entre el promedio y la moda, pero existen algunos usuarios con gran cantidad de interacciones (sesiones largas con muchas consultas). La media (10,59), mayor que la mediana (9), indicando que la distribución está sesgada hacia la derecha, sugiere que estos usuarios extremos están influyendo en el valor del promedio. Las sesiones con más de 22 consultas podrían indicar: Usuarios avanzados/expertos que exploran profundamente el sistema.

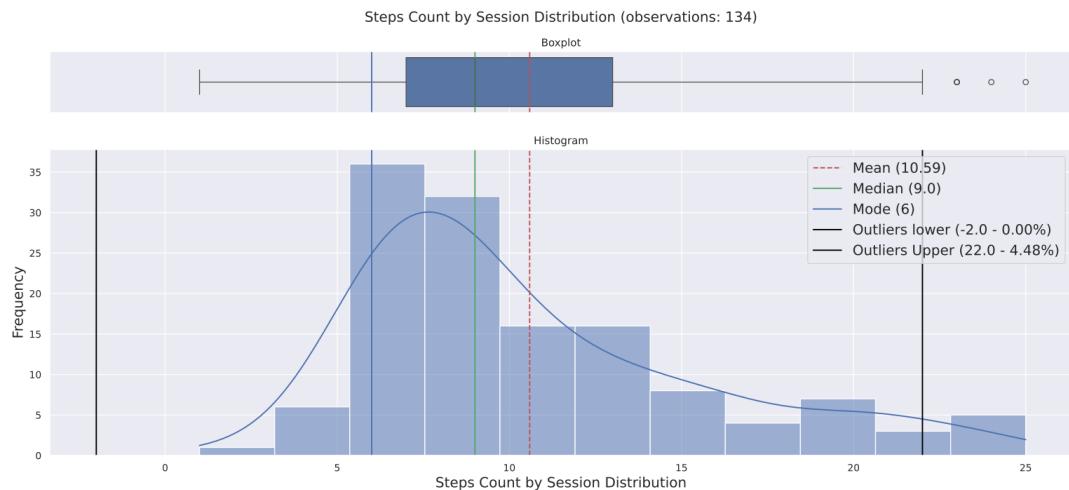


Fig. 9.1: Este gráfico representa la cantidad de pasos por sesión de usuario. Un paso representa una consulta de recomendación realizada por un usuario.

En la figura 9.2 se muestra la cantidad de usuarios(o sesiones activas) por paso. Cada paso es una consulta al sistema de recomendación, indicando una secuencia ordinal de consultas. Es decir, en el paso 1, se registran 134 usuarios que realizaron la primera consulta, en el paso 2, 133 usuarios y así hasta llegar al último paso (25), al cual solo llegó un único usuario. Cabe aclarar que la sesión que realiza el paso 25, también realizó todo los pasos anteriores. Se puede apreciar que se mantiene un nivel muy alto de retención hasta el sexto paso, con solo pequeñas disminuciones. Se aprecia una caída progresiva después del paso 6. A partir del séptimo paso, el número de sesiones disminuye de manera constante. En el paso 10, el número de sesiones ya ha disminuido casi a la mitad (59 sesiones). La reducción

es más drástica a medida que se avanza, hasta que solo unas pocas sesiones alcanzan o superan los 20 pasos. Pocos usuarios llegan a pasos muy altos: A partir del paso 21, el número de sesiones activas es muy bajo, con menos de 10 sesiones. Solo una sesión llega al paso 25.

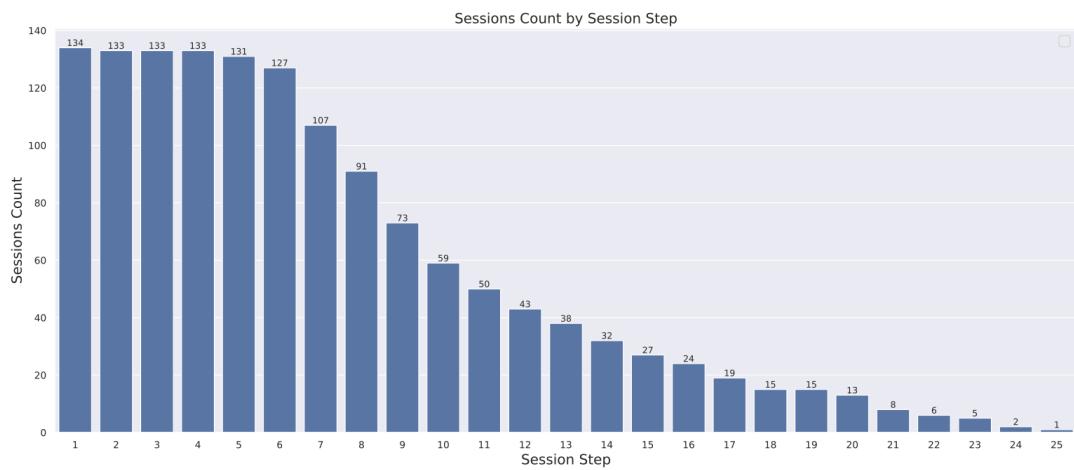


Fig. 9.2: Este gráfico muestra el número de sesiones de usuarios por cada paso de sesión, donde cada paso corresponde a una consulta al sistema de recomendación. Se observa que un usuario llegó a realizar 25 consultas y 134 usuarios completaron el primer paso.

La figura 9.3 muestra la cantidad de ítems relevantes encontrados en los resultados a consultas de usuario. Los ítems relevantes son aquellos cuya interacción es conocida, mientras que los no relevantes son ítems que el usuario no buscó, pero que aparecen en la lista de recomendaciones, ya sea por su similitud a la consulta o por el grado de preferencia que el sistema de recomendación infiere del usuario. Cada consulta devuelve un máximo de 5 ítems relevantes. Los usuarios, por lo general, encuentran entre 2 y 3 ítems por paso, lo que indica que el sistema no siempre localiza todos los ítems solicitados. Esto sugiere que se podría mejorar la búsqueda por similitud (*embeddings*). Una opción sería utilizar *embeddings* generados por *gpt-40* de *OpenAI*. Sin embargo, esto no es un indicador definitivo del rendimiento del sistema, ya que puede que no retorne películas por su nombre exacto (como se realiza en esta prueba de evaluación), pero aun así recomendar opciones relevantes y tener buen desempeño en métricas como *MAP@K* u otras especializadas para evaluar el sistema de recomendación.

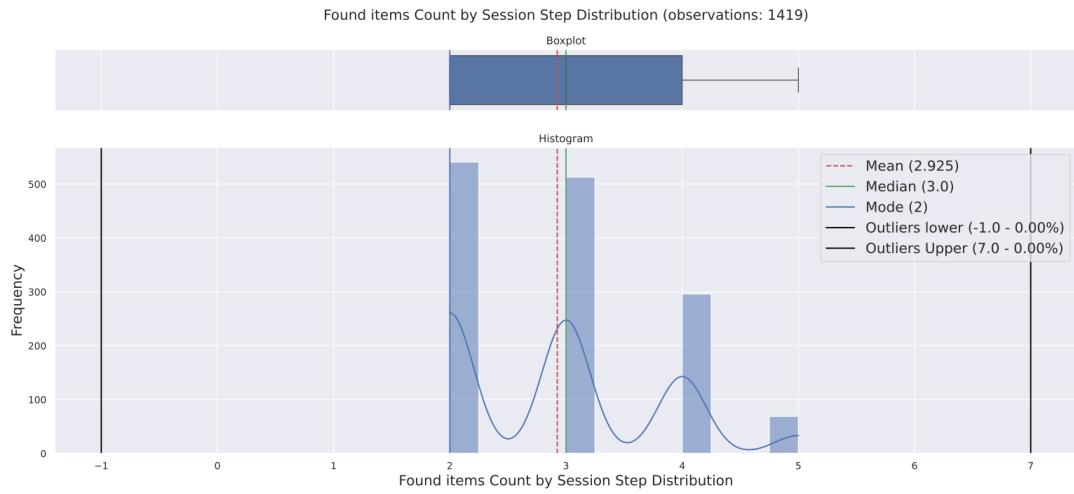


Fig. 9.3: En este gráfico se aprecia la cantidad de ítems relevantes encontrado en cada respuesta del sistema de recomendación. Donde la mayoría de las consultas (Moda) contienen 2 ítems relevantes con una media de 3.

En la figura 9.4 se muestra la distribución de la métrica *Mean Average Precision (MAP)* por sesión de usuario. La distribución muestra que la mayoría de los valores de *MAP* se concentran entre 0,45 y 0,65, con algunos *outliers* en valores más altos. Se aprecian valores clave como la media (0,548), la mediana (0,545) y la moda (0,58). La curva de densidad sobre el histograma indica una ligera asimetría positiva (cola hacia la derecha), lo que sugiere que existen sesiones con *MAP* relativamente alto. No hay *outliers* inferiores, pero los valores superiores (por encima de 0,765) se destacan. Estos puntos podrían indicar sesiones inusuales, tal vez relacionadas con usuarios específicos o películas más populares.

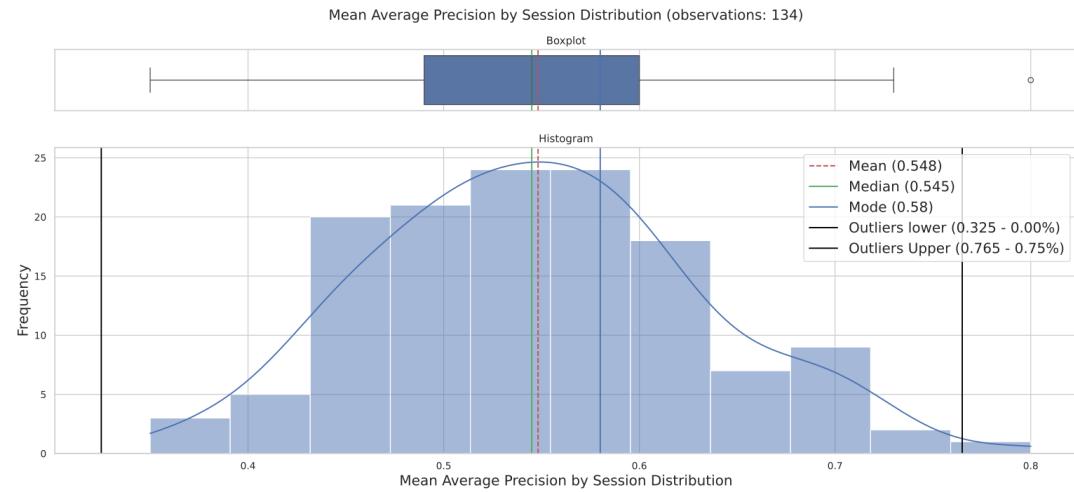


Fig. 9.4: En este gráfico se aprecia la distribución de la métrica *Mean Average Precision (MAP)* por sesión de usuario.

En la figura 9.5 se muestra una serie que describe la evolución del *Mean Average Precision(MAP)* promedio por paso.

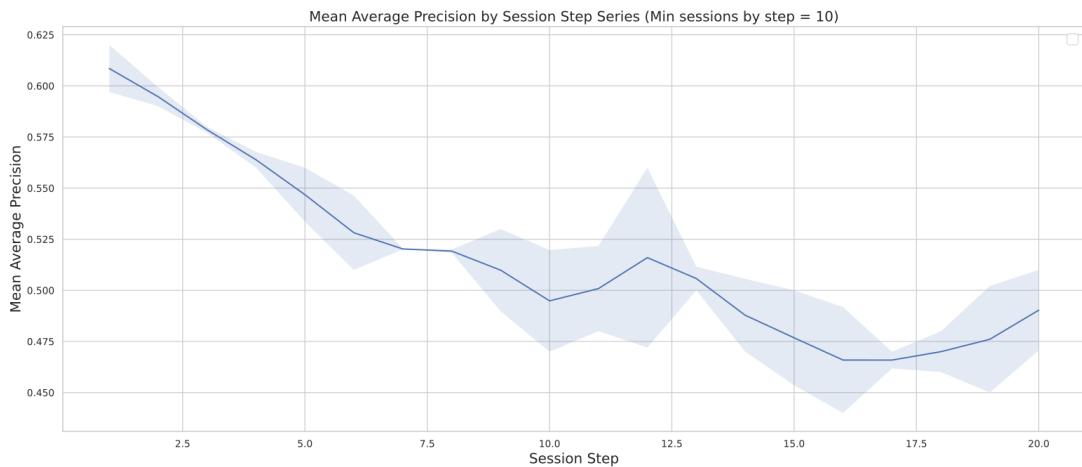


Fig. 9.5: En este gráfico se aprecia una serie que muestra la evolución del de *Mean Average Precision(MAP)* promedio por paso. Es decir se realiza la media de *Mean Average Precision(MAP)* de cada sesión de usuario en cada paso(*Session Step*).

