Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Facultad de Ingeniería

Sistemas de Recomendación Colaborativos

Trabajo Final de la *Especialización en Explotación de Datos y Descubrimiento del Conocimiento*

Proyecto en Gihub

Adrian Norberto Marino

Buenos Aires, 2022

RESUMEN

Este trabajo cubre la comparación de sistemas de recomendación basados en filtros colaborativos. Es una explicación exhaustiva del funcionamiento e implementación de la batería de modelos de recomendación colaborativos mas utilizados como son: *GMF*, *Biased-GMF*, *KNN Item Based*, *KNN User Based*, *DeepFM* y *NN-FM*. Se pretende comparar todos los modelos, utilizando métricas especializadas como el promedio de la precisión ($AP@K$) y la media del promedio de la precisión ($mAP@k$), y otras menos especializada como la raíz del error cuadrático medio *RMSE*. Todos los modelos se entrenaron utilizando el mismo *dataset*, construido a partir de los *datasets TMDB* y *Movie Lens*. A grande rasgos, se ha encontrado que no existe una diferencia sustancial en precisión para los modelos propuestos. Por otro lado, se ha encontrado que modelo basados en *Deep Learning* obtiene resultados ligeramente superiores a modelos mas clásicos, como la familia de modelos *KNN*.

Palabras claves: Sistemas de Recomendación, Basados en Filtro Colaborativos, Basados en Contenido, Modelos Híbridos, *GMF*, *KNN*, *NN-FM*, *DeepFM*, *mAP@k*.

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

Los sistemas de recomendación tienen como objetivo principal proporcionar a los usuarios productos, promociones y contenidos relevantes a sus preferencias o necesidades. Estos sistemas permiten a los usuarios encontrar de manera más fácil y eficiente lo que están buscando. Formalizando esta definición, podemos decir que los sistemas de recomendación buscan ayudar a un usuario o grupo de usuarios a descubrir ítems que se ajusten a sus preferencias, dado un conjunto de ítems que puede ser extenso o en un amplio espacio de búsqueda.

Este objetivo puede variar dependiendo de cada negocio: Si consideramos un *e-commerce* de *delivery* gastronómico, su propósito sería ofrecer a los clientes platos relevantes a un precio asequible y con un tiempo de entrega aceptable.

Si hablamos de un *e-commerce* de productos, su objetivo consiste en proporcionar a los usuarios aquellos productos que satisfacen sus necesidades, a un precio que estén dispuestos a pagar. Además, se busca garantizar una experiencia satisfactoria con los vendedores.

En el negocio de visualización de contenido (audio, video, texto, etc..), el objetivo es acercar a sus usuarios contenido a fin a sus preferencias para mejorar su experiencia en la plataforma.

El objetivo principal en todos los casos es mejorar la conversión. En el campo del *marketing*, se define la conversión como las acciones realizadas por los usuarios que están alineadas con los objetivos de la empresa. Por ejemplo, aumentar el volumen de compras en un *e-commerce* de productos, incrementar la cantidad de entregas mensuales en un *e-commerce* de *delivery* gastronómico, aumentar las impresiones de publicidad en aplicaciones de visualización de contenido, prolongar el tiempo de permanencia en plataformas de *streaming* de audio o video, entre otros. Existen numerosos ejemplos en los que el objetivo común es mejorar la conversión y el compromiso del usuario con la marca, es decir, el *engagement*.

Desde un enfoque técnico, los sistemas de recomendación se utilizan para predecir el grado de preferencia de un usuario con un artículo específico. Esto se logra aplicando algoritmos de optimización que minimizan la diferencia entre el grado de preferencia esperado y el grado de preferencia real del usuario. También existen otros enfoques que utilizan medidas de distancia para determinar este grado de preferencia. En secciones posteriores, exploraremos estos conceptos con mayor detalle.

1.1. Tipos de sistemas de recomendación

A continuación, en la figura 1.1, se pueden observar las diferentes categorías y subcategorías de los sistemas de recomendación:

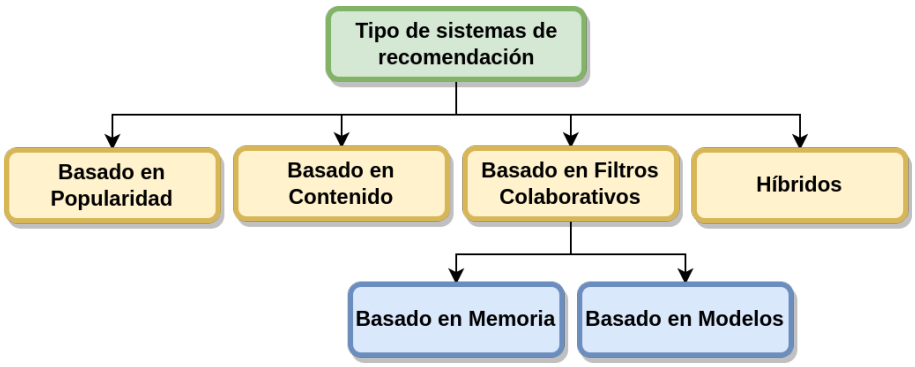

Fig. 1.1: Clasificación de tipos de sistemas de recomendaciones.

1.1.1. Basados en Popularidad

Este tipo de sistema de recomendación utiliza alguna característica de popularidad de los ítems en cuestión. Algunos ejemplos de estas características podría ser la cantidad de vistas, la cantidad de compras o la cantidad de comentarios positivos, o una combinación de ellas. Luego, estos sistemas buscan los K elementos más populares. Si bien este tipo de enfoque proporciona buenos resultados para nuevos usuarios, sus recomendaciones no tienen en cuenta las preferencias individuales de cada usuario, ya que se basan en estadísticas comunes a todos los usuarios. Por esta razón, a menudo no se consideran sistemas de recomendación en sentido estricto. No obstante, siguen siendo ampliamente utilizados debido a su capacidad para generar una alta tasa de conversión, a pesar de la falta de personalización.

1.1.2. Basados en Contenido

Este tipo de sistema de recomendación necesita un trabajo previo de ingeniería de *features* sobre los ítems. Se busca definir cuales son los *features* mas significativos para la tarea en cuestión, y cual es el grado de adecuación de cada ítems a los *features* seleccionados. Por otro lado, es necesario registrar las interacciones de los usuarios. Dadas estas interacciones, se puede definir el grado de preferencia de los usuarios a cada *feature* definido para los ítems. Con esta información, es posible encontrar tanto ítems como usuarios similares y realizar recomendaciones del tipo:

- Dado el *Usuario A*, el cual tiene preferencia por el *Ítem X*, también podría tener preferencia por el *Ítem Y*, por ser muy cercano o similar al *Ítem X*.
- Dos *Usuarios A y B* cercanos o similares, tendrán preferencias similares. De esta forma es posible recomendar ítem consumidos por el *Usuario A* al *Usuario B* y vise versa.

La principal desventaja de este enfoque, es la necesidad de realizar ingeniería de *features* para encontrar los *features* que produzcan recomendaciones relevantes al usuario. El modelo no encuentra estos *features* automáticamente, sino que deben ser definidos de antemano manualmente. Se puede apreciar que esto introduce un sesgo al momento de seleccionar los *features* o construirlos en base a datos referentes a los ítems. Como ventaja, si se encuentran los *features* correctos se pueden lograr muy buenos resultados.

1.1.3. Basados en Filtrado Colaborativos

Estos modelos, a diferencia de los modelos basados en contenido, no requirieren ingeniería de *features*, lo que hace muy simple su implementación, ya que únicamente es necesario registrar las interacciones de los usuarios para con los ítems. Luego, el propio modelo encuentra automáticamente los *features* mas relevantes dependiendo de la cantidad de columnas que se especifiquen (Dimensiones de un vector *Embedding*). Ejemplos de interacciones podrían ser:

- El *Usuario A* visualizo el *Ítem X* el dia 2 de marzo de 2022.
- El *Usuario A* compro el *Ítem X* el dia 10 de marzo de 2022.
- El *Usuario A* califico al *Ítem X* con 5 puntos el dia 25 de marzo de 2022.

Ambos tipo de modelos, basados en contenido y filtros colaborativos, personalizan sus recomendaciones. Es decir, ajustan las recomendaciones a las preferencias de cada usuario particular. Además, ambos permiten encontrar usuarios e ítems similares y recomendar ítems entre usuarios similares.

Por otro lado, los modelos basados en filtros colaborativos, descubren un espacio latente de soluciones sin necesidad de recolectar datos y definir *features* en forma manual, a diferencia de los modelos basados en contenido. La selección o construcción manual de *features* puede llevar a una solución sesgada, ya que no esta basada en datos sino en el juicio experto del científico de datos. Esto puede llevar a una selección subjetiva de los *features* que se aleje de la realidad, introduciendo un sesgo en la predicción.

No todo son rosas con estos modelos, dado que sufren un problema llamado *Cold start* o arranque en frio. Los usuarios nuevos son aquellos que aun no han realizado ninguna interacción con el sistema. Estos modelos no podrán realizar recomendaciones a estos usuarios, dado que requieren un mínimo de interacciones para comenzar a ofrecer recomendaciones con cierta precisión.

Además, existen otros problemas referidos al cambiar la cantidad de interacciones de los usuarios. Si pensamos en una solución donde alimentamos al modelo con una ventana de interacciones para los últimos N meses, tendremos las siguiente situaciones:

- Usuarios nuevos: Los usuarios nuevos no tendrán interacciones. Por lo tanto, este modelo no podrá realizar ninguna recomendación. En general, se establece un mínimo de interacciones para que el modelo pueda realizar recomendaciones de forma acertada.
- Usuarios con pocas interacciones: Por otro lado, tenemos a los usuarios que tienen una baja taza de interacciones con el sistema o aplicación. Por ejemplo, en un *e-commerce* de venta de productos, hay usuarios que compran con mucha frecuencia y otros muy de vez en cuando. Estos últimos, en general tendrán una baja taza de

interacción pudiendo caer por debajo del umbral mínimo que requiere el modelo. De esta forma, tendremos usuarios que quedarán fuera del modelo actual.

- Usuarios con muchas interacciones: En este caso, el usuario tiene una gran cantidad de interacciones con ítems. Para estos usuarios, el modelo podrá ofrecer recomendaciones relevantes, ya que cuanto mas interacciones se tenga, el modelo se ajusta con mas facilidad a sus preferencias. Por otro lado, esto puede ser una gran desventaja, ya que se produce un efecto de túnel. Es decir, el usuario obtiene recomendaciones muy ajustadas a sus preferencias, perdiendo la capacidad de descubrir nuevos ítems que podrían ser relevantes. Por esta cuestión se suelen mezclar tanto recomendaciones personalizadas como no-personalizadas, para favorecer el descubrimiento de nuevos ítems.

1.1.4. Modelos Híbridos

Son aquellos modelos que combinan mas de una técnica de recomendación, también llamados ensambles de modelos. Comúnmente están compuestos por modelos de recomendación por popularidad, basados en contenido y filtros colaborativos. De esta forma, cuando los usuarios caen por debajo del umbral de interacciones necesarias por el modelo de filtro colaborativos, se utiliza un modelos basado en contenido, popularidad, o algún otro modelo que no requiere de interacciones del usuario para realizar sus recomendaciones.

1.1.5. Categorías dentro de los modelos basados en filtros colaborativos

Dentro de los sistemas de recomendación basados en filtros colaborativos, tenemos dos sub-clasificaciones referidas a la forma en la que se realizan las predicciones.

Basados en Memoria

Este tipo de modelos, como su nombre lo indica, mantiene sus datos en memoria. Se recorren todos los datos (*full scan*) cada vez que se necesita realizar un inferencia o predicción (fijando un número de vecinos a comparar). Un ejemplo de estos modelos es el algoritmo de k vecinos cercanos (*KNN*), el cual mantiene una matriz rala de distancias en memoria, la cual se recorre completamente para comparar las distancias entre filas o columnas, usando alguna medida de distancia como puede ser la *distancia coseno*, *coseno ajustada*, *manhattan*, etc.. Para mitigar el problema de búsqueda exhaustiva (*full scan*), se puede utilizar una memoria *cache* y asi realizar estas búsquedas una única vez. Otro problema es su limitación al tamaño máximo de la memoria con la que se cuenta, es decir, que el tamaño de la matriz depende de la memoria máxima disponible. Esto puede mitigarse utilizando implementaciones de matrices rala, las cuales comprimen los datos en memoria guardando unicamente las celdas que tienen datos. Además, es posible utilizar un memoria *cache* que mantenga en memoria las búsqueda mas frecuentes y baje a almacenamiento secundario las menos frecuentes. Todos estos problemas de *performance* y uso de recursos se deben a que *KNN* no reduce la dimensionalidad de los datos, como si lo hacen varias implementaciones basadas en *embeddings*, *auto-encoder*, redes neuronales etc.., donde lo que se busca una representación mas compacta de los ítems y usuarios sin perder información. Mas allá de estos problemas, los resultados obtenidos por estos modelos no están muy alejados de aquellos que se encuentran en el estado del arte. Puede recomendarse su uso cuando tenemos un dominio reducido, dada su simplicidad.

Basados en Modelos

Algunos ejemplos de estos modelos son los clasificadores bayesianos, redes neuronales, algoritmos genéticos, sistemas difusos y la técnica de descomposición matricial (*SVD*). Estos modelos en general buscan directa o indirectamente reducir la dimensionalidad de los datos. De esta forma, es posible utilizarlos en dominios con una gran cantidad de datos.

1.2. Descripción del problema y motivación

Con este trabajo se busca contestar a las siguientes preguntas:

1.2.1. ¿Los modelos basado en filtro colaborativos que utilizan técnicos de *Deep Learning*, obtienen mejores resultados que aquellas que no las utilizan?

La idea detrás de esta pregunta es realizar *benchmarks* sobre distintos modelos del estado de arte basados en *Deep Learning* o no, utilizando el mismo set de datos y las mismas métricas. De esta forma, se busca comprender cual es la diferencia en *performance* entre los modelos seleccionados. Por otro lado, se busca comprender cuando es mas adecuado utilizar cada enfoque. Como ya se comentó en el apartado de introducción, hay modelos que están mas limitados que otros según el número de recursos de *hardware* o interacciones con los que se cuenta.