Es decir, se realiza el *Mean Average Precision(MAP)* promedio de cada sesión de usuario en cada paso. La línea azul representa el *Mean Average Precision(MAP)* promedio por paso de sesión, mientras que el área sombreada muestra la intervalo de confianza o la variabilidad de la métrica. Se observa una tendencia decreciente del *Mean Average Precision(MAP)* conforme avanzan los pasos de la sesión (*Session Step*). Esto sugiere que el sistema tiene mayor precisión en las recomendaciones iniciales, pero su desempeño disminuye progresivamente. A partir del paso 10, la tendencia se estabiliza y muestra ligeros altibajos. Esto podría implicar que el sistema llega a un límite en la calidad de las recomendaciones después de varios pasos, posiblemente debido a la menor relevancia o diversidad de los ítems sugeridos. Curiosamente, entre los pasos 17 y 20, el *Mean Average Precision(MAP)* parece mostrar un aumento leve. Esto podría deberse a la disminución de sesiones en estos pasos. Menos sesiones implica mas *leverage* o palanca de valores atípicos altos. El área sombreada es más amplia en los rangos intermedios (8 – 12 y 12 – 15), lo que sugiere una mayor dispersión en los valores de *Mean Average Precision(MAP)*. Esta área corresponde a los pasos donde los usuarios cambian de estado, pasando del estado frío al estado caliente. Este cambio de estrategia utilizada de forma diferenciada por cada usuario puede estar generando distintos valores de la métrica, produciendo la dispersión observada. Por otro lado, el hecho de que el usuario consulte siempre cinco ítems por su nombre indica que el sistema enfrenta dificultades para realizar esta búsqueda por término.

En la figura 9.6 se muestra la media de *Discounted cumulative gain (NDCG)* por paso.

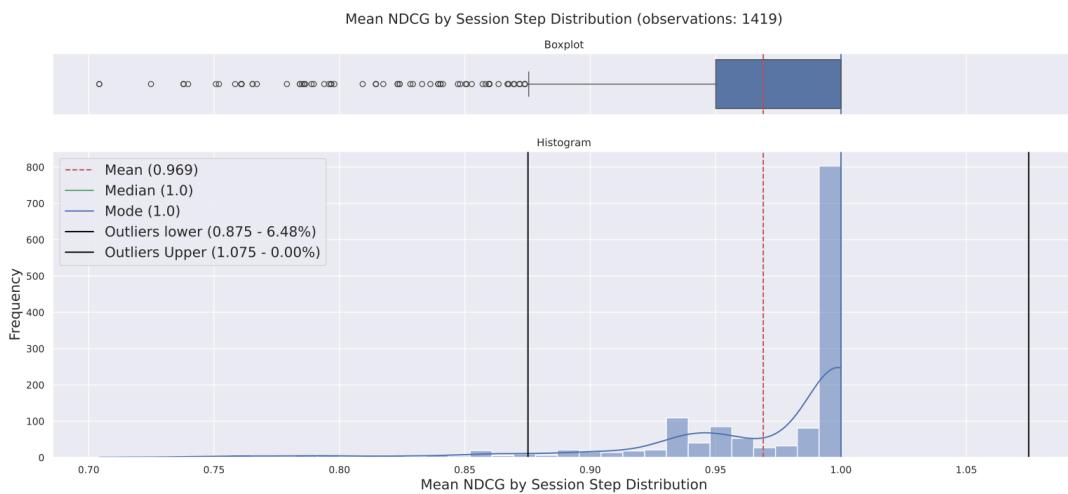


Fig. 9.6: En este gráfico se aprecia la media de *Discounted cumulative gain (NDCG)* por paso.

Para comenzar, se aprecia una distribución altamente sesgada. La mayoría de los valores de *Discounted cumulative gain (NDCG)* están concentrados cerca de 1,0 (la media y mediana son ambas 1,0). Esto sugiere que, en general, el sistema de recomendación logra posicionar ítems relevantes en los primeros lugares de las listas, lo que es un indicador de buen desempeño. Hay una cantidad considerable de *outliers* por debajo de 0,875, representando aproximadamente el 6,48 % de los datos. Estas sesiones podrían indicar casos en los que el sistema falla al generar recomendaciones relevantes. La media de 0,969 refleja un rendimiento global muy bueno. La mediana y moda, ambas iguales a 1,0, refuerzan la idea de que una gran parte de las sesiones tienen un rendimiento ideal. El rango intercuartil (*IQR*) es pequeño, lo que indica poca dispersión entre el 25 % y el 75 % de los datos, nuevamente señalando consistencia en el sistema. En el histograma, se observa un pico pronunciado en 1,0, mientras que los valores fuera de este rango son escasos. Esto confirma que la mayoría de las sesiones tienen un *NDCG* óptimo o cercano a óptimo. Los valores más bajos aparecen de manera dispersa, pero su frecuencia es relativamente baja. A grandes rasgos, el sistema parece funcionar muy bien en términos de *NDCG*, posicionando los ítems relevantes en los primeros lugares en la mayoría de las sesiones.

En la figura 9.7 se muestra la evolución de *Discounted cumulative gain (NDCG)* promedio por paso (*Session Step*).

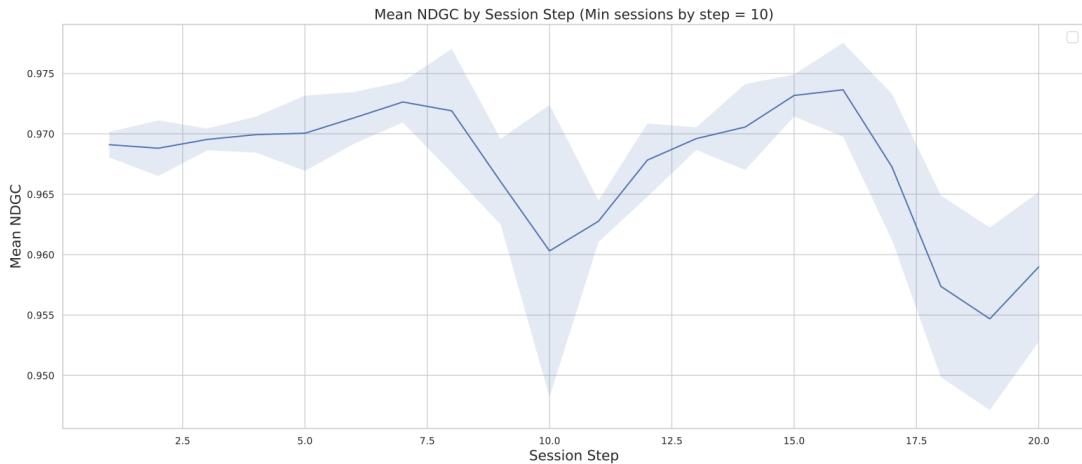


Fig. 9.7: En este gráfico se aprecia la evolución de *Discounted cumulative gain (NDCG)* promedio por paso(*Session Step*).

El *Discounted cumulative gain (NDCG)* promedio se mantiene relativamente alto en todos los pasos de la sesión, con valores superiores a 0,95, lo que demuestra un rendimiento consistente del sistema en términos de relevancia de las recomendaciones. Sin embargo, se observan fluctuaciones notables en ciertos pasos.

Fluctuaciones principales

- **Incremento inicial:** Al principio (hasta el paso 5), el *NDCG* aumenta ligeramente, lo que sugiere que el sistema está afinando las recomendaciones en función de las primeras interacciones del usuario.
- **Descenso en el paso 10:** Hay una caída pronunciada en el *NDCG* promedio alrededor del paso 10, alcanzando un valor cercano a 0,955. Este descenso muestra el cambio entre las estrategias de recomendación basada en contenido versus basada en filtros colaborativos. Se aprecia que en promedio el *NDCG* se recupera en 5 pasos.
- **Variabilidad en pasos finales:** A partir del paso 15, el *NDCG* disminuye nuevamente, pero parece recuperarse ligeramente hacia el paso 20. La amplitud del intervalo de confianza en estos pasos sugiere una mayor incertidumbre o dispersión en el rendimiento.

Intervalos de confianza

Los intervalos de confianza son más amplios en los pasos intermedios y finales (especialmente en los pasos 8–12 y 17–20), lo que indica mayor variabilidad en el rendimiento. Esto puede deberse a:

- **Entre los pasos 8 – 12:**
 - Como ya se mencionó, tenemos una baja debido al intercambio entre estrategias de recomendación.

- Tal vez debería retrasarse el número de pasos en que se cambia la estrategia de recomendación. Este valor es un hiperparámetro a optimizar.
 - Se podría realizar un *smooth* de ambas estrategias para minimizar la baja. Por ejemplo, seleccionando ambas estrategias intercaladas entre los pasos 8 y 12.
- **Entre los pasos 17 – 20:**
 - Usuarios con comportamientos menos predecibles en etapas avanzadas de la sesión.
 - Menor cantidad de datos para esos pasos (sesiones largas son menos comunes).

En la figura 9.8 se muestra el *Recall* promedio por paso(*Session Step*).

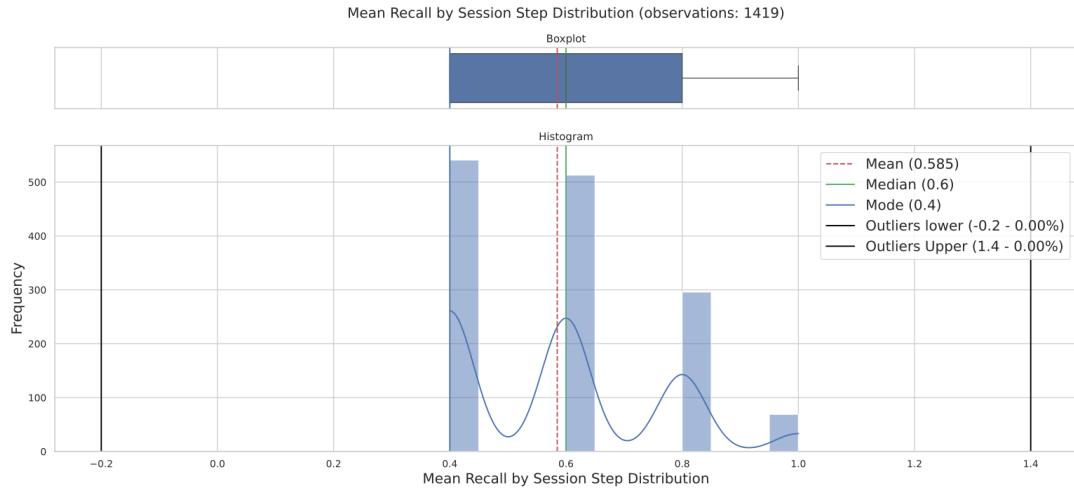


Fig. 9.8: En este gráfico se muestra el *Recall* promedio por paso(*Session Step*)

La media y la mediana permanecen constantes, con un *Recall* medio de 0,585 y una mediana de 0,6. La moda de 0,4 refuerza que hay un grupo considerable de sesiones con desempeño más bajo. No se visualizan valores extremos significativos, con un rango ajustado entre -0,2 y 1,4. La distribución refuerza la predominancia de valores alrededor de la mediana y un pequeño grupo de sesiones con *Recall* más bajo.

En la figura 9.9 se muestra la evolución del *Recall* promedio por paso(*Session Step*).