1.2.2. ¿Cuáles son las ventajas y desventajas de cada enfoque a la hora de aplicar estas técnicas?

Esta pregunta se refiere a comprender cuando es conveniente aplicar una técnica u otra teniendo en cuenta las ventajas y desventajas de cada enfoque y modelo.

1.2.3. ¿Cómo se puede solucionar el problema de *Cold start* que sufre el enfoque de recomendación basado en filtros colaborativos?

(Tesis)

Como ya se comentó en la introducción, los modelos de filtro colaborativos necesitan un número mínimo de interacciones usuario-ítem para poder operar y producir recomendaciones aceptables. La propuesta es explorar enfoques que permiten lidiar con este problema. Uno de los enfoques más comunes es utilizar ensambles de modelos basados en filtros colaborativos con otros modelo basados en contenidos o popularidad. Estos ensambles puede diferir en sus técnicas dependiendo del dominio de los datos.

1.3. Objetivos

Como primer objetivo, se pretender comprender cuales son los fundamentos teóricos sobre los que se apoya cada técnica aplicada y bajo que escenarios puede ser conveniente aplicarlas. Por otro lado, se intenta determinar cual es la diferencia en *performance* de cada técnica aplicada sobre el mismo set de datos, midiendo su *performance* utilizando las mismas métricas. ¿Obtenemos diferencias significativas?

Como segundo objetivo (Tesis), se busca proponer nuevas técnicas y/o explorar técnicas existentes que permite lidiar o solucionar el problema de *Cold start* que sufren los sistemas de recomendación basados en filtros colaborativos. Ademas, se compararan esta técnicas mediante un *benchmark* propuesto, para compara como se comporta cada modelos ante usuarios con escasas o ninguna interacción en el set de datos propuesto.

2. MATERIALES Y MÉTODOS

2.1. Datos

Para realizar este trabajo se selecciono el dominio del cine, ya que existen conjuntos de datos bien definidos y actualizados. Estos *datasets* en general están pensados 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, es necesario contar con datos de interacciones de usuarios con ítems (Películas en este caso). Además, dado que se busca solucionar el problema de *Cold start* para el enfoque de filtros colaborativos, se necesitará contar con otro enfoque de recomendación, el cual posiblemente pueda ser basado en contenido. Por esta cuestión, necesitamos contar con *features* completos y consistentes para los ítems (Películas).

Dadas estas necesidades, se decidió utilizar los *datasets* expuestos a continuación.

2.1.1. *MovieLens 25M Dataset*

Este *dataset* [1] prácticamente no tiene *features* para los ítems (Películas), pero si tiene la 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 baja *performance* por falta de estas. De esta forma, este *dataset* sera muy util para entrenar modelos de recomendación basados en filtros colaborativos. Además, cuenta con columnas extras como *tags*, que serán útiles a la hora de entrenar modelos basados en contenido.

Por último, este *dataset* 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 *dataset*:

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

Tab. 2.1: Columnas del *dataset Movie Lens*, relevantes para este trabajo.

2.1.2. *TMDB Movie Dataset*

Este *dataset* [2] no tiene calificaciones personalizadas para los ítems como sucede con el *dataset MovieLens* anterior, pero si tiene varios *features* referentes a películas. Estos *features* pueden ser muy útiles para modelos basados en contenido e inclusive modelos híbridos, los cuales se busca explorar en el trabajo de tesis. Contiene datos de 5,000 películas y sus columnas se especifican en la tabla 2.2 a continuación:

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

Tab. 2.2: Columnas del *dataset TMDB*, relevantes para este trabajo.

2.1.3. Pre-Procesamiento

Como parte inicial de la etapa de pre-procesamiento de datos, se utilizo una base de datos *MongoDB*. Se utilizo *MongoDB* y no *Pandas* dado que *Pandas* requiere cargar todo el *dataset* en memoria. Si bien se podria lidiar con este problema, aumentado el tamaño de la memoria *Swap* (Linux) o la memoria virtual(*Windows*), podria 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 *datasets* contienen varios archivos *csv*, los cuales vamos a llamar *tablas*. En la tabla 2.3 se especifican las columnas utilizadas:

File	Descripción
movie_metadata	Pertenece al *dataset TMDB*. Es la fuente de verdad de la cual se toman columnas referentes a *features* de una película.
tags	Pertenece al *dataset Movie Lens*. De esta tabla se tomaron los tags o palabras clave dadas de alta por los usuarios por cada película.
ratings	Pertenece al *dataset Movie Lens*. 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 *dataset* 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 únivoco de una película en ambas tablas.

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

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

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

Tabla de interacciones

La tabla *ratings_tags_v1* 2.5 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
user_id	Existen 13,281 usuarios.
movie_id	Existen 33,444 películas.
timestamp	Fecha en la cual el usuario califico el ítem(*movie_id*). Es un *string* de formato año-mes. Existen valores entre 1997-09 y 2019-11 inclusive.
rating	Calificación. Es un valor discreto numérico: *0.5*, *1*, *1.5*, *2*, *2.5*, *3*, *3.5*, *4*, *4.5* y *5*.
tags	Lista de palabras definidas por cada usuario para una película. Se cuenta con los *tags* a nivel usuario-película.

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

Tablas de metadata de películas

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

Columna	Descripción
title	Título de la película.
native_languaje	Lenguaje original en el cual fue filmada la película.
genres	Ambos *dataset* cuentan con una lista de géneros a los que adiare la película.
overview	Sinopsis de la película.
poster	Enlaces al detalle de la película en *imdb* y *TMDB*. Estos enlaces permiten hace join con mas datos que se encuentren en la descripción de estos sitios. En este trabajo solo se utilizara la imagen de la tapa de las película a modo de visualización.
release	Fecha de lanzamiento.
budget	Presupuesto destinado para la realización el film.
popularity	Popularidad.
vote_count	Cantidad de votos por película.
vote_mean	Cantidad media de votos por película.
tags	Son los *tags* cargados por todos los usuarios que interactuaron con la película. Es la mismas 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.

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
budget	70
poster	0.00085
release	0.0085
popularity	0.00085
vote_mean	4.8
vote_count	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 modifico, ya que no tenia valores faltantes en ninguna de sus columnas.

2.2. Análisis exploratorio

2.2.1. Variable *Rating*

En este análisis exploratorio analizaremos datos relevantes al problema de predicción de calificaciones de ítems por parte de los usuarios. Dentro de este análisis, la variable *ratings* o calificación es una de estas variables relevantes.

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:

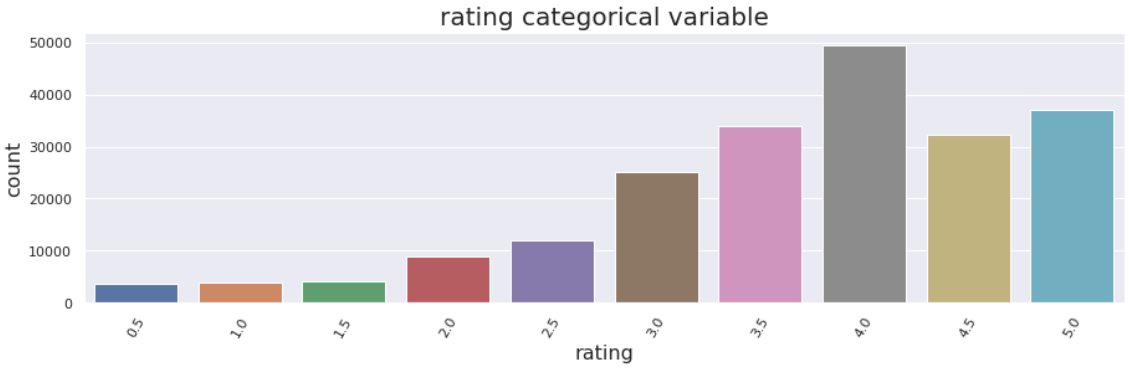

Fig. 2.1: Este diagrama de barras expone la frecuencia o cantidad de observaciones para cada valor discreto de calificación o *rating*.

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 mas 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:

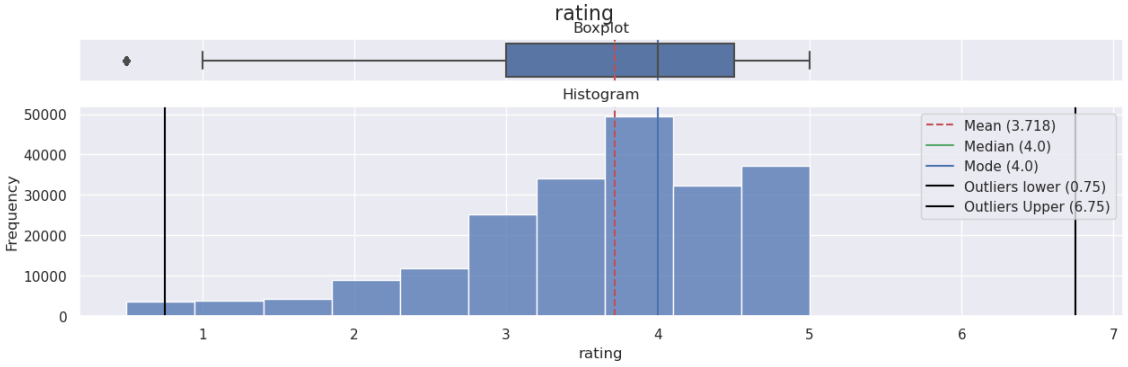

Fig. 2.2: Histograma y *Boxplot* de la variable *rating*. Los *ratings* son las calificaciones realizadas por los usuario para cada ítem o película.

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 $Q1$ y $Q3$ con 3,0 y 4,5 puntos (Rango inter-cuantil). La mediana (Cuantil $Q2$) 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 extreme 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 $Q1$ y $Q3$, donde tenemos el 50 % las calificaciones con major 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 $Q1$ y $Q2$ que entre $Q2$ y $Q3$. De esta forma, ambos intervalos conservan su 25 % de las observaciones pero hay menor dispersión entre $Q2$ y $Q3$.

A continuación segmentemos el anterior histograma por año:

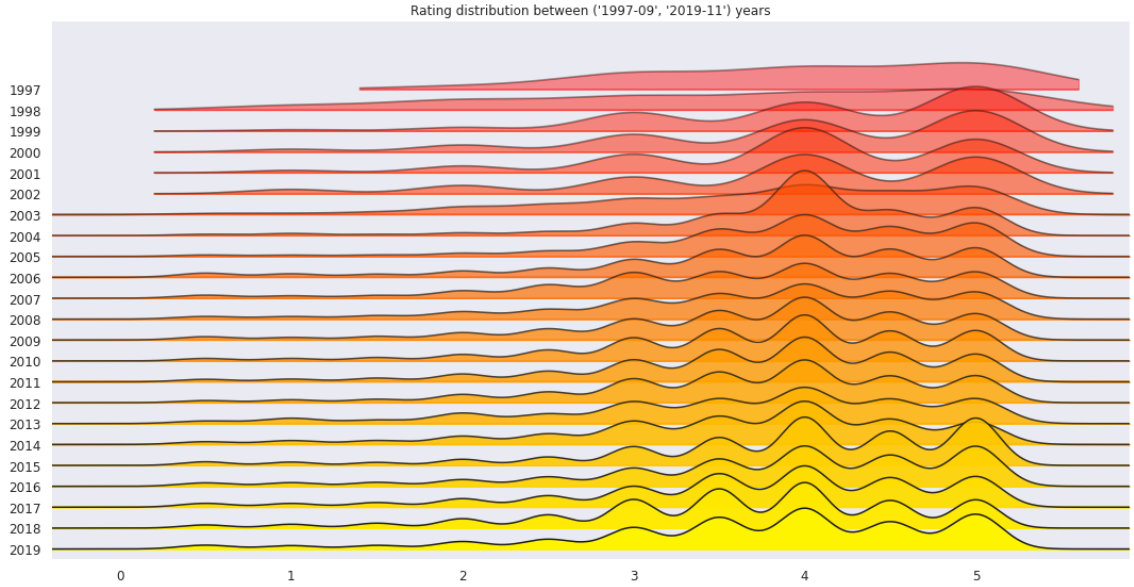

Fig. 2.3: Histograma de calificaciones segmentado por año. Como aclaración, en esta gráfica se describen los histogramas como funciones densidad, a pesar de que la variable *rating* es categóricas. De esta forma se puede apreciar con mas claridad la diferencia en cantidad de observaciones y el grado de dispersión de cada niveles 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 lo 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 dispersion en las puntuaciones.

2.2.2. 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:

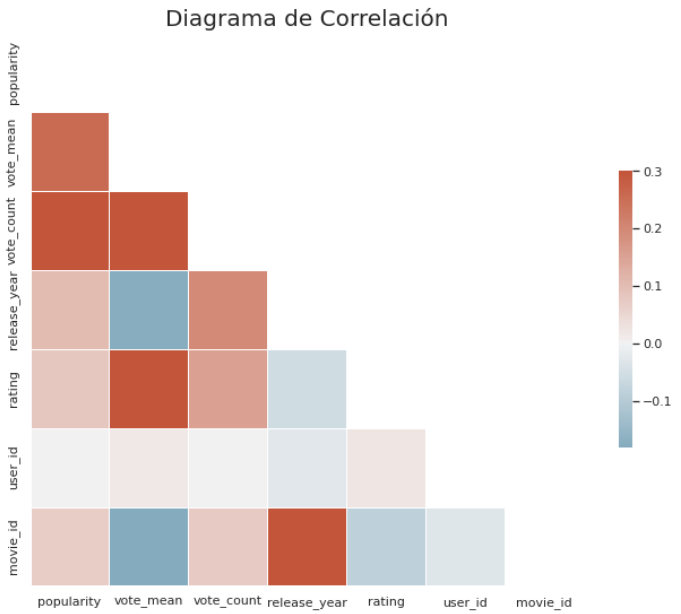

Fig. 2.4: Diagrama de correlación de *Person* aplicado a todas las variables numéricas resultado del *merge* entre las tablas *movies* e *interactions*.