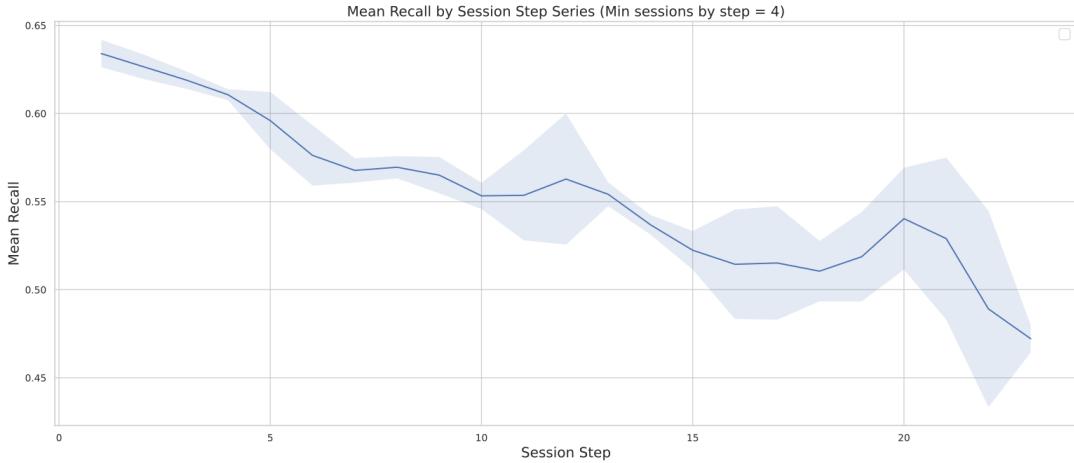


Fig. 9.9: En este gráfico se muestra la evolución del *Recall* promedio por paso(*Session Step*).

La tendencia muestra un descenso en el *Recall* a medida que avanzan los pasos de sesión. El sistema enfrenta dificultades para mantener el desempeño en pasos superiores a 15. La incertidumbre del intervalo de confianza aumenta hacia los pasos finales debido a la menor cantidad de sesiones en esa etapa, algo esperado en distribuciones de series de tiempo. Esto sugiere que el sistema de recomendación tiene problemas para retornar

contenido relevante a medida que aumenta el número de pasos por sesión, ya sea por agotamiento de contenido popular o dificultad para realizar búsquedas detalladas. Por otro lado en parte podría deberse a una dificultad para realizar el *ranking* personalizado.

En la figura 9.10 se muestra la distribución del *Mean Reciprocal Rank (MRR)* por sesión.

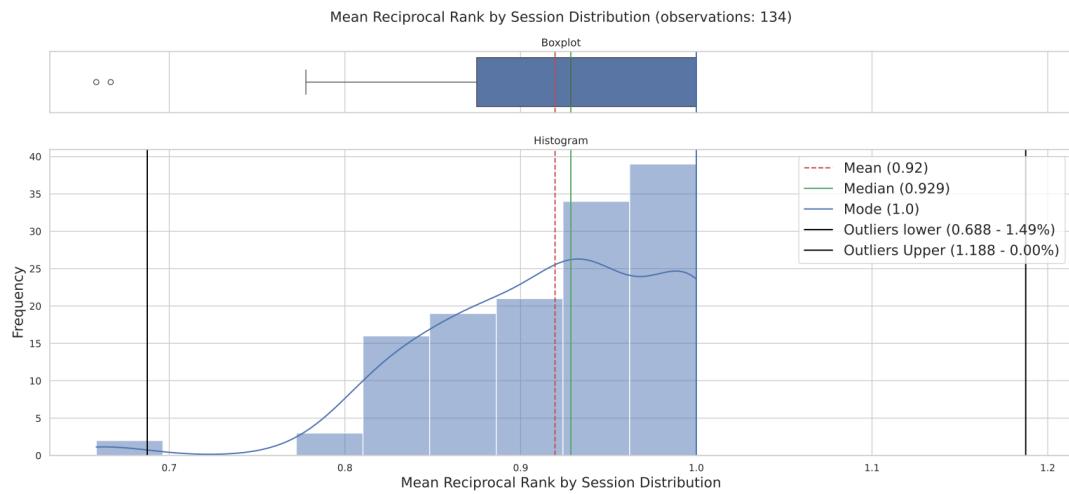


Fig. 9.10: En este gráfico se muestra la distribución del *Mean Reciprocal Rank (MRR)* por sesión.

En la figura 9.10 se observa una distribución sesgada hacia la derecha, con valores de *Mean Reciprocal Rank (MRR)* mayoritariamente altos. Una media de 0,92, mediana de 0,929 y finalmente una moda de 1,0, indicando que muchas sesiones lograron la mejor calificación posible (la recomendación más relevante fue presentada en primer lugar). Por otro lado, existen algunos valores atípicos inferiores (menor a 0,688), que representan aproximadamente el 1,49 % de las observaciones. Estas sesiones pueden haber tenido recomendaciones iniciales menos relevantes, lo que afectó negativamente al *Mean Reciprocal Rank (MRR)*.

En la figura 9.11 muestra una serie con la evolución del *Mean Reciprocal Rank (MRR)* por pasos de sesión.

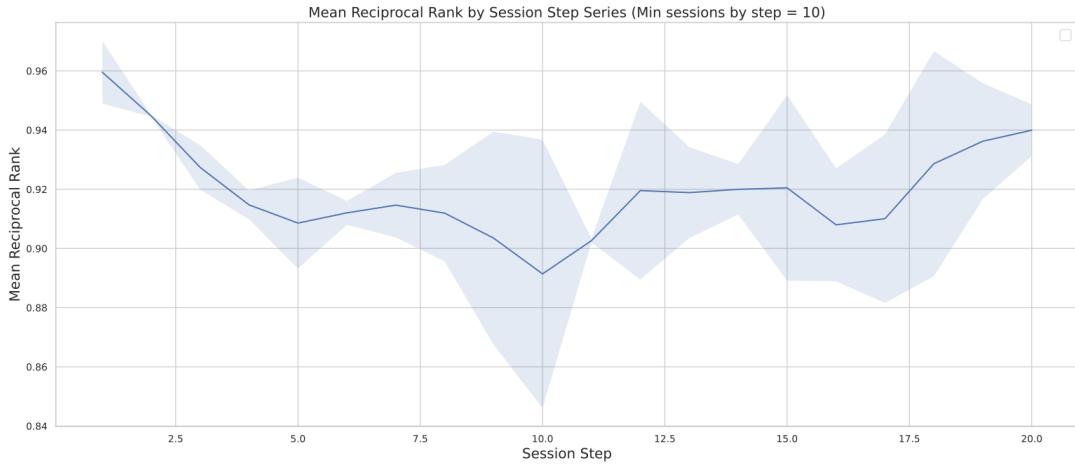


Fig. 9.11: Este gráfico muestra una serie con la evolución del *Mean Reciprocal Rank (MRR)* por pasos de sesión.

El *Mean Reciprocal Rank (MRR)* promedio comienza con valores altos en los primeros pasos (aproximadamente 0,96), lo que sugiere un buen inicio en términos de calidad de recomendaciones. Sin embargo, se observa un descenso gradual hasta el paso 10, donde alcanza su punto más bajo (0,88). Después del paso 10, vuelve a aumentar y se estabiliza cerca de 0,94 en los pasos finales de la sesión.

Este comportamiento se debe al cambio en la estrategia de recomendación basada en contenido (utilizado en las primeras interacciones del usuario), a la estrategia basada en filtros colaborativos. En este caso, a diferencia de la evolución del *NDGC*, la baja parece menos pronunciada. Los intervalos de confianza son amplios en los pasos intermedios y finales, indicando una variabilidad considerable del *MRR* en esos puntos. De esta forma en pasos intermedios la variabilidad de debe al cambio de estrategia de recomendación, que produce que los mejores resultado no siempre se encuentren en las primeras posiciones. Hacia el final, la variabilidad puede deberse a un reducido numero de sesiones afectado por el *leverage* o palanca de alguna sesión particular.

En la figura 9.12 se puede visualizar la distribución de la cantidad de ítems relevantes calificados por sesión de usuario.

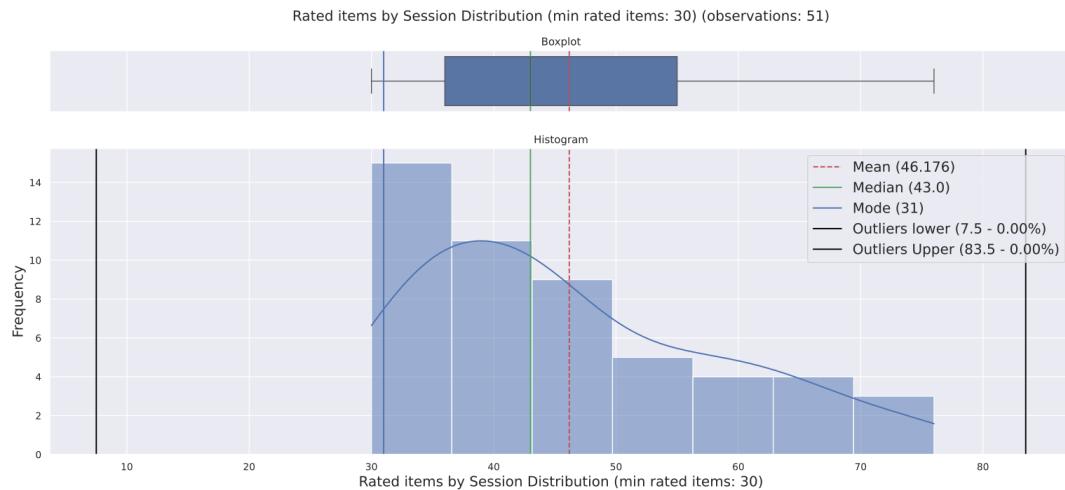


Fig. 9.12: En este gráfico se puede visualizar la distribución de la cantidad de ítems relevantes calificados por sesión de usuario.

En la figura 9.12 se aprecia una distribución unimodal con asimetría positiva. La cola se extiende hacia la derecha y el pico principal se encuentra alrededor de los 30 – 35 ítems. La densidad máxima está en el rango 25 – 45 ítems.

Interpretación por segmentos

- **Segmento medio (30–45 ítems):** Concentra la mayoría de las sesiones. Representa el comportamiento típico de los usuarios. Sugiere un nivel de participación saludable del usuario.
- **Segmento alto (45 – 80 ítems):** Cola larga con frecuencia decreciente. Representa sesiones extensas. Podría indicar usuarios muy comprometidos o experimentados.

La distribución sugiere que el sistema mantiene el interés del usuario durante un período considerable, siendo el rango más común entre 30 – 45 ítems. La ausencia de *outliers* indica un comportamiento bastante consistente entre usuarios.

En la figura 9.13 se muestra el número de ítems relevantes encontrados en cada paso para todas las sesiones de usuario, segmentado por la cantidad de ítems relevantes en las respuestas del sistemas de recomendación.

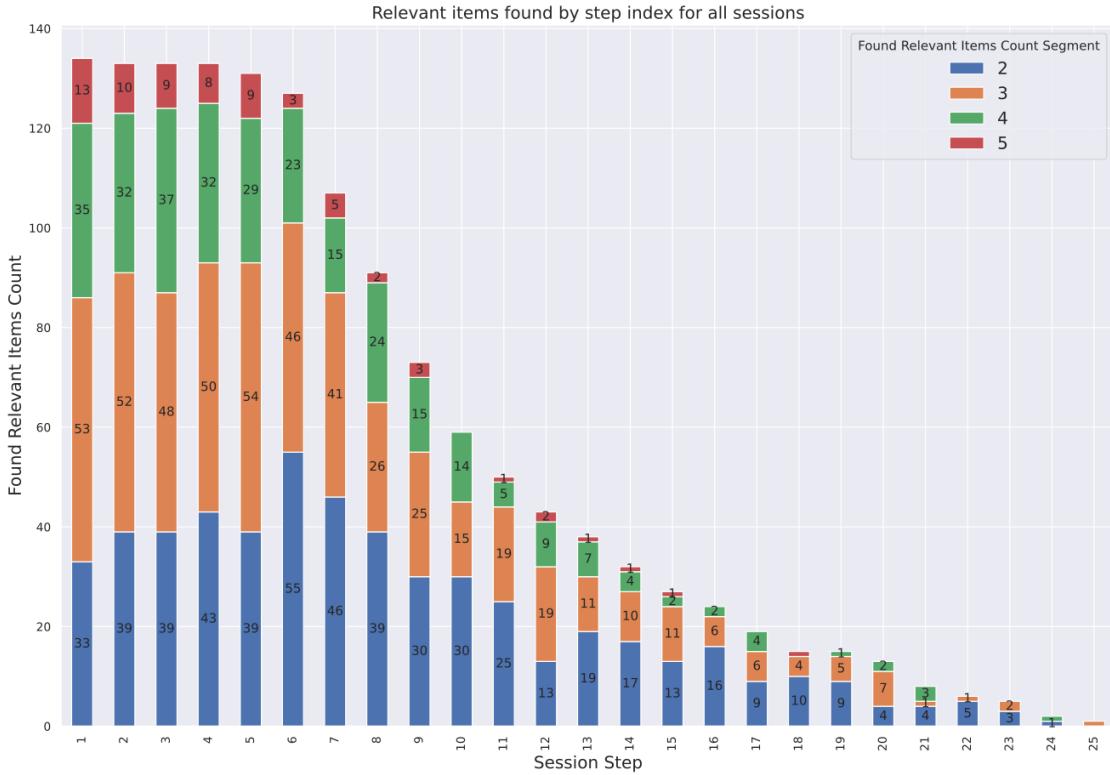


Fig. 9.13: En este gráfico se puede apreciar el número de ítems relevantes encontrados en cada paso para todas las sesiones de usuario, segmentado por la cantidad de ítems relevantes en las respuestas del sistemas de recomendación.

El número total de ítems encontrados disminuye gradualmente a medida que aumenta el numero de pasos de las sesiones. Los pasos iniciales tienen la mayor concentración de relevancia, lo que refleja un modelo más efectivo al inicio de las interacciones. Los segmentos para resultados con 2 y 3 ítems relevantes son dominantes a lo largo de todos los pasos. Los segmentos para resultados con 4 y 5 ítems relevantes aparecen con mayor frecuencia en los pasos iniciales y desaparecen casi por completo hacia los pasos finales. A partir del paso 15, la cantidad total de ítems relevantes es baja, con una tendencia a encontrar menos de 2 ítems relevantes. Esto esta relacionado con la perdida de *Recall* vista en la figura 9.9, donde se aprecia que la perdida del *Recall* a medida que aumenta el número de pasos por sesión. Claramente en sistema de recomendación tiene dificultades para encontrar contenido por nombre a medida que aumenta el número de pasos por sesión.

10. RESULTADOS UTILIZANDO EL COMPONENTE *LLM* *LLAMA3*

En esta sección se presentan los resultados de la evaluación del sistema de recomendación utilizando el componente *Llama3* para realizar el *ranking* de ítems recomendados a un usuario.

En la figura 10.1 se muestra la cantidad de pasos por sesión, donde un paso corresponde a una consulta realizada por un usuario.

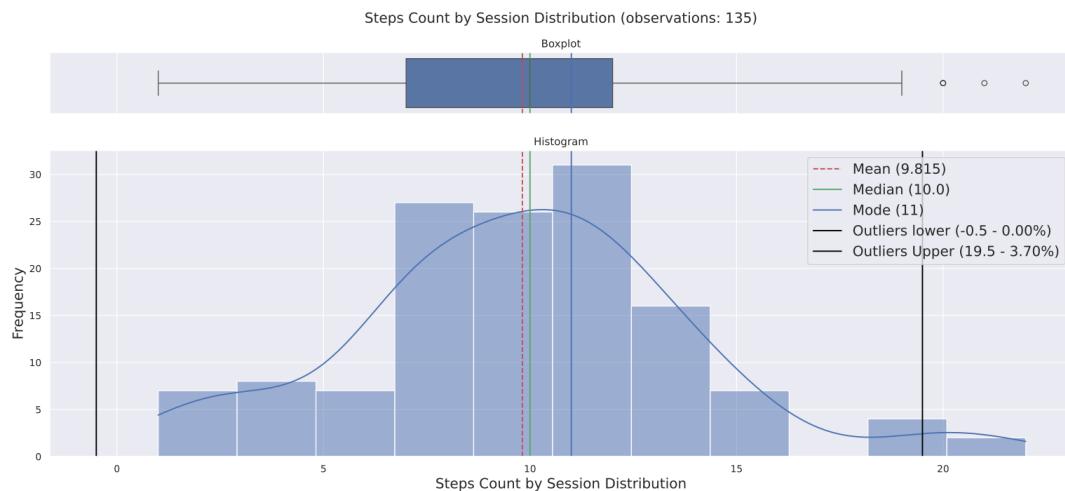


Fig. 10.1: En el gráfico se aprecia la cantidad de pasos(consultas realizadas) por sesión de usuario.

Se puede observar que la mayoría de las sesiones tienen una cantidad de pasos entre 7 y 13. Los valores atípicos superiores (*outliers*) se encuentran por encima de 19 pasos, representando el 3,7% de las observaciones. La media se encuentra en 9,815 pasos por sesión, mientras que la mediana está en 10 pasos, con una moda en 11 pasos. La distribución es aproximadamente simétrica, con un leve sesgo hacia la izquierda, indicando que algunas sesiones terminan antes de alcanzar el promedio. Al utilizar el componente *Llama2*, las sesiones tienen una cantidad promedio de pasos similar, pero la media y mediana muestran valores ligeramente más bajos, lo que sugiere que el sistema con *Llama2* presenta interacciones más cortas en promedio. Utilizando *Llama3* parece que se generan sesiones con una mayor dispersión en su cantidad de pasos (más amplitud en el *boxplot*) y más sesiones con pasos largos (evidenciado por más *outliers* superiores en este caso). El histograma con *Llama2* tiene una concentración más marcada alrededor de 7 – 10 pasos, mientras que en *Llama3* se observa una tendencia a extenderse más hacia los pasos superiores (10 – 15). Con *Llama3* hay un mayor porcentaje de sesiones con pasos fuera de lo esperado (3,7% por encima de 19), lo que puede indicar que este modelo tiende a mantener sesiones más largas para ciertos usuarios o escenarios específicos.

En la figura 10.2 se muestra el número de sesiones que realizaron consultas a lo largo de la secuencia de pasos definida por todas las sesiones generadas durante la prueba de evaluación.

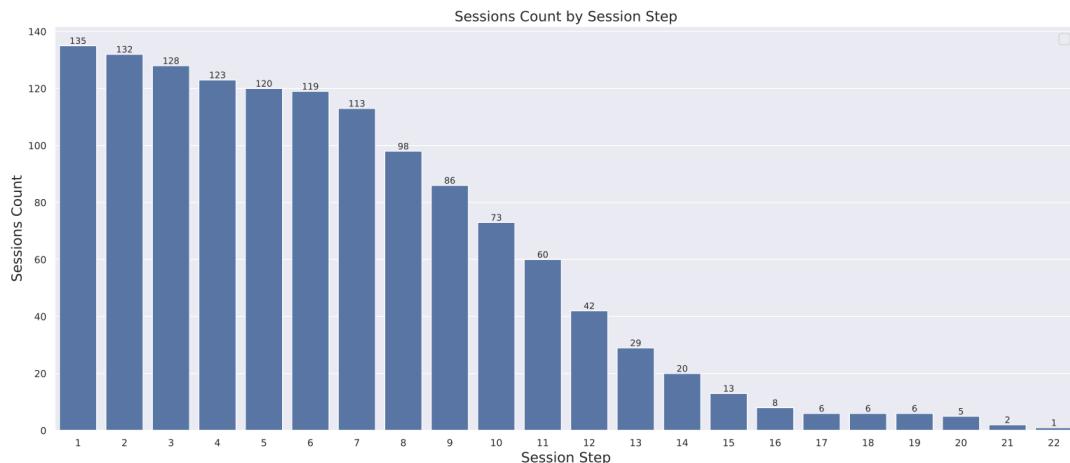


Fig. 10.2: En este gráfico se aprecia el número de sesiones que realizaron consultas a lo largo de la secuencia de pasos definida por todas las sesiones generadas durante la prueba de evaluación.

El mayor número de sesiones se concentra en los primeros pasos: 135 sesiones en el primer paso, 132 en el segundo, con un decrecimiento constante a medida que aumenta el número de pasos. A partir del paso 9, la cantidad de sesiones disminuye más significativamente, llegando a 1 sola sesión en el paso 22. La tendencia muestra que la mayoría de las sesiones terminan antes del paso 10. Esto es coherente con una interacción eficiente donde los usuarios obtienen resultados relevantes en los primeros intentos. Existen sesiones que persisten más allá de los 20 pasos, aunque son minoritarias. Esto indica posibles escenarios donde el modelo no logra satisfacer la necesidad del usuario rápidamente. Con *Llama2*, el conteo de sesiones por paso también disminuye progresivamente, pero el número de sesiones que persisten después del paso 10 son menos en comparación con *Llama3*. Esto sugiere que *Llama3* puede estar generando más recomendaciones en pasos posteriores, aumentando la duración de ciertas sesiones. *Llama3* tiene una mayor proporción de sesiones extendidas (más de 15 pasos). Esto podría deberse a un diseño que busca explorar más alternativas antes de finalizar una sesión o a una dificultad en encontrar contenido relevante en los primeros pasos. En los primeros 5 pasos, ambos modelos parecen manejar un volumen similar de sesiones. Sin embargo, *Llama2* muestra una mayor proporción de sesiones finalizadas dentro de este rango, indicando una mayor eficiencia en cerrar interacciones rápidamente.

En la figura 10.3 se muestra la cantidad de ítems relevantes encontrados en cada respuesta del modelo de recomendación. Un ítem relevante es aquel que cuenta con una calificación del usuario en el conjunto de evaluación. Por lo tanto, si el sistema retorna un ítem que no está en la consulta, consideramos que no es relevante.



Fig. 10.3: En el gráfico se aprecia la cantidad de ítems relevantes encontrado en cada respuesta del modelo de recomendación

La mediana de 3 indicando que en la mayoría de los pasos de sesión se encuentran 3 ítems relevantes. La moda de 2, el número más frecuente de ítems relevantes encontrados por paso. Existe un pequeño porcentaje de valores atípicos (3,32 %) de pasos en los que se encuentran más de 4,5 ítems relevantes, lo que representa situaciones excepcionales. El histograma muestra una clara concentración en 2 y 3 ítems relevantes encontrados, con una rápida caída de frecuencias para valores superiores. El *boxplot* refuerza esta observación, con un rango inter-cuartílico ajustado entre 2 y 3,5 ítems relevantes encontrados. Aunque minoritaria, hay una pequeña proporción de pasos de sesión donde se encuentran más de 4 ítems relevantes. Al utilizar *Llama2*, la media es ligeramente inferior, cercana a 2,5. Esto sugiere que *Llama3* tiene un mejor rendimiento en términos de la cantidad de ítems relevantes encontrados por paso. Ambos modelos comparten una moda de 2 ítems, pero *Llama3* muestra una mediana más alta (3 frente a 2,5 en *Llama2*). Esto indica que el modelo más nuevo tiene un comportamiento más consistente para encontrar más ítems relevantes. Con *Llama2*, los pasos donde se encuentran más de 4 ítems relevantes son aún menos frecuentes, lo que indica que *Llama3* ha mejorado en identificar un mayor número de recomendaciones útiles en casos excepcionales.

En la figura 10.4 se muestra la distribución de la métrica *Mean Average Precision (MAP)* por sesión de usuario.