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 correlacionadas 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* esten correlacionadas, 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 *relese_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 colinealidad 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 pelicula) 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* (id de usuario), *movie_id* y *rating*, es de interés analizar las variables correspondiente a features 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.

2.2.3. Variables de tipo texto

Las variables de tipo texto son muy útiles para genera *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 clase a una película. Un *tag* es una palabra o frase corta que identifica o describe una película. Es una forma simple de clusterizar, 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.

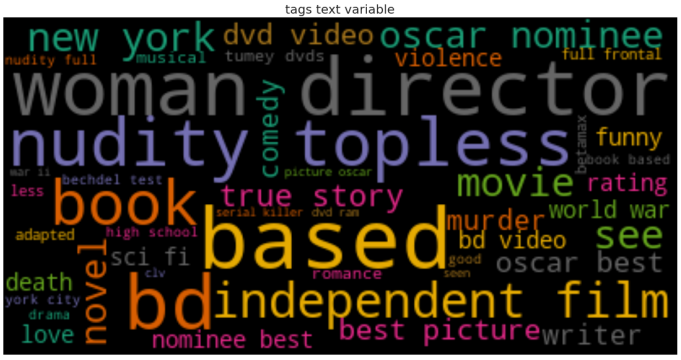

Fig. 2.5: Frecuencia de frases encontradas en la variable *Tags*. El tamaño de cada frase representan la cantidad de apariciones de la misma.

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.

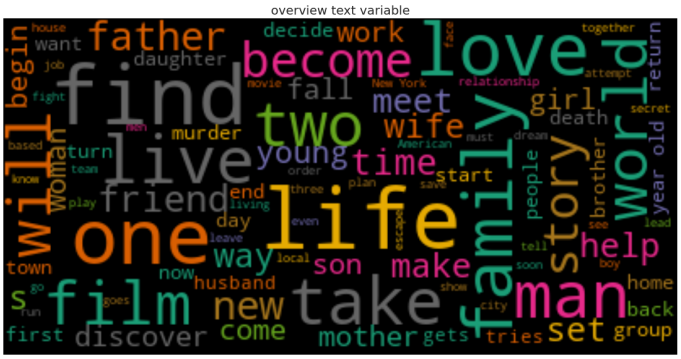

Fig. 2.6: Frecuencia de palabras encontradas en la variable *Overview*. El tamaño de cada palabra representan la cantidad de apariciones de la misma.

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 clústers(*Embeddings*) que pudieran generarse utilizando esta variable, podrían ser mas específicos que aquellos *clusters* generados a partir de la variable *tags*.

Title

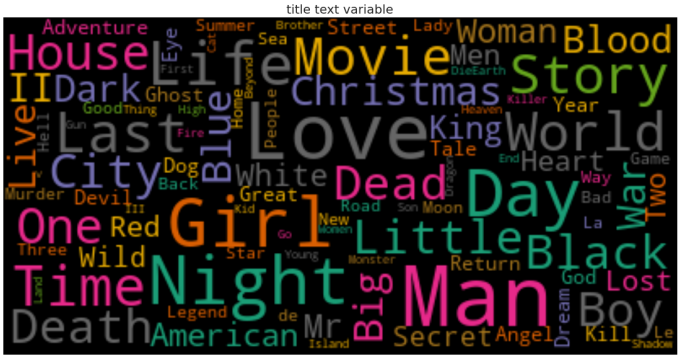

Fig. 2.7: Frecuencia de palabras encontradas en la variable *Title*. El tamaño de cada palabra representan la cantidad de apariciones de la misma.

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.

2.2.4. Análisis de Componentes Principales

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

Varianza Explicada

Las componentes principales son las variables resultado al aplicar el algoritmo *PCA* [3]. 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).

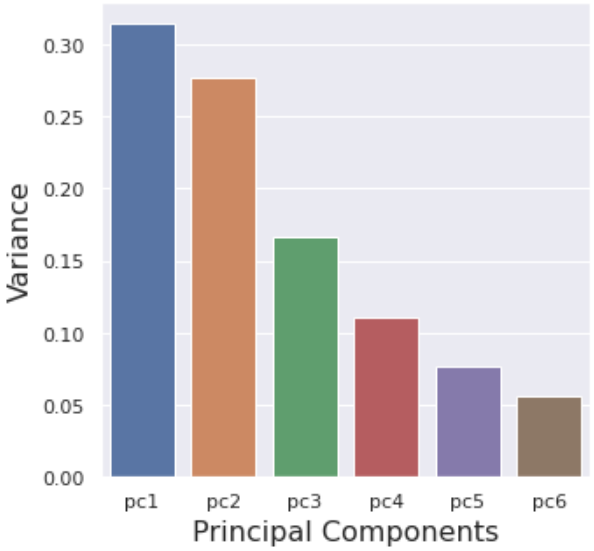

Fig. 2.8: Este diagrama de barras describe el grado de variabilidad o varianza explicada para cada componente principal resultado de aplicar el algoritmo*PCA* [3] sobre el conjunto de variables numericas originales.

Varianza

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

En la figura 2.8, inicialmente podemos apreciar que toda las componentes tiene 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 citerio 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 unicamente 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 otra dos ($PC2$ y $PC3$). También vemos que las variable *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

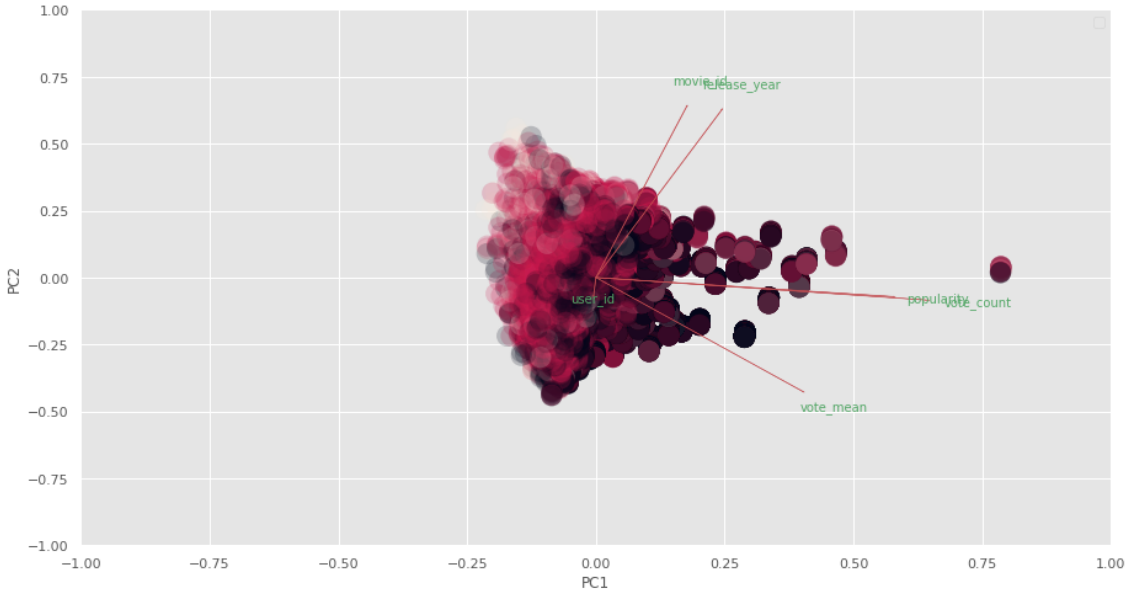

Fig. 2.9: Diagrama [4]*Biplot*. Este diagrama representa los valores de las variables originales coloreados en color rojo, negro y gris. Estos colores corresponden a tres segmentos de calificaciones: > 2, entre 2 y 3,5 y > 4. También se pueden apreciar los vectores correspondientes a las variables originales.

En la figura 2.9 se expone un diagrama *Biplot* [4]. 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 colinealidad. 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 media). 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 varaibles *Popularity* (Popularidad), *Vote Count* (Cantidad de votos) tienen casi aporte nulo a la componente $PC2$ en correspondencia con el analisis de cargas anteriormente expuesto. Si visualizamos los puntos que representan a las observaciones originales en el espacio latente generado por *PCA* [3], 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 lo puntos correspondientes a las observaciones en las direcciones de la componente $PC2$, vemos

que a major 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. MÉTODOS

En este capitulo se describirán los modelos utilizados para realiza la predicción de la clasificación de un usuario para una película que aun no ha visto. Para realizar esto, se utilizaron varios modelos basados en filtros colaborativos.

Cada implementación tiene sus particularidades: Su nivel de escalabilidad, sus tiempos de entrenamiento y predicción, su implementación, la exactitud de las predicciones, su tendencia al sobre ajuste, etc.. Para este trabajo se eligieron dos grandes grupos. Por un lado, una implementación sencilla basada en memoria, como es el algoritmo de K vecinos cercanos. Por otro lado, modelos basados en *Deep Learning*. Los modelos basados en *Deep Learning* utilizan *Embeddings* en todos los casos, como una forma de reducir la dimensionalidad de las variables categóricas que se utilizan como entradas al modelo. Además, cada modelo tiene su propia arquitectura, algunas clásicas y otras basadas en modelos del estado del arte. Luego, la idea fue medir los resultados de todo los modelos, utilizando distintas métricas comparables, entrenando con el mismo *dataset* en todos los casos. De esta forma podemos comparar los resultados de todos los modelos. Como *baseline* se tomo el modelo de K vecinos cercanos (*KNN*), dado que es el modelo mas simple. Este servira como punto partida para comparar resulados con otros modelos mas complejos.

Por otro lado, dada la cantidad de datos con la que se cuenta y teniendo en cuenta que estos modelos son muy demandantes en cuanto a recursos de *hardware*, se opto por usar el framework *PyTorch*. Dado que permite hacer uso tanto de *CPU* como *GPU*. De esta forma, se puede elegir cuando usar cada dispositivo y en que parte del flujo (pre-procesamiento, entrenamiento e inferencia). Ya elegido el *framework*, se opto por implementar todos los modelos desde sus bases, ya que *PyTorch* no cuenta con mucho modelos del estado del arte desarrollados de forma oficial. De esta forma se implemento cada modelo desde cero para poder hacer uso de *CPU* y *GPU* de forma granular y realizar un uso mas eficiente de los recursos disponible. Mas adelante veremos que esto puede inpactar fuertemente en los tiempos de entrenamiento e inferencia.

3.1. Enfoque Basados en Memoria

3.1.1. Algoritmo de los K vencinos cercanos (*K-Nearest-Neighbor* o *KNN*)

Esta es la implementación clásica y mas intuitiva para realizar recomendación de ítems. Una vez entrenado el modelo, se cuenta con una matriz de distancias que pueden ser: distancias entre usuarios o ítems, y otra matriz de calificaciones usuario-ítem. De esta forma, en la etapa de inferencia, el modelo toma como entrada un usuario (*user_id*) y un ítem (*item_id*) y retorna la predicción de la calificación. Estas matrices se puede mantener en memoria, persistir en una base de datos (como puede ser *Redis*) o en un archivos indexado. Por esta cuestión, la categoría en memoria no tiene por que ser estricta, pero si se entiende que los mejores tiempos de inferencia y entrenamiento se lograran cuando se tenga parte o la totalidad de estas matrices en memoria.

Luego, para realizar el entrenamiento del modelo se necesita una lista de tuplas, donde cada tupla contiene:

Lista de tuplas

$$Tuplas = [< u_1; i_1; r_{u_1,i_1} >, ..., < u_n; i_m; r_{u_n,i_m} >] \tag{3.1}$$

Donde:

- u es un identificador secuencial, univoco y numérico de un usuario. Estos identificadores se generan a partir de una secuencia numérica, es decir que no debemos tener huecos para minimizar el uso de memoria en caso de no usar matrices ralas.
- i es un identificador secuencial, univoco y numérico de un ítem. En este caso los ítems son películas, pero podrían ser cualquier entidad identificable como productos, usuarios, comidas, etc..
- $r_{u,i}$ es la calificación otorgada al ítem i por parte del usuario u.
- n es la cantidad total de usuarios en el *dataset* de entrenamiento.
- m es la cantidad total de ítems en el *dataset* de entrenamiento.

Dada esta lista de tuplas, podemos construir una matriz esparza donde cada fila representa a un usuario y cada columna a un ítem o vise versa, y las celdas o valores de la misma contienen las calificaciones.

Matriz de calificaciones

$$Calificaciones_{u,i} = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,i} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,i} \\ \vdots & \vdots & \ddots & \vdots \\ r_{u,1} & r_{u,2} & \cdots & r_{u,i} \end{pmatrix} \tag{3.2}$$

Donde:

- $r_{u,i}$ es la calificación otorgada al ítem i por parte del usuario u.
- Cada vector fila F_u contiene todas la calificaciones realizadas por el usuario u para todos los ítems. Los ítems que aun no tiene calificación contiene el valor 0.
- Cada vector columna C_i contiene las calificaciones realizadas por todos los usuarios para el ítem i. Las posiciones correspondientes a los usuarios que aun no calificaron el ítem i tendrán el valor 0.

En el siguiente paso, se debe construir la matriz $Distancias_{u_a,u_b}$ que contiene las distancias entre todos los vectores fila F_u de la matriz de $Calificaciones_{u,i}$. Cabe aclarar que cada vector fila F_u de la matriz de $Calificaciones_{u,i}$ representa a un usuario, ya que contiene todas las calificaciones realizadas por el mismo.

Matriz de distancias

$$Distancias_{u_a,u_b} = \begin{pmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,u_b} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,u_b} \\ \vdots & \vdots & \ddots & \vdots \\ d_{u_a,1} & d_{u_a,2} & \cdots & d_{u_a,u_b} \end{pmatrix} \tag{3.3}$$

Donde:

- d_{u_a,u_b} es la distancia entre el vector fila Fu_a y Fu_b de la matriz de $Calificaciones_{u,i}$.

En cuanto a las distancias, no hay una restricción acerca de cual utilizar. En general las distancias que mejor ajustan apra este dominio son las siguientes:

- Distancia Coseno Ajustado.
- Distancia Coseno.
- Distancia de *Pearson* (1 - Correlación de *Pearson*).

Luego, para este trabajo se eligió utilizar la distancia coseno.