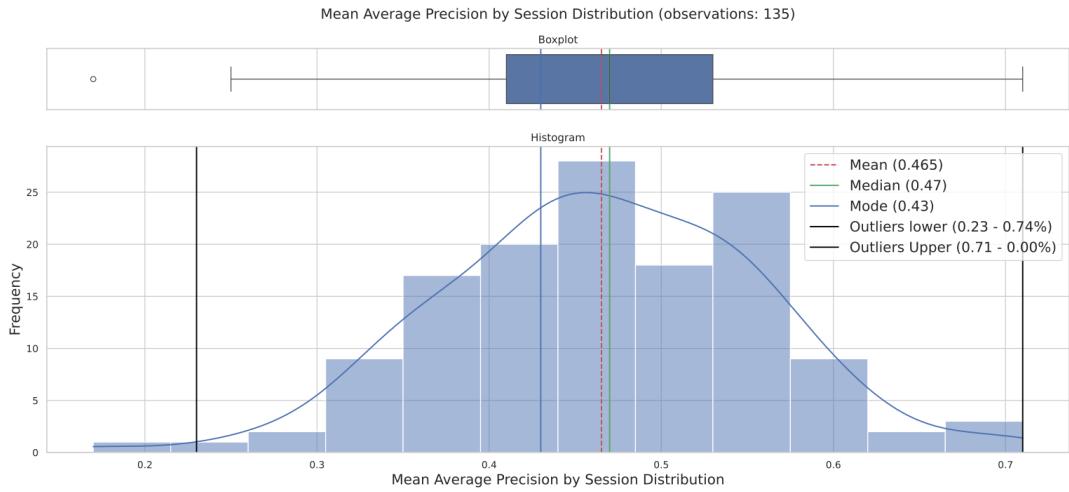


Fig. 10.4: En este gráfico se muestra la distribución de la métrica *Mean Average Precision (MAP)* por sesión de usuario.

El modelo presenta una precisión media de 0,465 (46,5 %), con una mediana de 0,47 (47 %) y una moda de 0,43 (43 %). Esta dispersión entre las medidas de tendencia central sugiere una distribución con cierta asimetría. La forma del histograma revela una distribución más aplanaada indicando variabilidad en el rendimiento del sistema.

El rango de valores se extiende desde aproximadamente 0,2 hasta 0,7, mostrando una amplitud considerable en la precisión de las recomendaciones. Hay presencia de valores atípicos, con *outliers* inferiores a 0,23 (0,74 % de las observaciones), lo que indica casos ocasionales de rendimiento significativamente bajo. La distribución muestra una concentración importante de observaciones en el rango de 0,4 a 0,5, pero también presenta una dispersión considerable hacia ambos extremos. Esta característica sugiere que el sistema, aunque funcional, muestra variabilidad en su capacidad de generar recomendaciones precisas. Al contrastar estos resultados con el sistema basado en *Llama2*, emergen diferencias significativas: *Llama3* muestra una precisión media (0,465) significativamente menor que *Llama 2* (0,548). La diferencia de 8,3 puntos porcentuales representa una brecha considerable en eficacia. *Llama 3* exhibe mayor variabilidad en sus predicciones, evidenciada por su distribución más dispersa. La presencia de múltiples picos contrasta con la distribución más uniforme de *Llama 2*. Los valores atípicos son más frecuentes en *Llama 3*, especialmente en el rango inferior. *Llama 3* opera en un rango más amplio (0,2 – 0,7) comparado con *Llama 2* (0,4 – 0,8). La concentración de valores en rangos más bajos sugiere un rendimiento general inferior. La mayor dispersión en las medidas de tendencia central de *Llama 3* indica menor confiabilidad. Ademas, la presencia de valores extremos más frecuentes sugiere mayor interpretabilidad en las recomendaciones. Las diferencias expuestas hasta el momento sugieren que, la versión del sistema de recomendación que utiliza *Llama3* presenta desafíos significativos en términos de consistencia y precisión cuando se compara con el modelo que utiliza *Llama2*. La mayor variabilidad y menor precisión media indican que podría requerir ajustes o mejoras adicionales para alcanzar el nivel de rendimiento mostrado por el modelo con *Llama2*.

En la figura 10.5 se muestra la evolución de la media de *Mean Average Precision (MAP)* por paso.

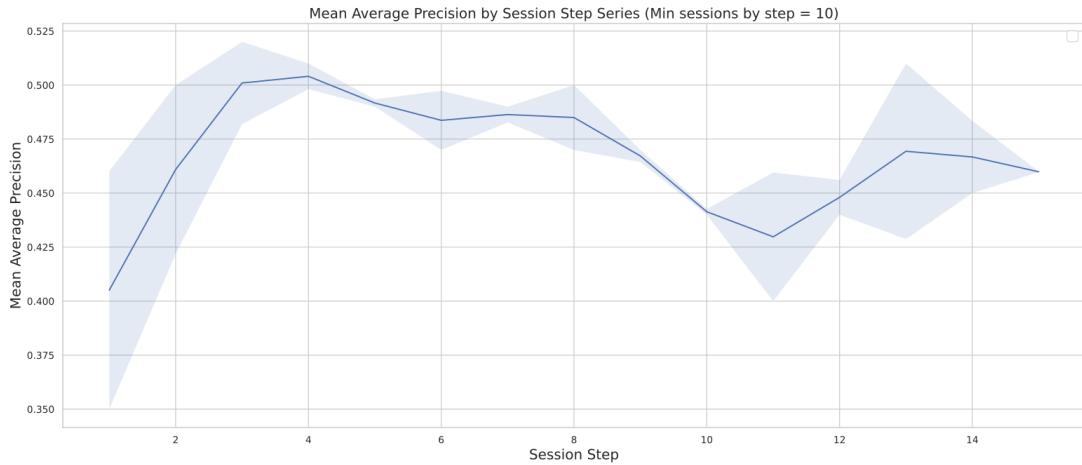


Fig. 10.5: En este gráfico se muestra la evolución la media de *Mean Average Precision (MAP)* por paso.

Se observa un patrón distintivo en su comportamiento a lo largo de las sesiones. El modelo inicia con un rendimiento relativamente bajo, aproximadamente en 0,35, pero muestra una notable capacidad de adaptación durante las primeras sesiones. Esta fase inicial de aprendizaje rápido se caracteriza por un incremento pronunciado en el rendimiento hasta alcanzar un punto máximo alrededor del paso 5, donde la precisión se estabiliza cerca de 0,50.

Tras alcanzar su pico de rendimiento, el modelo mantiene una relativa consistencia en sus predicciones durante las sesiones intermedias (5 – 8), aunque se observa una ligera tendencia descendente. Las últimas sesiones presentan oscilaciones más significativas, sugiriendo cierta inestabilidad en el rendimiento a largo plazo. El intervalo de confianza muestra una mayor incertidumbre tanto en las sesiones iniciales como en las finales, con un período de mayor estabilidad en las sesiones intermedias.

La comparación al utilizar ambos componentes (*Llama 3* y *Llama 2*) revela patrones casi opuestos en su evolución temporal. Mientras que el modelo con *Llama2* exhibe una degradación gradual desde un punto inicial alto, el modelo con *Llama 3* muestra una mejora significativa desde un punto inicial bajo, seguida de una mayor estabilidad. Esta diferencia fundamental en sus comportamientos sugiere distintas fortalezas y casos de uso óptimos para cada modelo.

Resulta particularmente interesante observar cómo ambos modelos tienden a converger a niveles similares de rendimiento hacia el final de las sesiones, aunque llegan a este punto por caminos muy diferentes. Fuera de la diferencias en la evolución del *MAP*, *Llama2* se mantiene desde el inicio con valores de por encima de *Llama3* a pesar de converger a valores similares hacia el final de la serie.

En la figura 10.6 se muestra la distribución de la métrica *Normalized Discounted Cumulative Gain (NDCG)* por paso.

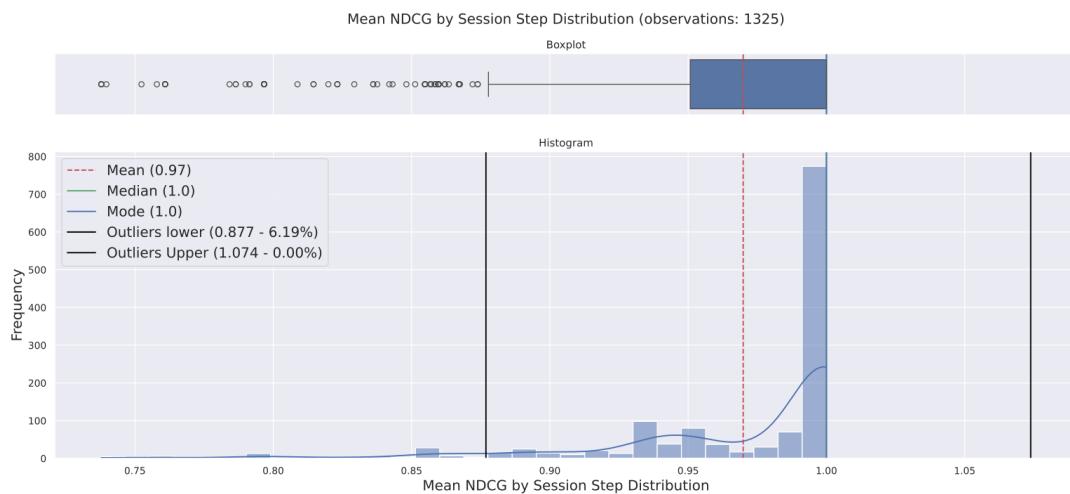


Fig. 10.6: Este gráfico describe la distribución de la métrica *Normalized Discounted Cumulative Gain (NDCG)* por paso.

El gráfico de la figura 10.6 muestra la distribución del *Normalized Discounted Cumulative Gain (NDCG)* por paso basada en 1325 observaciones. La distribución presenta características notables: una media de 0,97, con la mediana y moda coincidiendo en 1,0, lo que indica un sesgo significativo hacia valores óptimos en el rendimiento del sistema de recomendación. El histograma revela una estructura marcadamente asimétrica, donde se destaca un pico prominente en el valor 1,0, que concentra aproximadamente 750 observaciones. Esta característica sugiere que una proporción mayoritaria de las sesiones alcanza un rendimiento óptimo en términos de posicionamiento de recomendaciones relevantes. La distribución muestra una cola izquierda que se extiende hacia valores más bajos, con una concentración significativa de observaciones en el rango entre 0,95 y 1,0. En cuanto a los casos atípicos, se presenta un 6,19 % de *outliers* por debajo de 0,877, mientras que no se observan *outliers* a derecha. El *boxplot* muestra una distribución compacta en el rango inter-cuartil, lo que indica una alta consistencia en el rendimiento del sistema para la mayoría de las sesiones. Al comparar estos resultados con *LLama2*, se observan algunas diferencias interesantes. *LLama2* cuenta con un mayor número de observaciones (1419) y presenta una media ligeramente inferior (0,969). La proporción de *outliers* en *LLama2* es mayor (6,48 %) y su límite inferior es más bajo (0,875). Esta comparación sugiere que *LLama3* logra un manejo más eficiente de los casos problemáticos, aunque mantiene un patrón de distribución similar al de su predecesor. La evolución de *LLama2* a *LLama3* muestra una mejora sutil pero consistente en el manejo de casos atípicos, manteniendo al mismo tiempo la robustez general del sistema de recomendación. Aunque las diferencias no son dramáticas, indican una optimización en el tratamiento de sesiones que podrían presentar desafíos particulares, como perfiles de usuario atípicos o casos con datos problemáticos.

En la figura 10.7 se muestra la evolución de *Discounted cumulative gain (NDCG)* promedio por paso (*Session Step*).

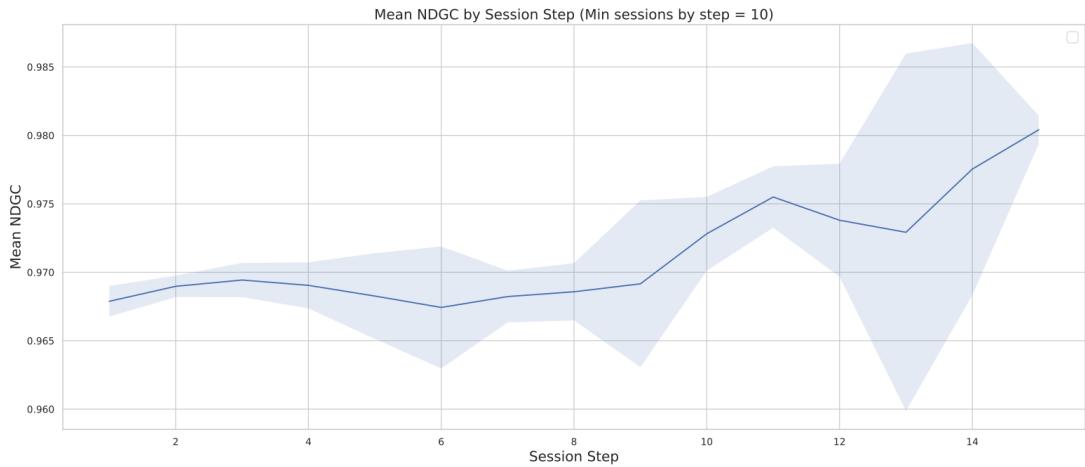


Fig. 10.7: Este gráfico se muestra la evolución de *Discounted cumulative gain (NDCG)* promedio por paso (*Session Step*).