Distancia Coseno

La distancia coseno es una medida de similitud entre dos vectores en un espacio vectorial que posee un producto interno. La distancia coseno entre dos vectores se mide en grados. De esta forma, cuanto menor es el ángulo entre dos vectores mas similares son entre sí. De forma contraria, cuando mayor es el ángulo entre dos vectores menos similares seran. A contunaicon se define la distancia coseno como:

$$Distancia\,Coseno_{ua,ub} = \frac{\sum_{i \in I} r_{ua,i}.r_{ub,i}}{\sqrt{\sum_{i \in I} r_{ua,i}^2}.\sqrt{\sum_{i \in I} r_{ub,i}^2}}, ua \neq ub \tag{3.4}$$

Donde:

- ua y ub son los indices de dos vectores fila F_u de la matriz de $Calificaciones_{u,i}$. Cada uno de estos vectores fila F_u representan a un usuario.
- $ua \neq ub$, es decir que cada índice representa a un usuario distinto.
- I es la cantidad total de columnas de la matriz de $Calificaciones_{u,i}$.
- i el índice de una columna de la matriz de $Calificaciones_{u,i}$. Cada columna representa a un ítem y contiene todas las calificaciones realizadas por todos los usuarios sobre el ítem i.
- $0 <= Distancia\,Coseno_{ua,ub} <= 1$. Cuanto menor sea el valor de $Distancia\,Coseno_{u_a,u_b}$ mas similares serán los usuarios con los indices u_a y u_b.

Similitud Coseno

$$Similitud\,Coseno_{ua,ub} = 1 - Distancia\,Coseno_{ua,ub} \tag{3.5}$$

Donde:

- $0 <= Similitud\,Coseno_{ua,ub} <= 1$. Cuanto mayor sea el valor de $Similitud\,Coseno_{ua,ub}$ mas similares serán los usuarios con los indices u_a y u_b.

Volviendo a nuestro algoritmo, la idea es calcular la distancia de cada vector fila F_u de la matriz de $Calificaciones_{u,i}$ contra todos los demás vectores fila de la misma matriz, obteniendo asi la matriz de $Distancias_{u_a,u_b}$, donde cada fila y columnas representa a los vectores fila F_u de la matriz de $Calificaciones_{u,i}$.

Aquí es donde finaliza la etapa de entrenamiento. Luego la inferencia o predicción depende de la implementación que se elige para predecir las calificaciones. En todos los casos se utilizan ambas matrices para realizar las predicciones. A continuación una explicación del paso de inferencia o predicción para cada implementación elegida.

3.1.2. Algoritmo de los K vencinos cercanos basado en usuarios (*KNN User Based*)

En el apartado anterior se explico cómo calcular las matrices de $Calificaciones_{u,i}$ y $Distancias_{u_a,u_b}$. El cálculo de estas matrices es parte del proceso de entrenamiento del modelo *KNN*. En este apartado se explicará el proceso para realizar la predicción o inferencias de la calificación de un ítem por parte de un usuario [5] utilizando la variante basada en usaurios.

Entonces, el enfoque de K vecinos cercanos basado en usuarios para predecir la calificación de un ítem, se define a continuación:

$$Prediccion\,basada\,en\,usuarios\,_{u,i} = \overline{r}_u + \frac{\sum_{o \in O}(r_{o,i} - \overline{r}_o).w_{u,o}}{\sum_{o \in K} w_{u,o}}, u \neq o \tag{3.6}$$

Donde:

- $Prediccion\,basada\,en\,usuarios\,_{u,i}$ es la predicción de la calificación del usuario u para el ítem i.
- o (Minúscula) pertenece al conjunto O (Mayúscula) de usuarios. O es el conjunto de todos los usuarios menos el usuario u.
- $w_{u,o}$ es la similitud entre los usuarios u y o. Se calcula mediante $Similitud\,Coseno_{u,o}$
- $u \neq o$, es decir que cada índice representa a un usuario distinto.
- $\overline{r}_u$ es el promedio de todas las calificaciones realizadas por el usuario u. Se pueda calcular como $\overline{r}_u = \frac{1}{N}\sum_{i=1}^{N} Calificaciones_{u,i}$, siendo N el la cantidad de total de ítems.
- $r_{o,i} - \overline{r}_o$ es la diferencia entre la calificación del usuario o para el ítem i y el promedio de calificaciones del usuario o. Esta diferencia se utiliza para ajustar el sesgo de calificación de cada usuario. Este sesgo se da debido a la subjetividad que tiene cada usuario al momento de calificar un ítem. Algunos usuarios tienden a calificar todo de forma optimista, otorgando calificaciones mas bien altas; otros usuarios son mas pesimistas y tienden a otorgar calificaciones bajas. Al restar por la media de calificaciones de cada usuario o, estamos normalizando las calificaciones, haciéndolas mas o menos comparables. Aplicando esta transformación, las calificaciones se definen en terminos de cuanto se alejan de la media de calificaciones del usuario o. Por supuesto que cada usuario tendra una media sesgada por su comportamiento al calificar. Mas allá de esto, sigue siendo una forma de disminuir este fenómeno de subjetividad al momento de calificar un ítem.

Finalmente, el calculo de la predicción no es mas que el promedio de calificaciones del usuario u sumado al promedio pesado de las calificación de los demás usuarios para el ítem i, donde los pesos son las distancias del usuario u con los demás usuarios.

Ahora, por un tema de performance, el conjunto O no contiene a todos los demás usuarios, sino un conjunto de tamaño K el cual contiene a los usuario mas cercanos en términos de distancia. Es decir que, como paso previo a la predicción, es necesario encontrar a los K usuarios mas cercanos al usuario u. De esta forma, el parámetro K se convierte en un hiper-parámetro del modelo. Luego a mayor K, mayor sera número de vecinos a tener en cuenta para calcular la predicción, y mayor será el tiempo de inferencia del modelo.

Por otro lado, a mayor K estaremos incluyendo mas vecinos que son menos similares en términos de distancia. Debido a esto, siempre se busca encontrar el mejor valor posible para K. Este valor se buscado a través de una optimización de hiper parámetros regida por una métricas que valide la exactitud del modelo al momento de la predicción, obteniendo como resultado el K para el cual el modelo tiene el resultado mas exactos posibles.

3.1.3. Algoritmo de los K vencinos cercanos basado en ítems (*KNN Item Based*)

Este modelo es muy similar al anterior, la diferencia radica en que la matriz de $Calificaciones_{i,u}$ tiene ítems como filas en vez de usuarios, y usuarios como columnas en vez de ítems. Es decir, es la matriz transpuesta de la matriz de $Calificaciones_{u,i}$ original [5]. De esta forma, la matriz de $Distancias_{i_a,i_b}$ mide las distancias entre vectores fila F_i los cuales representan ítems. Por otro lado, también existen diferencias con el modelo anterior, al momento de calcular las predicciones. A continuación la definición del cálculo de las predicciones:

$$Prediccion\,basada\,en\,items\,_{u,i} = \frac{\sum_{o \in O} r_{u,o}.w_{i,o}}{\sum_{o \in O} w_{i,o}}, i \neq o \tag{3.7}$$

Donde:

- $Prediccion\,basada\,en\,items\,_{u,i}$ es la predicción de la calificación del usuario u para el ítem i.
- O (Mayúscula) es el conjunto de los K ítems mas cercanos o similares al ítem i. El ítem o (Minúscula) pertenece al conjunto O de todos los ítems menos el ítem i.
- $i \neq o$: Son los indices de los ítems i y o, representando a ítems distintos.
- $w_{i,o}$ es la similitud entre los ítems i y o.

Finalmente, la predicción es un promedio pesado de las calificaciones del usuario u para los ítems cercanos o similares al ítem i, pesadas por la el grado de similitud de cada ítem o con el ítem i.

3.1.4. Modelo ensamble de los algoritmos de los K vecinos cercanos basados en usuarios e ítems (*KNN User-Item Based Ensemble*)

Dado que contamos dos modelos basados en *KNN* se realizo un samble de ambos modelos, el cual realiza un promedio de las salidas de ambos modelos.

3.2. Enfoque basado en modelos

Hasta aquí realizamos una descripción de los modelos *KNN* utilizados en este trabajo y sus distintas implementaciones. Estos modelos tiene varias falencias. Entre las mas importantes encontramos el problema de escala, ya que el tamaño de los datos a procesar depende casi linealmente de los recursos de memoria, *CPU* y/o *GPU* disponibles. De esta forma, cuando es necesario procesar una gran cantidad de datos para realizar predicciones, se opta por modelos que realicen algún tipo de reducción de dimensionalidad para construir su representación internal, la cual luego se utilizada para realizar las predicciones. A

esta presentación interna muchas veces se la llama modelo, ya que el modelo en si no es el algoritmo utilizado si no el estado internal al que se llega luego del entrenamiento.

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

Particularmente en el ámbito de recomendaciones, se cuenta con variables categóricas de alta dimensionalidad. Para este trabajo, tenemos dos variable 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 variable categóricas en una representación numérica. El enfoque mas 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 numero 1, 0 y 0 respectivamente. Ahora pensemos en la cantidad de usuarios que tiene *Google* o *Amazon*, ¿Que tamaño tendría el vector que representa a un solo usuario?¿Por qué usar un vector 99 % ralo para representar un valor?¿No hay una forma mas compacta de realizar esta codificación?

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

Un *Embedding* no es mas 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 hiper-parámetro 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 gráficas 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 las 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 entrenan *Embeddings* captan esta información de forma automática en base a las observaciones usadas en el entrenamiento, generando estos espacios latentes llamados *Embeddings*.

3.2.2. 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 hiper-parámetro mas a optimizar. Por otro lado, debemos definir la cantidad de vectores *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 categorica Estado del Tiempo, deberiamos 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 usa 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 *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 entranamiento, el modelo irá ajustando los valores de cada vector *embedding*.

En la etapa de *froward pass*, se pasa como entrada un valor de la variable categórica codificado como númerico. 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.

Finamente, 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.

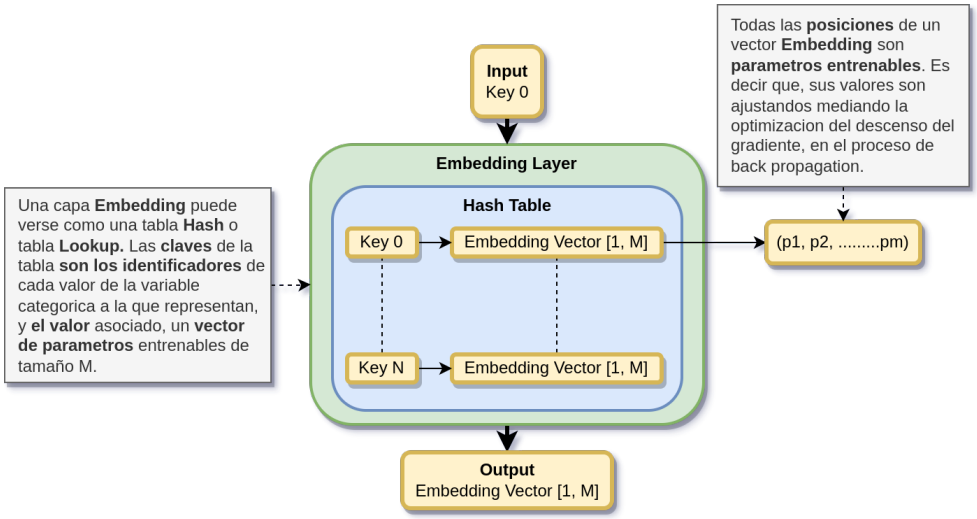

Fig. 3.1: Esquema de una capa o módulo *Embedding*.

3.2.3. Arquitecturas Utilizadas

En la sección anterior se explicó uno de los componentes básicos y más utilizados en los modelos de recomendación basados en *Deep Learning*. A partir de este punto, se procederá a describir las arquitecturas utilizadas en este trabajo.

3.2.4. Factorización Matricial General (*General Matrix Factorization o GMF*)

Esta es una arquitectura clásica en sistemas de recomendación basados en filtros colaborativos. El algoritmo de factorización de matrices [6] funciona desacoplando la matriz de interacciones usuario-ítem en un producto escalar de dos matrices regulares de baja dimensionalidad. Este algoritmo o familia de algoritmos fue popularizado por primera por *Simon Funk* en la competencia *Netflix prize* [7] en 2006. La idea principal del algoritmo es representar a usuarios e ítems en un espacio latente de baja dimensionalidad. A partir del trabajo inicial realizado por *Funk* en 2006, se han propuesto multiples enfoques de factorización de matrices para sistemas de recomendación, siento este el modelo mas simple y efectivo.

Este modelo se puede construir fácilmente realizando el producto escalar de dos matrices de vectores *Embedding*, las cuales tiene una baja dimensionalidad debido al principio de funcionamiento de los vectores *Embedding*. A continuación se puede ver un esquema del modelo, el cual toma como entradas los identificadores de un usuario e ítem, luego resuelven los vectores *Embedding* correspondientes a ambos identificadores, y finalmente se realiza el producto escalar de ambos vectores. Este producto escalar tiene como resultado la calificación del usuario para el ítem dado. Por otro lado, el algoritmo del optimización de gradiente descendente sera quien se ocupe de ajustar los pesos de ambas matrices para que, dados los identificadores de un usuario e ítem, se obtenga la calificación correspondiente a la observación utilizada como ejemplo de entrenamiento.

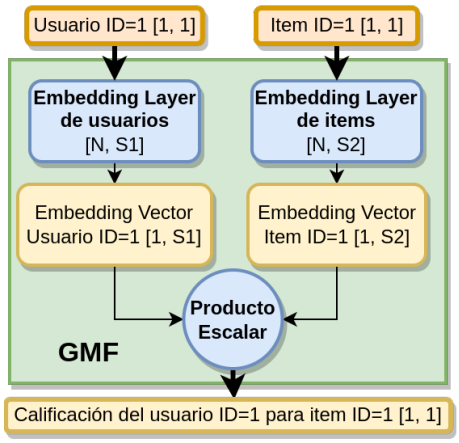

Fig. 3.2: Esquema de un modelo *General Matrix Factorization (GMF)*.

En términos matemáticos, este modelo realiza la siguiente operación en cada paso hacia adelante (*forward-pass*):

$$\tilde{r}_{u,i} = V_u.V_i^T \tag{3.8}$$

Donde:

- V_u es el vector *embedding* correspondiente al usuario u.
- V_i^T el vector *embedding* correspondiente al ítem i.
- $\tilde{r}_{u,i}$ es la predicción de la calificación realizada por el usuario u al ítem i (valor escalar).

En términos matriciales podemos verlo de la siguiente manera:

$$\tilde{R} = U.I \tag{3.9}$$

Donde:

- $U \in \mathbb{R}^{u \times f}$ es la matriz de vectores *Embedding* de usuarios, la cuál tiene tantas filas u como usuarios. f es la dimensión del número de columnas (también llamada factor latente) corresponde al tamaño seleccionado para los vectores *Embedding* (Como ya se aclaro anteriormente, este es un hiper-parametro a ajustar).
- $I \in \mathbb{R}^{f \times items}$ es la matriz de vectores *Embedding* de ítems, la cual tiene tantas filas como factores latentes(posiciones) en los vectores *Embedding*, y tantas columnas como ítems se tenga.
- $\tilde{R} \in \mathbb{R}^{usuarios \times items}$ es la matriz de calificaciones, donde cada fila corresponde a un usuario y cada columna a un ítem.

El tamaño de la dimensión de factores latentes, como ya se vio anteriormente en el apartado *One-Hot vs. Embeddings*, es un hiper-parámetro mas a ajustar. Se ha demostrado [8] que realizar factorización de matrices con un factor latente de tamaño 1 es equivalente a un modelo de recomendación por popularidad, es decir que se recomiendan los ítems mas populares sin tener en cuenta la personalización de las recomendaciones. Luego, a medida que vamos incrementando el tamaño del factor latente, estas recomendaciones serán cada vez más personalizadas, aumentando la calidad de las mismas. Cuando el tamaño del factor latente es muy grande, el modelos comienza a sobre ajustar (*overfitting*) y por ende la calidad de las recomendaciones comenzara a empeorar. Para solucionar este problema, se suelen agregar términos de regularización en la función de error a minimizar:

$$\arg\min_{H,W} \|R - \tilde{R}\|_{\mathrm{F}} + \alpha\|H\| + \beta\|W\| \tag{3.10}$$

Donde:

- $\|.\|_{\mathrm{F}}$ se define como [[norma matricial]]
- $\|H\|$ y $\|W\|$ pueden ser normas matriciales u otro tipo de norma dependiendo del sistema de recomendación.

3.2.5. Factorización Matricial General con Sesgo (*Biased General Matrix Factorization o B-GFM*)

El modelo *GMF* de *Simon Funk* [6, 9], visto en el apartado anterior, realiza recomendaciones de muy buena calidad, pero tiene una limitación: sólo utilizar interacciones usuario-ítem que tengan que ver con valores numéricos referidos a interacciones explicitas, como calificaciones. Los sistemas de recomendación modernos deben explotar todas las interacciones posibles, tanto explícitas (calificaciones numéricas) como implícitas (compras, vistas, favoritos, etc..). Para solucionar este nuevo problema, donde es necesario usar cualquier tipo de interacción usuario-ítem (explicita o implícita), se agrega un *bias* o sesgo para los usuarios, y otro para los ítems.

A Continuación se puede aprecia el diagrama del modelo, muy similar al diagrama 3.2:

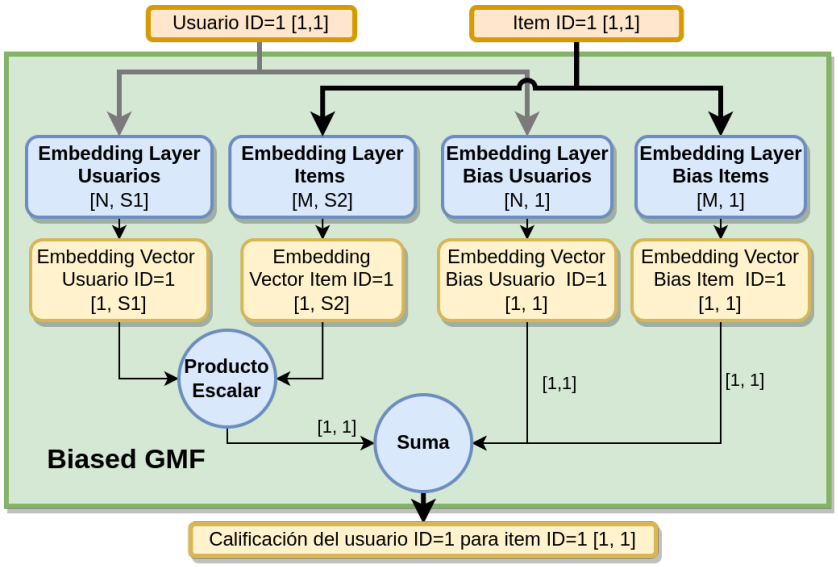

Fig. 3.3: Esquema de un modelo *Biased General Matrix Factorization (B-GMF)*. A diferencia del modelo *GMF*, este suma a la salida un *bias* o sesgo por cada variable de entrada.

En este caso se agregan dos nuevas capas *Embedding*, las cuales representa a los sesgos de usuarios e ítems respectivamente. El tamaño de los factores latentes o vectores *Embedding* correspondiente a cada *bias* o sesgo es 1, dado que son valores escalares. Finalmente, luego de calcular el producto escalar, se suman los factores latentes resultado de ambas capas *Embedding* correspondiente a los *bias* o sesgos.

3.2.6. Factorización Matricial mediante Redes Neuronales (*Neural Network Matrix Factorization o NN-FM*)

En los enfoques anteriormente vistos (*GFM* y *Biased GFM*), dadas dos matrices de baja dimensionalidad se realiza un producto escalar y se suman sesgos, dependiendo del caso, para calcular o inferir la calificación de un usuario para un ítem dado. Estos modelos, como ya se explico anteriormente, aprenden los pesos o parámetros de los vectores *Embedding* en el proceso se entrenamiento.

El enfoque de *NN-MF* [10] [11] es levemente distinto. En este caso se reemplaza el producto interno, el cual podemos pensarlo como conocimiento a priori del problema, por otra función desconocida, que sera la que aprenderá el modelo a partir de las observaciones suministradas en el entrenamiento. En particular, se remplaza el producto escalar sumado a los sesgos, por una red neuronal multi capa de capas densas o *fully connected*. De esta forma, el modelo no solo aprender los parámetros de los vectores *embedding*, sino también los pesos de la red multi capa. En definitiva el modelo aprende cual es la mejor función para predecir las calificaciones del usuario.

El uso de redes neuronales presenta una ventaja significativa: la posibilidad de utilizar múltiples variables como entrada, no limitándose únicamente a las variables categóricas usuario e ítem. Es precisamente en esta característica donde radica su mayor potencial. No obstante, es importante mencionar que este modelo muestra una ligera disminución en la precisión de sus predicciones en comparación con modelos anteriores.

A continuación podemos ver un esquema del modelo, muy similar a *GFM* como ya se dijo, con la diferencia que tenemos una red multi capa en vez de un producto escalar.

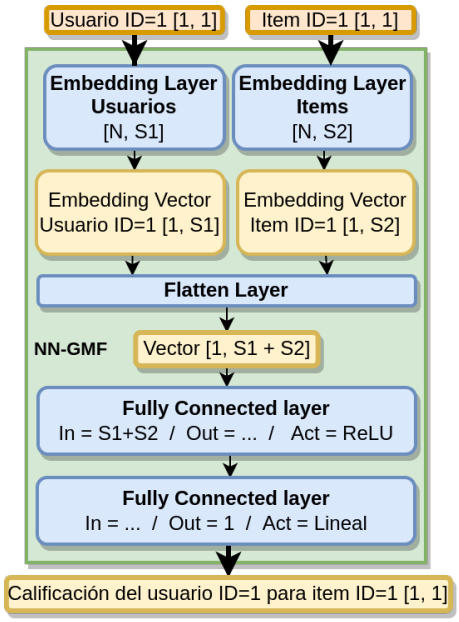

Fig. 3.4: Esquema de un modelo *Neural Network Matrix Factorization (NN-MF)*.

Para comenzar, el modelo tiene como entradas independientes los identificadores de usuarios e ítems. Cada capa *Embedding* retorna el correspondiente vector *embedding* asociado a estos identificadores. A continuación, el bloque *Flatter* toma ambos vectores y produce uno nuevo mediante su concatenación. Este vector resultante se convierte en la

entrada de una red multi-capa.

Es importante destacar que la red multi-capa tendrá tantas entradas como dimensiones posea este nuevo vector combinado. La cantidad de capas y el número de neuronas por capa son hiper-parámetros que se ajustarán durante el proceso de optimización. Por esta razón, no se especifica un número fijo de capas o neuronas por capa.

Cada capa, excepto la última, utiliza la función de activación *ReLU*, mientras que la capa final emplea una activación *Lineal*, similar a una regresión lineal. Esto se debe a que el objetivo es predecir las calificaciones, las cuales tienen un rango de valores reales entre 0,5 y 5. Es importante señalar que se está considerando utilizar una activación *Softmax* en lugar de una activación *Lineal* en la última capa, lo que permitiría abordar el problema como uno de clasificación.

3.2.7. Máquinas de Factorización (*FM*)

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

Las máquina de factorización propuestas por *Steffen Rendle* en 2010 [14], 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 continacion, se define el modelo formalmente:

- $x \in \mathbb{R}^d$ es un vector de *features* donde cada una de sus componentes representa a una variable del *dataset*, 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 *dataset* seleccionado, seria la calificación del usuario.

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

$$\hat{y}(x) = \mathbf{w}_0 + \sum_{i=1}^{d} \mathbf{w}_i x_i + \sum_{i=1}^{d} \sum_{j=i+1}^{d} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \tag{3.11}$$

Donde:

- d es la cantidad de *features* 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 disminuí 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, llevandolo 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:

$$
\begin{aligned}
&= \sum_{i=1}^{d} \sum_{j=i+1}^{d} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \\
&= \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{d} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j - \frac{1}{2} \sum_{i=1}^{d} \langle \mathbf{v}_i, \mathbf{v}_i \rangle x_i x_i \\
&= \frac{1}{2} \big(\sum_{i=1}^{d} \sum_{j=1}^{d} \sum_{l=1}^{k} \mathbf{v}_{i,l} \mathbf{v}_{j,l} x_i x_j - \sum_{i=1}^{d} \sum_{l=1}^{k} \mathbf{v}_{i,l} \mathbf{v}_{i,l} x_i x_i \big) \\
&= \frac{1}{2} \sum_{l=1}^{k} \big((\sum_{i=1}^{d} \mathbf{v}_{i,l} x_i)(\sum_{j=1}^{d} \mathbf{v}_{j,l} x_j) - \sum_{i=1}^{d} \mathbf{v}_{i,l}^2 x_i^2 \big) \\
&= \frac{1}{2} \sum_{l=1}^{k} \big((\sum_{i=1}^{d} \mathbf{v}_{i,l} x_i)^2 - \sum_{i=1}^{d} \mathbf{v}_{i,l}^2 x_i^2 \big)
\end{aligned}
\tag{3.12}
$$

Con esta re-formulación del último termino, 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:

$$
\hat{y}(x) = \mathbf{w}_0 + \sum_{i=1}^{d} \mathbf{w}_i x_i + \frac{1}{2} \sum_{l=1}^{k} \big((\sum_{i=1}^{d} \mathbf{v}_{i,l} x_i)^2 - \sum_{i=1}^{d} \mathbf{v}_{i,l}^2 x_i^2 \big) \tag{3.13}
$$

3.2.8. 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 explicitas. 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 *features* para disminuir estos sesgos.

El modelo de máquinas de factorización profundas (*DeepFM*) [15, 16] 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* [17] 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 multicapa 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:

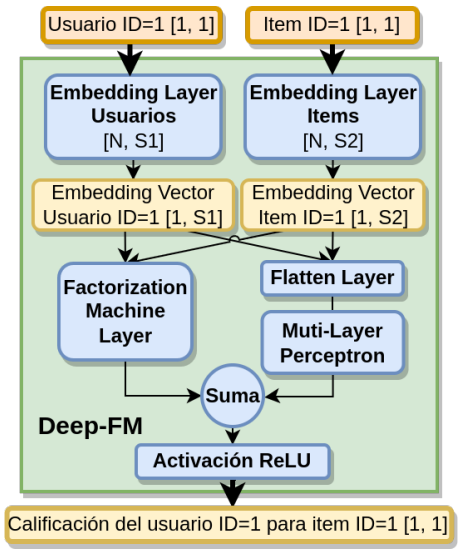

Fig. 3.5: Esquema de un modelo *Deep Factorization Machine (DeepFM)* o maquina de factorización basada en *Deep Learning*.

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.

3.3. Métricas

Para medir y comparar el grado de exactitude de los modelos seleccionados, tanto en el conjunto de validación como entrenamiento, se han seleccionado dos métricas:

- *Root Mean Square Error (RMSE)*: Es la raíz cuadrada del error cuadrático medio.
- *Mean Average Precision at k (mAP@k)*: Es la media del promedio de la precision para un tamaño K de observaciones.

3.3.1. *Root Mean Square Error (RMSE)*

Dado que todos los modelos que se evaluaron en este trabajo tiene como salida una variable real (Calificación de los usuarios para un ítem), es posible utilizar *RMSE*. Esta métrica es utilizada en problemas de regresión donde la salida del modelo es una variable numérica real.

Si bien esta métrica no es la métrica por excelencia a usar en el ámbito de sistemas de recomendación, ayuda a comprender cuales el grado de ajuste de los modelos y puede servir como una métrica complementaria al momento de evaluar los mismos.

Definición:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{3.14}$$

Donde:

- y_i es el true value o verdad de campo de la observación.
- $\hat{y}_i$ es la predicción realizada por el modelo predictor.
- N es el numero de observaciones sobre las que se realizo la predicción del modelo.

3.3.2. *Mean Average Precision at k (mAP@k)*

Mean Average Precision at k (mAP@k) o media del promedió de la precision para K observaciones, es una de las métricas mas usada para evaluar sistemas de recomendación [18, 19, 20].