En la figura 10.7 se muestra la evolución del *Discounted cumulative gain (NDCG)*, la cual tiene un patrón notablemente diferente a la medición recabada utilizando el componente *LLama2* (figura 9.7). El gráfico presenta una tendencia general ascendente más estable, comenzando desde aproximadamente 0,965 y alcanzando valores cercanos a 0,98 hacia el final de la sesión. No se observa la pronunciada caída que caracteriza a *LLama2* en el paso 10, lo que sugiere una mejor integración entre las diferentes estrategias de recomendación. Las diferencias más notables entre ambos modelos son:

Al utilizar *LLama3* muestra una mayor estabilidad general, con fluctuaciones menos pronunciadas que *LLama2*. Esto sugiere una mejor gestión en la transición entre diferentes estrategias de recomendación. Mientras que al utilizar *LLama2* muestra una tendencia descendente en los pasos finales, *LLama3* mantiene una tendencia ascendente, indicando una mejor capacidad para mantener la calidad de las recomendaciones en sesiones más largas. *LLama3* presenta intervalos de confianza más amplios hacia el final de la sesión, pero sin la marcada variabilidad que muestra *LLama2* en los pasos 8 – 12 y 17 – 20. *LLama3* parece manejar mejor las sesiones largas, mostrando incluso una mejora en el rendimiento, a diferencia de la degradación observada en *LLama2*.

Estas diferencias sugieren que al utilizar *LLama3* el sistema de recomendación representa una mejora significativa en términos de estabilidad y consistencia en el *Discounted cumulative gain (NDCG)* promedio, especialmente en el manejo de sesiones largas y en la transición entre diferentes estrategias de recomendación. La eliminación de la caída pronunciada en el paso 10 y la tendencia ascendente en los pasos finales indican una mejor adaptación a los patrones de comportamiento del usuario y una mayor robustez en el manejo de casos diversos.

En la figura 10.8 se muestra la distribución del *Recall* promedio por paso (*Session Step*).



Fig. 10.8: Este gráfico muestra la distribución del *Recall* promedio por paso (*Session Step*).

En la figura 10.8 muestra la distribución del *Recall* promedio que presenta 1325 observaciones, con una media ligeramente menor a la obtenida al utilizar *Llama2* de 0,546 y la misma mediana de 0,6 (ver figura 9.8). También se mantiene la moda en 0,4, pero muestra algunas diferencias notables con *Llama2* en su distribución. Se observa la presencia de *outliers* superiores (por encima de 0,9) que representan el 3,32 % de las observaciones, y el límite inferior se establece en 0,1, a diferencia del rango más amplio de *Llama2*. *LLama2* cuenta con más observaciones (1419 vs 1325), lo que podría indicar una ligera diferencia en la cantidad de sesiones procesadas. Las medianas son idénticas (0,6) con ambos modelos. *Llama2* muestra un rendimiento promedio ligeramente superior (0,585 vs 0,546). Ambos mantienen la moda en 0,4, indicando una consistencia en el comportamiento base. Ambos modelos presentan una distribución multimodal similar. *LLama3* muestra una distribución más definida de los valores extremos, con *outliers* claramente identificados. La concentración de valores alrededor de los picos principales es más pronunciada en *LLama3*. Las diferencias del *Recall* promedio por paso entre ambos modelos son sutiles. Al utilizar *LLama2* se muestra un rendimiento promedio ligeramente superior, ya que con *LLama3* los valores mas altos de la métricas son atípicos a diferencia *LLama2* don de esos mismo valores con mas probables.

En la figura 10.9 se muestra la evolución del *Recall* promedio por paso (*Session Step*).

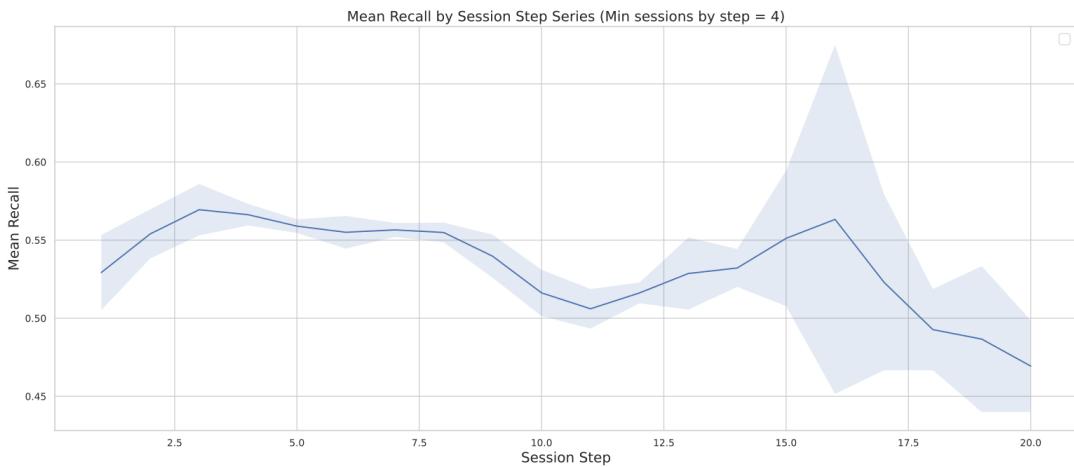


Fig. 10.9: En el gráfico se aprecia la evolución del *Recall* promedio por paso (*Session Step*).

En comparación con la figura 9.9, se puede apreciar un comportamiento más estable, iniciando en 0,52 y manteniendo valores entre 0,50 – 0,57 hasta el paso 15. Después, experimenta una caída similar al utilizar *llama2* (figura 10.9), pero con un intervalo de confianza más amplio entre los pasos 15 – 17. Al utilizar *Llama2* (figura 10.9) la serie inicia con mejor rendimiento pero decae constantemente mientras que utilizando *Llama3* se mantiene mayor estabilidad en pasos intermedios. Ambas series muestran deterioro después del paso 15. Utilizando *Llama3* se exhibe mayor variabilidad en intervalos de confianza. Ademas con *Llama3* se muestra mejor capacidad para mantener el rendimiento en sesiones largas, aunque con mayor incertidumbre. La evolución sugiere que utilizar el componente *Llama3* logra mayor estabilidad en el rendimiento a lo largo de la sesión, aunque sacrifica el *Recall* inicial más alto que se muestra al utilizar el componente *Llama2*.

En la figura 10.10 se muestra la distribución del *Mean Reciprocal Rank (MRR)* por paso sesión.

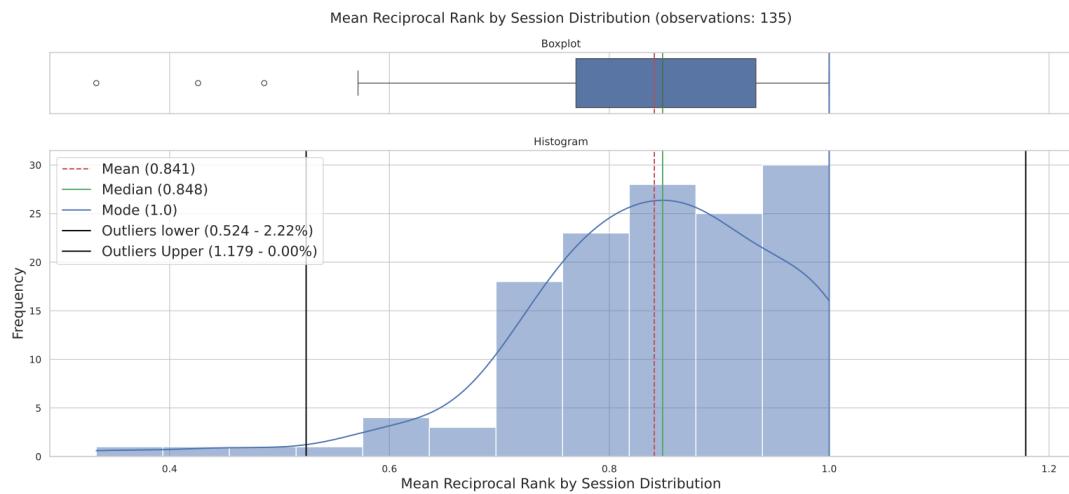


Fig. 10.10: En este gráfico se aprecia la distribución del *Mean Reciprocal Rank (MRR)* por paso sesión.

En la figura 10.10, al observar la distribución de los valores de *Mean Reciprocal Rank (MRR)*, podemos deducir varias cosas sobre cómo se comporta el sistema de recomendación utilizando el componente *Llama3*. En términos generales, al utilizar el *Llama3* parece estar haciendo un trabajo bastante sólido. El valor promedio del *Mean Reciprocal Rank (MRR)* indica que, en promedio, los elementos relevantes se encuentran relativamente cerca del principio de las listas de recomendaciones. Esto sugiere que el modelo está captando las preferencias de los usuarios con una precisión aceptable. Sin embargo, la distribución de los valores no es completamente uniforme. Se observa una cierta dispersión en los resultados, lo que indica que en algunos casos el sistema de recomendación funciona excepcionalmente bien, mientras que en otros su desempeño es menos destacado. Estos casos extremos (*outliers*) podrían ser el resultado de factores como la complejidad de las preferencias del usuario, la calidad de los datos de entrenamiento o incluso errores aleatorios. Un aspecto interesante es que el valor de *Mean Reciprocal Rank (MRR)* más frecuente (la moda) es 1,0. Esto significa que en una proporción significativa de las ocasiones, el sistema logra recomendar el elemento más relevante justo en la primera posición de la lista. Este es un resultado muy positivo y sugiere que el modelo está aprendiendo a captar las preferencias de los usuarios de manera efectiva. Al comparar la distribución del *Mean Reciprocal Rank (MRR)* utilizando ambos componentes *Llama2* y *Llama3*, podemos extraer las siguientes conclusiones clave: Al utilizar *Llama3* se tiende a mostrar un desempeño ligeramente superior en términos de *MRR* promedio, lo que indica que, en general, sus recomendaciones son más precisas. Utilizando *Llama2* se presentan un mayor porcentaje de recomendaciones perfectas (*MRR=1*), lo que podría ser crucial en aplicaciones donde la precisión absoluta es prioritaria. Además al utilizar *Llama3*, suele mostrar una distribución más concentrada alrededor de la media, lo que sugiere un desempeño más consistente, con *Llama2* puede presentar una mayor variabilidad en los resultados, con más casos extremos (*outliers*).

En la figura 10.11 se muestra la evolución del *Mean Reciprocal Rank (MRR)* por pasos de sesión.

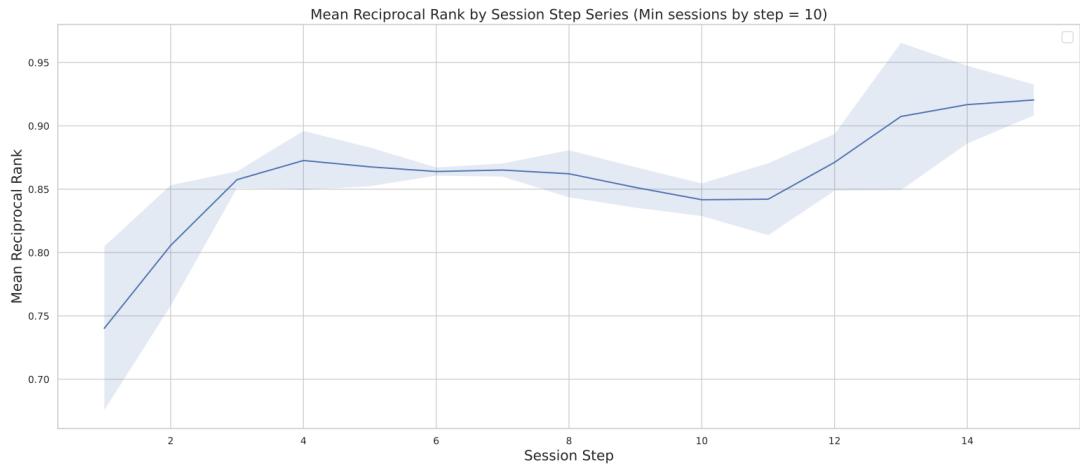


Fig. 10.11: En este gráfico se aprecia la evolución del *Mean Reciprocal Rank (MRR)* por pasos de sesión.

En la figura 10.11 se aprecia que al inicio, el *Mean Reciprocal Rank (MRR)* presenta una tendencia al alza, lo que indica que la calidad de las recomendaciones mejora a medida que el sistema de recomendación se expone a mas interacciones de los usuarios y aprende a ajustar mejor sus parámetros. A partir de cierto punto, el *Mean Reciprocal Rank (MRR)* se estabiliza y fluctúa alrededor de un valor promedio. Esto sugiere que el sistema de recomendación ha alcanzado un nivel de saturación en su aprendizaje. La zona sombreada indica que existe una cierta variabilidad en el *Mean Reciprocal Rank (MRR)* a lo largo de las sesiones. Esto puede deberse a diferentes factores, como la complejidad de los datos, la diversidad de los usuarios o la naturaleza estocástica del proceso de aprendizaje.

En la figura 10.12 se puede visualizar la distribución de la cantidad de ítems relevantes calificados por sesión de usuario.

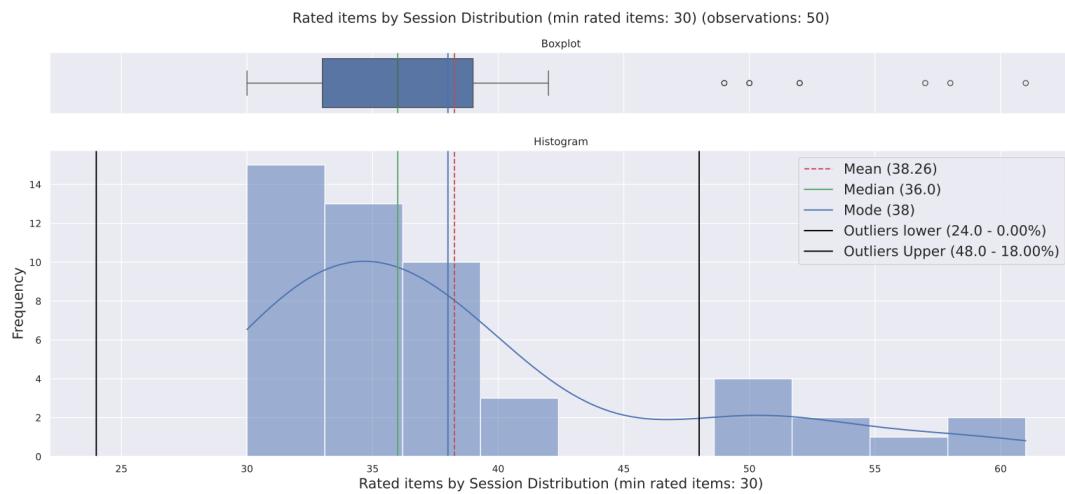


Fig. 10.12: En este gráfico se puede visualizar la distribución de la cantidad de ítems relevantes calificados por sesión de usuario.

La figura 10.12 nos brinda una visión clara sobre la cantidad de ítems que, en promedio, los usuarios evalúan durante cada sesión. Cada punto de datos representa una sesión y el valor asociado indica el número total de ítems que el usuario ha calificado dentro de esa sesión. En promedio, los usuarios evalúan alrededor de 38 ítems por sesión. Existe una considerable variabilidad en el número de ítems evaluados por sesión. Algunos usuarios evalúan un número significativamente mayor o menor de ítems en comparación con el promedio. Esta variabilidad puede deberse a diversos factores, como las características individuales de los usuarios, el tipo de contenido recomendado o el contexto en el que se realiza la evaluación. La distribución de los datos sugiere que la mayoría de las sesiones se concentran alrededor del valor promedio, con una ligera tendencia hacia la derecha. Esto indica que hay un mayor número de sesiones con una cantidad de ítems evaluados superior al promedio. Al comparar los resultados al utilizar los componentes *Llama2* y *Llama3* en cuanto a la cantidad de ítems calificados por sesión, podemos observar algunas diferencias significativas: *Llama2* tiende a generar una mayor cantidad de recomendaciones relevantes por sesión. Esto sugiere que este modelo es más eficaz a la hora de identificar los intereses de los usuarios y presentarles contenido que consideran valioso. Sin embargo, esta mayor cantidad de recomendaciones viene acompañada de una mayor variabilidad en los resultados, lo que significa que la calidad de las recomendaciones puede fluctuar más de un usuario a otro. *Llama3* presenta resultados más consistentes. La cantidad de ítems calificados por sesión es menos variable, lo que indica que la calidad de las recomendaciones es más estable. No obstante, el número promedio de recomendaciones relevantes es ligeramente inferior en comparación con *Llama2*.

En la figura 10.13 se muestra el número de ítems relevantes encontrados en cada paso para todas las sesiones de usuario, segmentado por la cantidad de ítems relevantes en las respuestas del sistemas de recomendación.

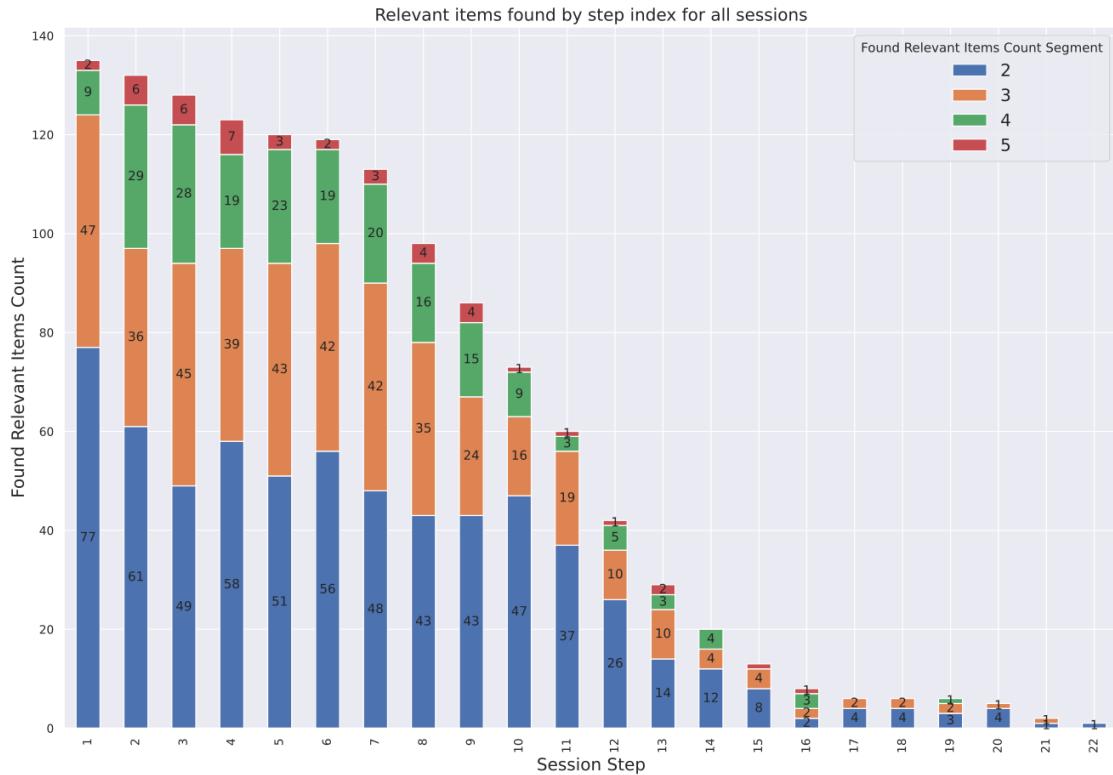


Fig. 10.13: En este gráfico se puede apreciar el número de ítems relevantes encontrados en cada paso para todas las sesiones de usuario, segmentado por la cantidad de ítems relevantes en las respuestas del sistemas de recomendación.

En la figura 10.13 se observa una disminución en la cantidad de ítems relevantes a medida que avanza la sesión. Esto podría indicar que el sistema es más efectivo en encontrar ítems relevantes al inicio de la sesión, y que a medida que se exploran más opciones, la relevancia de las recomendaciones puede disminuir. Existe una gran variabilidad entre las diferentes sesiones. Algunas sesiones logran encontrar una gran cantidad de ítems relevantes en los primeros pasos, mientras que otras mantienen un nivel más constante a lo largo de toda la sesión. La cantidad de ítems relevantes encontrados parece estar influenciada por el paso de la sesión. En los primeros pasos, se suelen encontrar más ítems relevantes, mientras que en los últimos pasos la cantidad tiende a disminuir. Al utilizar *Llama2* se muestra una mayor consistencia en sus recomendaciones, lo que se traduce en una menor variabilidad entre sesiones. Esto sugiere que *Llama2* es más confiable a la hora de proporcionar resultados similares para diferentes usuarios en situaciones similares. Sin embargo, podría ser que *Llama3* sea más innovador y capaz de sugerir ítems más sorprendentes o menos convencionales, aunque con una mayor tasa de aciertos y fallos. *Llama3*, por su parte, podría destacar en la capacidad de adaptarse rápidamente a las preferencias del usuario, alcanzando un pico de relevancia más temprano en la sesión. Esto indica que *Llama3* podría ser más eficaz en captar los intereses iniciales del usuario. No obstante, esta

rapidez en la adaptación podría venir acompañada de una mayor tasa de falsos positivos en etapas posteriores de la sesión.

11. COMPARACIÓN DE RESULTADOS

Al comparar el modelo de recomendación utilizando los componentes *llama2* y *llama3*, se observan ciertas diferencias:

- *Normalized Discounted Cumulative Gain (NDCG)*: Ambos componentes presentan diferencias mínimas. La mayoría de las sesiones logran un rendimiento óptimo en el posicionamiento de ítems relevantes. Ambas distribuciones tienen una cola izquierda que se extiende hacia valores bajos, con una notable concentración de observaciones entre el 95 % y el 100 %. El *boxplot* refleja una distribución compacta en el rango inter-cuantil, lo que sugiere alta consistencia en el rendimiento del sistema para la mayoría de las sesiones. Por otro lado, en la evolución del *NDCG*, *LLama3* muestra una mayor estabilidad general, con fluctuaciones menos pronunciadas a diferencia de *LLama2*. Esto sugiere una mejor gestión en la transición entre diferentes estrategias de recomendación. Mientras que al utilizar *LLama2* se muestra una tendencia descendente en los pasos finales, *LLama3* mantiene una tendencia ascendente, indicando una mejor capacidad para mantener la calidad de las recomendaciones en sesiones más largas.
- *Mean Average Precision (MAP)*: Al emplear el modelo con el componente *LLama3*, se observa una mayor precisión en el posicionamiento de ítems relevantes en las primeras posiciones. Para esta métrica, los ítems relevantes son aquellos calificados por el usuario entre 4 y 5 puntos. Por otro lado, si observamos la evolución temporal de *Mean Average Precision (MAP)* vemos patrones casi opuestos. Mientras que el modelo con *LLama2* exhibe una degradación gradual desde un punto inicial alto, el modelo con *LLama3* se mantiene siempre por debajo de *LLama2* pero con un comportamiento más estable. *Mean Average Precision (MAP)* es una métrica más estricta en términos de puntuación en comparación a *Normalized Discounted Cumulative Gain (NDCG)*, donde solo se considera el orden de los ítems sin requerir un mínimo en su puntuación. Se puede decir que *MAP* y *NDCG* son métricas complementarias en esta área de evaluación.
- *Mean Reciprocal Rank (MRR)*: Al utilizar *LLama3* se tiende a mostrar un desempeño ligeramente superior en términos de *MRR* promedio, lo que indica que, en general, sus recomendaciones son más precisas. Además al utilizar *LLama3*, se puede visualizar una distribución más concentrada alrededor de la media, lo que sugiere un desempeño más consistente, con *LLama2* se presenta una mayor variabilidad en los resultados, con más casos extremos (*outliers*).
- Recall: Al observar la distribución del Recall promedio por paso, las diferencias entre ambos modelos son sutiles. Al utilizar *LLama2* se muestra un rendimiento ligeramente superior, ya que con *LLama3* los valores más altos de la métrica son atípicos a diferencia de *LLama2*, donde esos mismos valores son más probables. En cuanto a la evolución por paso, con *LLama3* se puede apreciar un comportamiento más estable. Al utilizar *LLama2* la serie comienza con mejor rendimiento pero decrece constantemente, mientras que utilizando *LLama3* se mantiene mayor estabilidad en pasos intermedios. La evolución sugiere que utilizar el componente *LLama3* logra mayor estabilidad en el rendimiento a lo largo de las sesiones.