Antes de comenzar, definamos a los sistemas de recomendación en terminos de sus entradas y salida, donde:

- Entradas: Como entradas tenemos el identificador del usuario al cual queremos presentarle recomendaciones y otro parámetro opcional que podría ser el identificador de un ítem. ¿Por que opcional? Bueno, en general si no se especifica un identificador de ítem, igualmente es posible encontrar cuales son los ítems de mayor preferencia para el usuario y luego recomendar nuevos ítems en base a este ítem inicial. Por otro lado, si ya se cuenta con un identificador de ítem, se puede recomendar ítems similares a este. Este último caso es muy común cuando un usuario navega al detalle de un producto en un *e-commerce*. En esta instancia, ya se conoce el identificador del usuario e ítem. Finalmente, se recomiendan ítems similares al ítem visualizado.

- Salidas: Es una lista de ítems recomendados similares a otro ítem (entrada del modelo) ordenados descendente-mente por la calificación predicha por el modelo para el usuario en cuestión (entrada del modelo).

Entonces, se podria decir que dada la salida de un sistema de recomendación, si encontramos en las primeras posiciones de la lista aquellos ítems con mayor calificación predicha, seria un buen indicador de que el modelo es preciso al momento de recomendar. En palabras mas simples, se desea que los primeros ítems de la lista de recomendaciones sean de mayor agrado para el usuario.

¿Finalmente, como funciona esta métricas? La métrica *mAP@k* funciona de la siguiente forma: Supongamos que tenemos un usuario y una lista K de ítems a recomendar. En base a estas entradas el modelo de recomendación predice las calificaciones de cada ítem para el usuario dado. Luego, podemos ordenar la lista de ítems descendente-mente de acuerdo a las calificaciones predichas por el modelo.

Teniendo esta lista, se puede calcular el promedio de la predicción *mAP@k* sobre los K ítems de la lista.

Esta métrica es utilizada en problemas de clasificación pero también se puede utilizar en problemas donde el modelo produce una salida numérica como en este caso. Los niveles o clases a utilizar dependen mucho de que se quiera evaluar. Supongamos, en este caso particular, que queremos medir con que precisión aparecen las puntuaciones entre 4 y 5 en las primera posiciones de la lista. Para este fin se utilizara la métricas *mAP@k*.

Promedio de la precisión sobre una lista de K elementos AP@k:

$$\text{AP@k} = \frac{1}{N(k)} \sum_{i=1}^{k} \frac{TP(i)}{i}, \qquad \text{N(k)} = min(k, TP_{total}) \tag{3.15}$$

Donde:

- i es la posición del ítem i^{th} en la lista de k elementos. i toma valores entre 1 y k.
- TP_i es 1 si la precision y el valor verdadero concuerdan.
- $N(k)$ es el mínimo entre el tamaño de la lista y la cantidad de TP_{total} encontrados en esa lista.

Por ejemplo, si se quiere saber con que precisión aparecen ítems con calificaciones entre 4 y 5 puntos en las primeras posiciones de la lista:

- Si TP_i es igual a 1, entonces la calificación en la posición i^{th} se encuentra entre los 4 y 5 punto.
- Si TP_i es igual a 0, entonces la calificación en la posición i^{th} NO se encuentra entre los 4 y 5 punto.

De esta forma, se esta transformando la salida del modelo en una lista de valores binarios. Donde la clase 1 indica que se cumple con la condición esperada y la clase 0 lo contrario.

Luego se realiza el calculo de *AP@k* para cada usuario del *dataset* de validación y finalmente se calcula la media:

Media del promedio de la precisión sobre una lista de K elementos mAP@k:

$$\text{mAP@k} = \frac{1}{N}\sum_{i=1}^{N} AP@k_i \tag{3.16}$$

De esta forma, la métrica *mAP@k* da una noción del grado de precisión en que aparecen ítems con mayor puntuación en las primeras posiciones de una lista de tamaño k. Cabe aclarar, que la condición *ítems con mayor puntuación* es arbitraria, aun que al ser una condición, podríamos intercambiarla por cualquier otro criterio. Por ejemplo, ítems con las peores puntuaciones (entre 1 y 2 puntos), ítems con puntuaciones medias, ítems con mas de 3 puntos, menos de 3 puntos, etc...

4. EXPERIMENTOS

Para comparar todos los modelos implementados, se utilizo el mismo *dataset*, tomando una muestra con el tamaño suficiente para obtener buenos resultados, evitando el sobre ajuste(*overfitting*) con aquellos modelos que tienden a sobre ajustar mas. Por otro lado, se realizo una modificación al modelo *KNN* para poder guardar sus resultados en una memoria *Cache*. De esta forma, no se repite la inferencia de las predicciones al momento de seleccionar muestras del conjuntos de validación, disminuyendo notablemente el tiempo de inferencia.

Por otro lado, cabe aclarar que dada la tendencia de los modelos a la variabilidad o varianza de sus predicciones, se realizo N veces un muestreo sobre el conjunto de validación para cada métrica utilizada. Luego, se gráfico un histograma de la distribución de la métricas, y un boxplot para tener una mejor idea de cual es su valor medio y la dispersion que se puede esperar.

A continuación se describen los resultados de todos los modelos comparados mediando las métricas *mAP@k* y *RMSE*.

4.1. Algoritmo de los K vencinos cercanos (*K-Nearest-Neighbor* o *KNN*)

4.1.1. Algoritmo de los K vencinos cercanos basado en usuarios (*KNN User Based*)

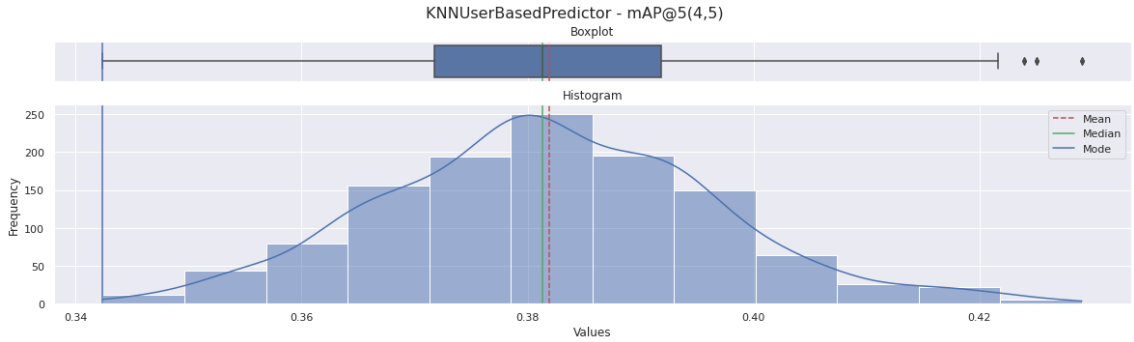

Fig. 4.1: Esta gráfica describe la distribución de valores de la métrica *mAP@5(4,5)* evaluada en el conjunto de observaciones de validación, luego de N procesos de entrenamiento del modelo *KNN User Based* sobre las observaciones de entrenamiento.

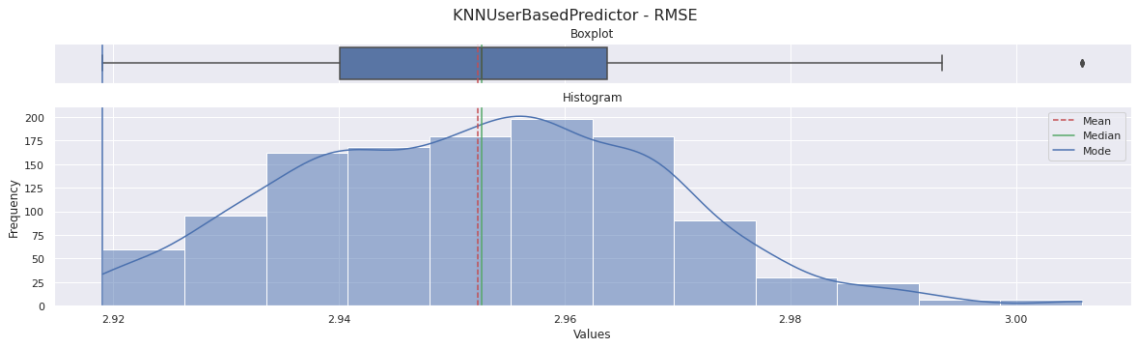

Fig. 4.2: Esta gráfica describe la distribución de valores de la métrica *RMSE* evaluada en el conjunto de observaciones de validación, luego de N procesos de entrenamiento del modelo *KNN User Based* sobre las observaciones de entrenamiento.

4.1.2. Algoritmo de los K vencinos cercanos basado en ítems (*KNN Item Based*)

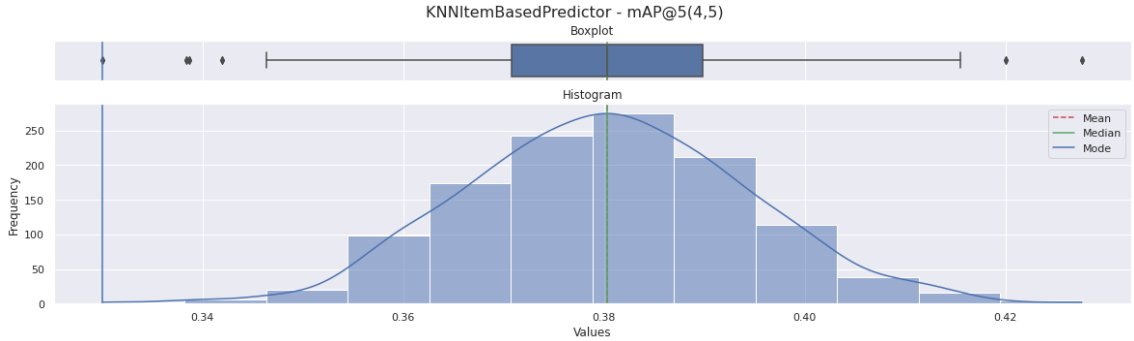

Fig. 4.3: Esta gráfica describe la distribución de valores de la métrica *mAP@5(4,5)* evaluada en el conjunto de observaciones de validación , luego de N procesos de entrenamiento del modelo *KNN Item Based* sobre las observaciones de entrenamiento.

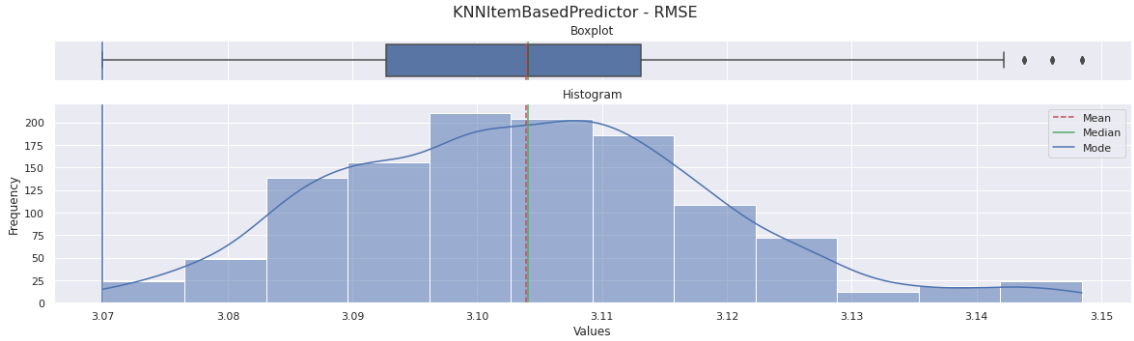

Fig. 4.4: Esta gráfica describe la distribución de valores de la métrica *RMSE* evaluada en el conjunto de observaciones de validación, luego de N procesos de entrenamiento del modelo *KNN Item Based* sobre las observaciones de entrenamiento.

4.1.3. Modelo ensamble de los algoritmos de los K vecinos cercanos basados en usuarios e ítems (*KNN User-Item Based Ensemble*)

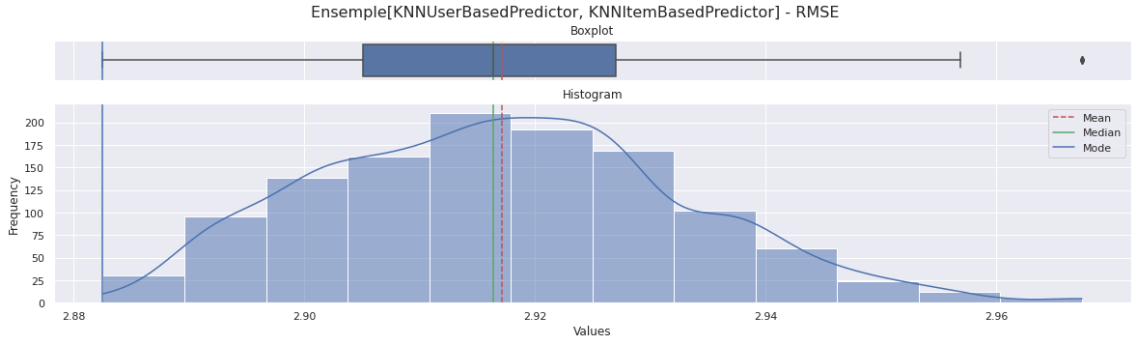

Fig. 4.5: Esta gráfica describe la distribución de valores de la métrica *RMSE* evaluada en el conjunto de observaciones de validación, luego de N procesos de entrenamiento del modelo *KNN* sobre las observaciones de entrenamiento.

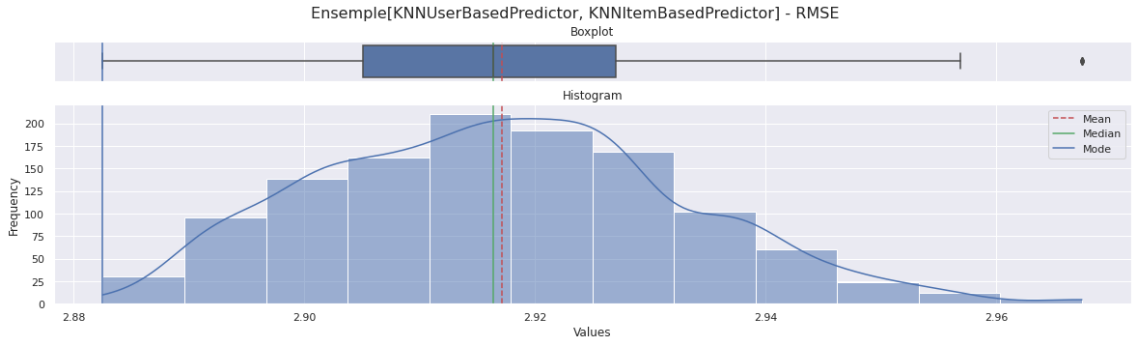

Fig. 4.6: Esta gráfica describe la distribución de valores de la métrica *RMSE* evaluada en el conjunto de observaciones de validación, luego de N procesos de entrenamiento del modelo *KNN* sobre las observaciones de entrenamiento.

4.2. Factorización Matricial General (*General Matrix Factorization o GMF*)

A continuación se puede apreciar las curvas de error (*MSE*) para el conjunto de validación y entrenamiento:

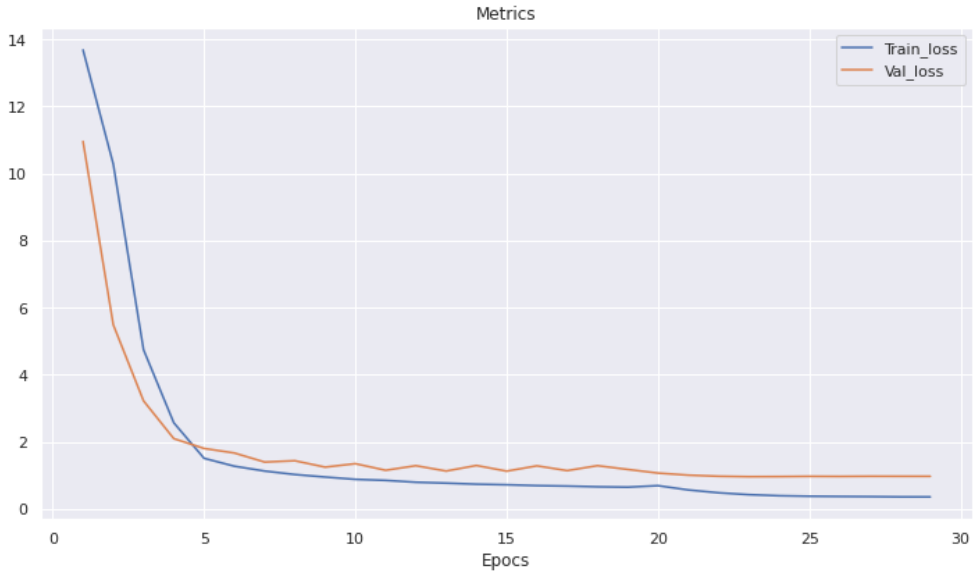

Fig. 4.7: Esta gráfica describe el nivel de error sobre los conjuntos de observaciones de entrenamiento y validación durante el entrenamiento del modelo *GFM*. Cada epoch o época indica una iteración de entrenamiento del modelo sobre el conjunto completo de entrenamiento.

Se puede apreciar que inicialmente el modelo tiene un error de valoración menor al error de entrenamiento. Es posible que se deba a que una pocas primeras observaciones de entrenamiento fueron suficientes para predecir con un error menor el conjunto de validación. A media que se incrementa el numero de épocas ya no es subiente y el modelo comienza a sobre ajusta hasta estabilizarse ambos errores.

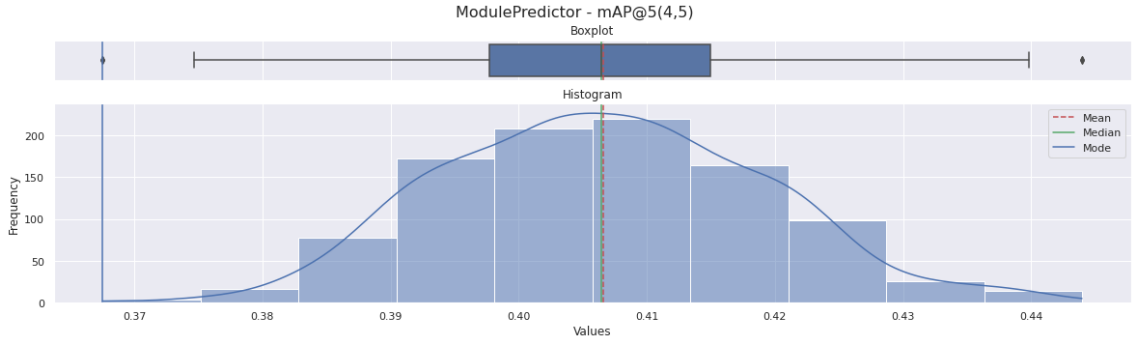

Fig. 4.8: Esta gráfica describe la distribución de valores de la métrica *mAP@5(4,5)* evaluado en el conjunto de observaciones de validación, luego de N procesos de entrenamiento del modelo *GFM* sobre las observaciones de entrenamiento.

Dado la tendencia de los modelos a la variabilidad o varianza de sus predicciones se realizo un *sampling* de cada métrica sobre el conjunto de validación N veces para comprender cual es su valor medio y dispersión.

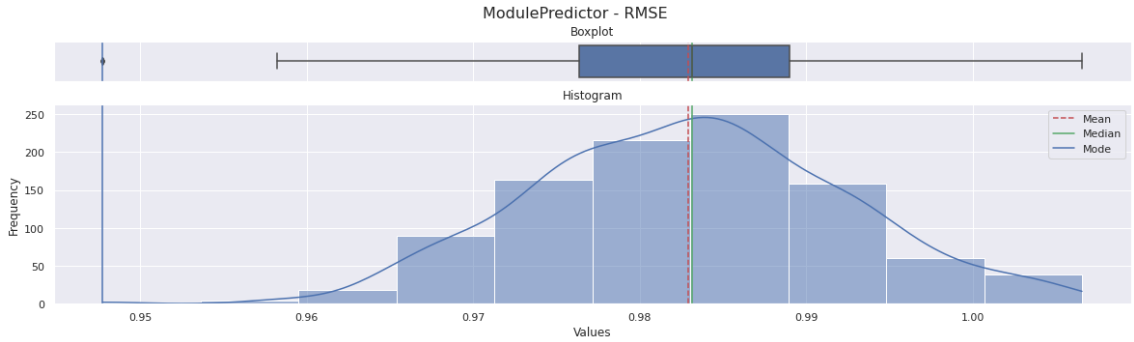

Fig. 4.9: Esta gráfica describe la distribución de valores de la métrica *RMSE* evaluado en el conjunto de observaciones de validación, luego de N procesos de entrenamiento del modelo *GFM* sobre las observaciones de entrenamiento.

4.3. Factorización Matricial General con Sesgo (*Biased General Matrix Factorization o B-GFM*)

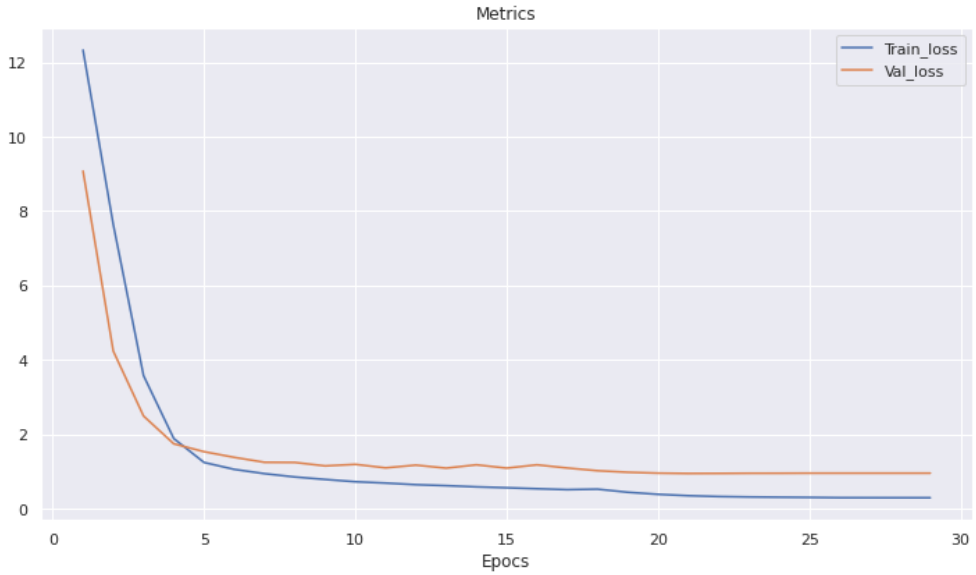

Fig. 4.10: Esta gráfica describe el nivel de error sobre los conjuntos de observaciones de entrenamiento y validación durante el entrenamiento del modelo *B-GFM*. Cada epoch o época indica una iteración de entrenamiento del modelo sobre el conjunto completo de entrenamiento.

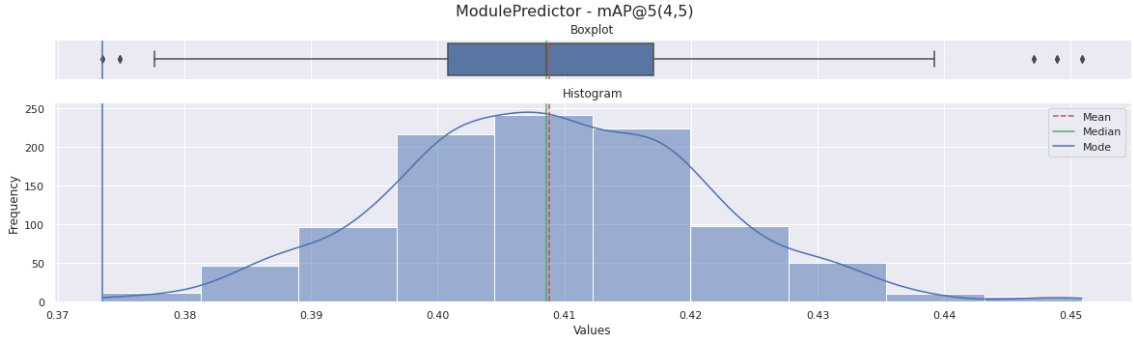

Fig. 4.11: Esta gráfica describe la distribución de valores de la métrica *mAP@5(4,5)* evaluado en el conjunto de observaciones de validación, luego de N procesos de entrenamiento del modelo *B-GFM* sobre las observaciones de entrenamiento.

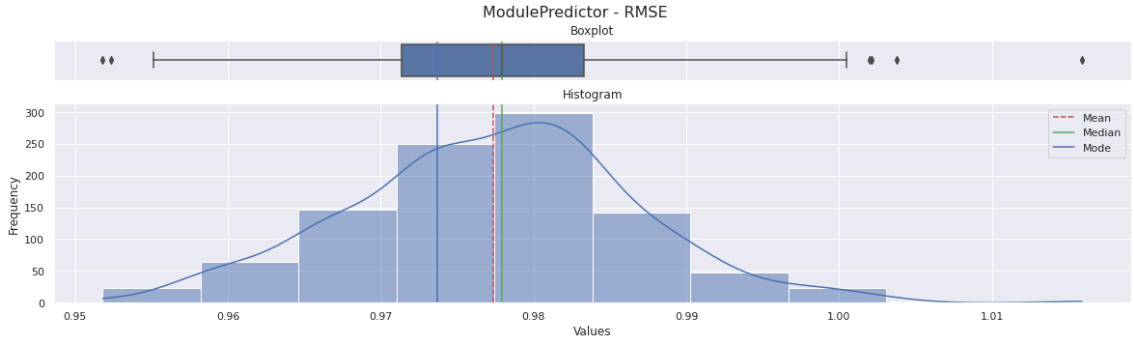

Fig. 4.12: Esta gráfica describe la distribución de valores de la métrica *RMSE* evaluado en el conjunto de observaciones de validación, luego de N procesos de entrenamiento del modelo *B-GFM* sobre las observaciones de entrenamiento.

4.4. Factorización Matricial mediante Redes Neuronales (*Neural Network Matrix Factorization o NN-FM*)

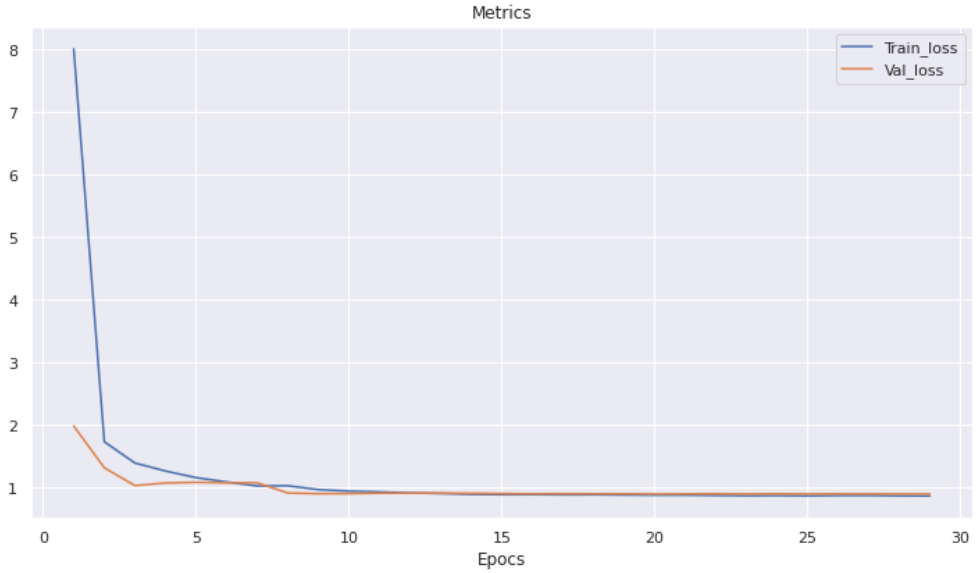

Fig. 4.13: Esta gráfica describe el nivel de error sobre los conjuntos de observaciones de entrenamiento y validación durante el entrenamiento del modelo *NN-FM*. Cada epoch o época indica una iteración de entrenamiento del modelo sobre el conjunto completo de entrenamiento.