- Cantidad de ítems relevantes por paso: Al utilizar ambos modelos, se observa una gran variabilidad entre las sesiones. Algunas logran identificar más ítems relevantes en los primeros pasos, mientras que otras mantienen un nivel más constante a lo largo de la sesión. A medida que avanzan los pasos, la cantidad de ítems relevantes tiende a disminuir.

El modelo Llama2 muestra menos dificultad para encontrar ítems relevantes en comparación con Llama3. A lo largo de las sesiones, Llama2 logra identificar más resultados con 4 y 5 ítems relevantes. No se observan diferencias significativas con 2 y 3 ítems, ya que ambos modelos presentan cantidades similares por paso. Aunque las diferencias son muy sutiles, parece que Llama2 tiene menos dificultades para encontrar ítems relevantes.

Se observa que una opción viable sería utilizar el componente *llama3*, ya que presenta una tendencia más estable en términos de *NDGC*. Además, muestra mayor precisión en términos del *MAP* y *MRR* promedio. Aunque no hay grandes diferencias en Recall y en la cantidad de ítems relevantes, se puede notar que *llama2* tiene un mayor recall promedio, evidenciado por un mayor número de resultados con 4 y 5 ítems relevantes. La elección entre modelos dependerá de la prioridad al recomendar ítems; un modelo con menor recall también ofrece más oportunidades de descubrir nuevo contenido similar. Desde esta perspectiva, *llama3* podría ser un mejor candidato. Sin embargo, *llama2* no tiene un recall perfecto y enfrenta el mismo desafío.

12. CONCLUSIONES

En este trabajo se exploró el uso de grandes modelos de lenguaje en sistemas de recomendación, analizando su impacto en la personalización de contenidos y la experiencia del usuario [3]. A través de la integración de técnicas como *Retrieval-Augmented Generation (RAG)* [7] y filtros colaborativos, se logró mejorar la capacidad del sistema para ofrecer recomendaciones adaptadas a las preferencias de cada usuario.

Uno de los principales hallazgos fue la limitación en la memoria de los grandes modelos de lenguaje locales, lo que afectó la capacidad de recordar interacciones previas y generar recomendaciones consistentes. Asimismo, se identificaron problemas en la aleatoriedad de los modelos, que en algunas instancias generaban respuestas con estructuras inconsistentes, lo que requirió el desarrollo de *parsers* especializados.

En términos de eficiencia computacional, se constató que el tiempo de inferencia de los grandes modelos de lenguaje locales es significativamente mayor en comparación con soluciones en la nube como *Claude AI* o *ChatGPT-4o* de *Open AI*. Esto plantea un desafío en la escalabilidad del sistema, ya que el costo de los recursos computacionales podría volverse prohibitivo a gran escala.

El problema de *Cold Start* o arranque en frío, característico de los sistemas de recomendación, se abordó mediante la utilización de *embeddings* utilizando la técnica *RAG(Retrieval Augmented Generation o Generación Aumentada por Recuperación)* [7], permitiendo mejorar la calidad de las recomendaciones iniciales sin requerir interacciones previas del usuario. No obstante, se observó que, con el tiempo, la relevancia del contenido recomendado disminuye a medida que se agotan los elementos más relevantes dentro del catálogo disponible.

Como parte del *pipeline* de entrenamiento y actualización de modelos, se utilizó *Apache Airflow* [1] para orquestar los flujos de datos y garantizar la actualización continua de *embeddings* y modelos de recomendación. Este enfoque demostró ser eficiente para manejar grandes volúmenes de datos y asegurar la disponibilidad de modelos actualizados.

En futuras investigaciones, se planea explorar modelos híbridos que combinen la eficiencia de los grandes modelos de lenguaje locales con la capacidad de procesamiento de modelos en la nube, así como optimizar técnicas de *prompt engineering* para mejorar la interpretabilidad de las respuestas generadas por los grandes modelos de lenguaje. Además, podría analizarse la incorporación de estrategias de aprendizaje continuo (*Continual Learning o Lifelong Learning*) que permitan adaptar las recomendaciones de manera dinámica según la evolución de las preferencias del usuario.

Este trabajo contribuye al estudio de la aplicación de modelos de lenguaje en sistemas de recomendación, destacando tanto sus beneficios como sus limitaciones y estableciendo una base para futuras optimizaciones en este campo.

13. GLOSARIO

- Filtrado Colaborativo* : Método de recomendación que se basa en las preferencias y comportamientos de múltiples usuarios para sugerir ítems. Se divide en dos enfoques: basado en usuarios y basado en ítems.
- Recomendación Basada en Contenido* : Técnica que sugiere ítems similares a aquellos que el usuario ha mostrado interés previamente, utilizando las características del contenido.
- Modelos Híbridos* : Sistemas de recomendación que combinan múltiples técnicas, como filtrado colaborativo y basado en contenido, para mejorar la precisión de las sugerencias.
- Retroalimentación Implícita* : Datos obtenidos indirectamente sobre las preferencias del usuario, como clics, tiempo de visualización o historial de navegación, en contraste con la retroalimentación explícita como calificaciones o reseñas.
- Recomendación Basada en Contexto* : Método que tiene en cuenta el contexto del usuario, como ubicación, hora del día o dispositivo utilizado, para ofrecer sugerencias más relevantes.
- Serendipia (Serendipity)* : Métrica que evalúa la capacidad del sistema para sorprender al usuario con recomendaciones inesperadas pero relevantes.
- Sistemas de Recomendación Basados en Popularidad* : Método que sugiere ítems que son populares entre todos los usuarios, sin personalización individual.
- Filtrado Basado en Conocimiento* : Técnica que utiliza información específica sobre los ítems y las necesidades del usuario para realizar recomendaciones.
- Sistemas de Recomendación Basados en Redes Neuronales* : Modelos que emplean redes neuronales para capturar relaciones complejas entre usuarios e ítems y mejorar las recomendaciones.
- LLM (Large Language Models)* : Modelos de lenguaje de gran escala entrenados con grandes volúmenes de datos para comprender y generar texto de manera sofisticada.
- Retrieval-Augmented Generation (RAG)* : Técnica que combina recuperación de información y generación de texto para mejorar la precisión y relevancia de las respuestas generadas por modelos de lenguaje.
- Cold Start* : Problema en los sistemas de recomendación que ocurre cuando no hay suficiente información sobre un usuario o ítem para generar recomendaciones precisas.
- Embeddings* : Representaciones vectoriales de palabras, frases o elementos de datos que permiten medir similitudes en un espacio continuo.
- DeepFM* : Modelo basado en redes neuronales profundas y factorization machines para mejorar la predicción en sistemas de recomendación.
- Apache Airflow* : Plataforma de orquestación de flujos de trabajo utilizada para programar, gestionar y monitorear procesos de datos.
- Prompt Engineering* : Técnica para diseñar instrucciones o entradas optimizadas que mejoran la calidad de las respuestas de modelos de lenguaje.
- Aprendizaje Continuo* : Estrategia de machine learning en la que un modelo se actualiza dinámicamente con nuevos datos sin necesidad de ser re-entrenado completamente desde cero.
- Factorization Machines (FM)* : Algoritmo utilizado en sistemas de recomendación para

modelar interacciones entre variables categóricas y continuas.

Transfer Learning : Técnica de aprendizaje automático en la que un modelo pre-entrenado en un dominio se ajusta a un nuevo dominio con menos datos.

NDCG (Normalized Discounted Cumulative Gain) : Métrica utilizada para evaluar la calidad de las recomendaciones basadas en la relevancia y el orden de los ítems recomendados.

MAP (Mean Average Precision) : Promedio de las precisiones obtenidas en cada consulta en un sistema de recomendación.

MRR (Mean Reciprocal Rank) : Promedio del recíproco de la posición del primer resultado relevante en una lista de recomendaciones.

ChromaDB Base de datos orientada a almacenamiento de embeddings utilizada para búsqueda de similitud en recomendaciones.

REFERENCIAS

- [1] Airbnb. *Apache Airflow*. Fecha de acceso: 2022. 2014. URL: <https://airflow.apache.org>.
- [2] ROUNAK BANIK. *TMDB Movie Dataset*. Fecha de acceso: 2 de febrero de 2021. 2017. URL: https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset?select=movies_metadata.csv.
- [3] Yunfan Gao et al. *Chat-REC: Towards Interactive and Explainable LLMs-Augmented Recommender System*. Fecha de acceso: 23 de agosto de 2023. 2023. URL: <https://arxiv.org/abs/2303.14524>.
- [4] GroupLens. *MovieLens 25M Dataset*. Fecha de acceso: 2 de febrero de 2021. 2019. URL: <https://grouplens.org/datasets/movielens/25m>.
- [5] Jeremiah Harmsen HengTze Cheng Levent Koc. *Wide and Deep Learning for Recommender Systems*. Fecha de acceso: 2 de marzo de 2021. 2016. URL: <https://arxiv.org/pdf/1606.07792.pdf>.
- [6] Yunming Ye Huifeng Guo Ruiming Tang. *DeepFM: A Factorization-Machine based Neural Network for CTR Prediction*. Fecha de acceso: 2 de febrero de 2021. 2021. URL: <https://arxiv.org/pdf/1703.04247.pdf>.
- [7] Patrick Lewis. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. Fecha de acceso: 2023. 2005. URL: <https://arxiv.org/abs/2005.11401>.
- [8] Adrian Marino. *Sistemas de recomendación colaborativos, Trabajo Final de la Especialización en Explotación de Datos y Descubrimiento del Conocimiento*. Fecha de acceso: 1 de enero de 2022. 2022. URL: <https://github.com/adrianmarino/thesis-paper/blob/master/docs/specialization-project/thesis.pdf>.
- [9] Steffen Rendle. *Factorization Machines*. Fecha de acceso: 2 de febrero de 2022. 2010. URL: <https://www.csie.ntu.edu.tw/~b97053/paper/Rendle2010FM.pdf>.
- [10] Comunidad de editores de Wikimedia. *Análisis de componentes principales*. Fecha de acceso: 23 de julio de 2023. 2023. URL: [https://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales#:~:text=En%20estad%C3%ADstica%2C%20el%20an%C3%A1lisis%20de,%C2%ABcomponentes%C2%BB\)%20no%20correlacionadas..](https://es.wikipedia.org/wiki/An%C3%A1lisis_de_componentes_principales#:~:text=En%20estad%C3%ADstica%2C%20el%20an%C3%A1lisis%20de,%C2%ABcomponentes%C2%BB)%20no%20correlacionadas..)
- [11] Comunidad de editores de Wikimedia. *Biplot*. Fecha de acceso: 23 de julio de 2023. 2023. URL: <https://es.wikipedia.org/wiki/Biplot>.
- [12] Aston Zhang et al. *Deep Factorization Machines*. Fecha de acceso: 20 de junio de 2022. 2021. URL: https://d2l.ai/chapter_recommender-systems/deepfm.html.
- [13] Aston Zhang et al. «Dive into Deep Learning». En: *arXiv preprint arXiv:2106.11342* (2021).
- [14] Aston Zhang et al. *Factorization Machines*. Fecha de acceso: 20 de junio de 2022. 2021. URL: <https://medium.com/qloo/popular-evaluation-metrics-in-recommender-systems-explained-324ff2fb427d#:~:text=Precision%20and%20recall%20are%20evaluation,user%20query%20in%20our%20case..>