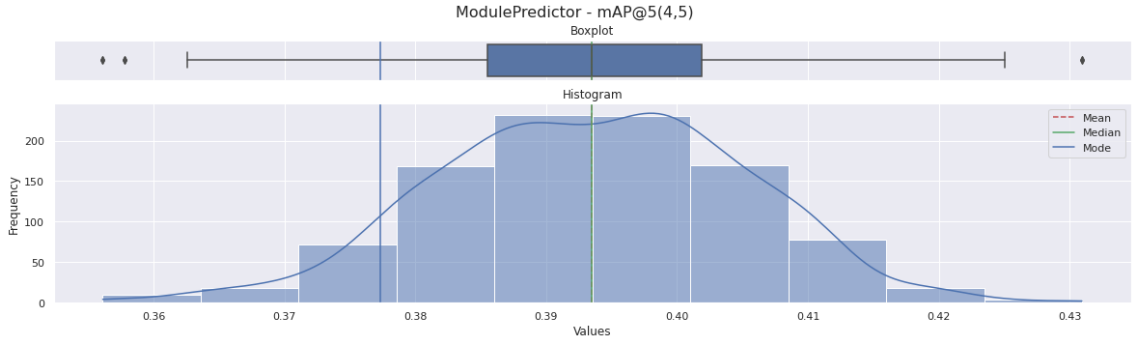

Fig. 4.14: Esta gráfica describe la distribución de valores de la métrica *mAP@5(4,5)* evaluado en el conjunto de observaciones de validación, luego de N procesos de entrenamiento del modelo *NN-FM* sobre las observaciones de entrenamiento.

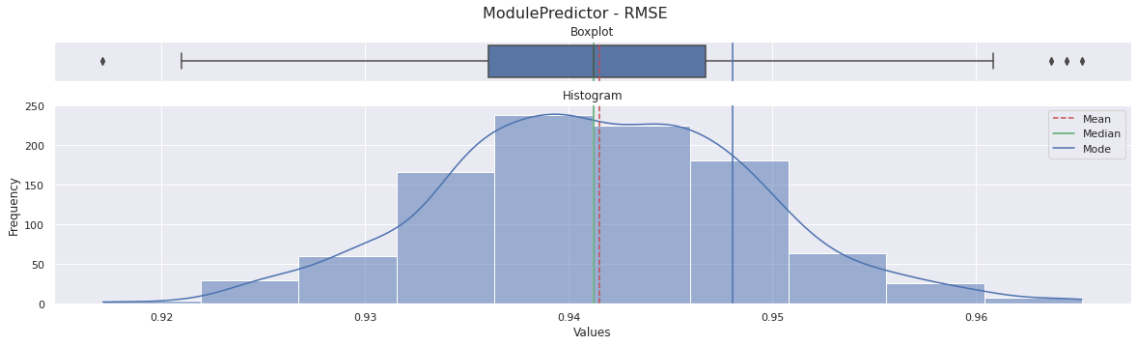

Fig. 4.15: Esta gráfica describe la distribución de valores de la métrica *RMSE* evaluado en el conjunto de observaciones de validación, luego de N procesos de entrenamiento del modelo *NN-FM* sobre las observaciones de entrenamiento.

4.5. Máquinas de factorización profundas (*DeepFM*)

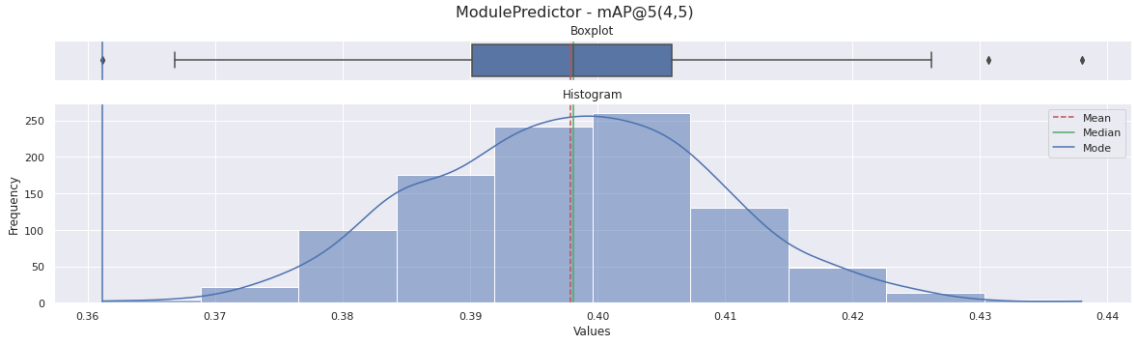

Fig. 4.16: Esta gráfica describe la distribución de valores de la métrica *mAP@5(4,5)* evaluado en el conjunto de observaciones de validación, luego de N procesos de entrenamiento del modelo *DeepFM* sobre las observaciones de entrenamiento.

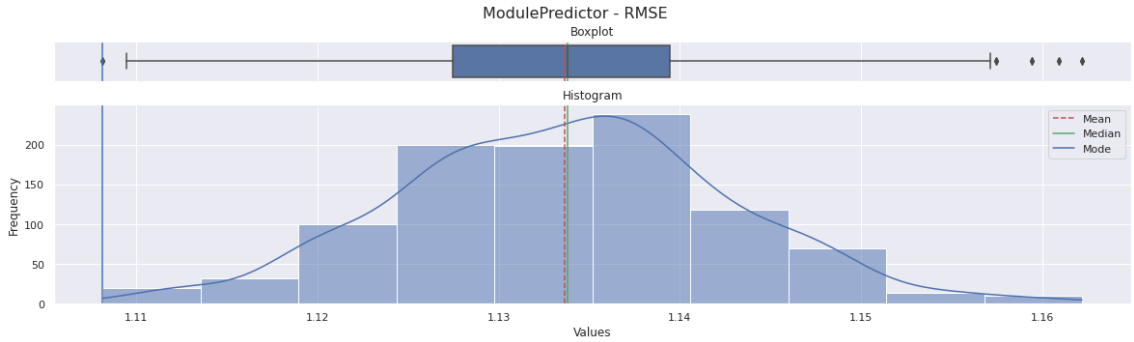

Fig. 4.17: Esta gráfica describe la distribución de valores de la métrica *RMSE* evaluado en el conjunto de observaciones de validación, luego de N procesos de entrenamiento del modelo *DeepFM* sobre las observaciones de entrenamiento.

5. RESULTADOS

En esta sección se llevará a cabo una comparación de todos los modelos previamente expuestos. Para realizar esta comparación, se emplearán las métricas presentadas en secciones anteriores. A modo de resumen, las métricas consideradas fueron las siguientes:

- *Mean Average Precision at k (mAP@k)*: Esta métrica representa el promedio de la precisión al calificar los primeros K ítems de una lista de recomendaciones para un usuario específico. Para obtener más detalles, se puede consultar la sección de referencia sobre *mAP@k* (consultar la Subsección 3.3.2).
- *Root Mean Square Error (RMSE)*: Esta métrica corresponde a la raíz cuadrada del error cuadrático medio entre la calificación real de un ítem, proporcionada por un usuario, y la calificación predicha. Para obtener más información, se puede consultar la sección de referencia sobre *RMSE* (consultar la Subsección 3.3.1).

Para calcular estas métricas, se realizo un muestreo para determinar su distribución y, de esta forma, definir cada métrica en términos de su mediana, media y desvío.

El uso de la media y el desvío estándar permite obtener el valor promedio de las métricas y evaluar su grado de dispersión en los datos obtenidos. La media nos proporciona una idea del valor típico de las métricas en el conjunto de prueba, mientras que el desvío estándar nos indica qué tan dispersos están los valores las metricas en relación con la media.

Además, se presenta la mediana como otra medida estadística relevante. A diferencia de la media, la mediana no se ve afectada por valores atípicos o extremos en el conjunto de datos, lo que la convierte en una medida más robusta para describir la ubicación central de las métricas.

A continuación, se presenta la comparación de todos los modelos expuestos utilizando el promedio de la precisión para el rango de calificaciones entre 4 y 5 puntos, considerando una lista de 5 items recomendados, que se denota como *AP@5(4,5)*:

Modelo	Mediana	Media	Desvío
B-GMF	0.408563	0.408787	0.012190
GMF	0.406422	0.406646	0.012513
DeepFM	0.398100	0.397895	0.011369
NN-MF	0.393499	0.393447	0.011711
KNN User-Item Based Ensemble	0.384570	0.384819	0.015066
KNN User Based	0.381297	0.381943	0.014709
KNN Item Based	0.380284	0.380327	0.014056

Tab. 5.1: Mediana, media y desvío correspondientes a la distribución de *AP@5(4,5)* muestreada para cada modelo. Las filas se encuentran ordenadas descendente-mente por la media.

En la tabla 5.1, se puede observar inicialmente que el modelo *Biased-GMF* muestra los mejores resultados en términos de la métrica de evaluación *AP@5(4,5)*. Por otro lado, los modelos *NN-MF* y *Deep-FM* exhiben el menor sesgo, pero aún así, tienen una precisión inferior al modelo *Biased-GMF*. Esto podría indicar un mayor grado de sobreajuste en estos modelos. Por lo tanto, sería recomendable reentrenar ambos modelos con un aumento en

el valor del parámetro *dropout* para mejorar la regularización y, posteriormente, comparar nuevamente sus resultados con el modelo *Biased-GMF* para validar si su precisión mejora.

En cuanto a la familia de modelos *KNN*, se observa que presenta el mayor sesgo. Sin embargo, a pesar de esto, se puede notar que la diferencia en la precisión, en comparación con los demás modelos, es bastante baja, siendo menos del 2

En resumen, los resultados muestran que el modelo *Biased-GMF* sobresale en precisión según la métrica *AP@5(4,5)*. Los modelos *NN-MF* y *Deep-FM*, aunque tienen menos sesgo, deben ser reentrenados con un mayor valor de *dropout* para potencialmente mejorar su precisión. Por otro lado, la familia de modelos *KNN*, a pesar de presentar un sesgo más alto, aún logra una precisión cercana a los demás modelos en comparación.

A continuación, se presenta la comparación de todos los modelos expuestos utilizando utilizando la raíz cuadrada del error cuadrático medio (*RMSE*):

Modelo	Mediana	Media	Desvío
NN-MF	0.941213	0.941493	0.007620
B-GMF	0.977914	0.977382	0.009387
GMF	0.983141	0.982894	0.009419
DeepFM	1.133796	1.133637	0.009183
KNN User-Item Based Ensemble	2.916418	2.917146	0.015642
KNN User Based	2.952661	2.952305	0.016170
KNN Item Based	3.104068	3.103917	0.014975

Tab. 5.2: Mediana, media y desvío correspondientes a la distribución de *RMSE* muestreada para cada modelo. Las filas se encuentran ordenadas descendente-mente por la media.

En la tabla 5.2, se puede observar que el modelo *NN-MF* parece ser el más estable en términos del error de validación, ya que presenta el menor error y dispersión. Sin embargo, a pesar de su estabilidad, como se puede apreciar en la tabla 5.1 (anteior) no es el modelo más preciso. Este hecho podría sugerir que *NN-MF* estaría experimentando un grado significativo de sobreajuste en comparación con otros modelos que logran una mayor precisión.

Un ejemplo de esto es el modelo *Biased-GMF*, el cual muestra una mayor precisión en comparación con *NN-MF*, pero también presenta un error mayor. Esto indica que la teoría del sobreajuste aplicada a *NN-MF* podría ser válida y explicar su menor precisión en comparación con *Biased-GMF*.

Además, se nota que la familia de modelos *KNN* muestra los errores más altos junto con desvíos elevados. Sin embargo, es interesante destacar que, a pesar de estos altos errores, estos modelos presentan una precisión muy similar al modelo más preciso, *B-GMF*.

6. CONCLUSIONES

En resumen, al analizar los modelos expuesto en este trabajo practico, se encontró que todos tienen una precisión similar, con una diferencia de menos del 2 %. Sin embargo, al considerar la implementación en un *e-commerce*, se recomendaría elegir un modelo basado en *deep learning*, como *B-GMF* o *GMF*, ya que utilizan el algoritmo del gradiente descendente y pueden procesar las observaciones de entrenamiento en lotes, lo que permite ajustar el tamaño del lote según la memoria *RAM* o *VRAM* disponible. Por el contrario, no seria posible seleccionar modelos de la familia *KNN* debido a que necesitan alocar todas las observaciones en entrenamiento en memoria, impidiendo escalar el conjunto de entrenamiento, limitando la ventana de datos a considerar para el entrenamiento del modelo.

Por otro lado, se observó que el modelo de estado del arte, *DeepFM*, no obtiene la precisión más alta y su rendimiento es prácticamente igual al de los modelos de la familiacpara el conjunto de datos estudiado. En general, los modelos de *deep learning* presentaron mejores resultados en este conjunto de pruebas en comparación con los modelos clásicos como *KNN*, aunque las diferencias fueron son muy pequeñas.

REFERENCIAS

[1] GroupLens, "Movielens 25m dataset," 2019, fecha de acceso: 2 de febrero de 2021. [Online]. Available: https://grouplens.org/datasets/movielens/25m

[2] R. BANIK, "Tmdb movie dataset," 2017, fecha de acceso: 2 de febrero de 2021. [Online]. Available: https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset?select=movies_metadata.csv

[3] C. de editores de Wikimedia, "Análisis de componentes principales," 2023, fecha de acceso: 23 de julio de 2023. [Online]. Available: 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.

[4] ——, "Biplot," 2023, fecha de acceso: 23 de julio de 2023. [Online]. Available: https://es.wikipedia.org/wiki/Biplot

[5] C. Saluja, "Collaborative filtering based recommendation systems exemplified.." 2018, fecha de acceso: 2 de febrero de 2021. [Online]. Available: https://towardsdatascience.com/collaborative-filtering-based-recommendation-systems-exemplified-ecbffe1c20b1

[6] S. Funk, "Algoritmo de factorización de matrices," 2006, fecha de acceso: 2 de febrero de 2021. [Online]. Available: https://sifter.org/~simon/journal/20061211.html

[7] Wikipedia, "Netflix prize," 2022, fecha de acceso: 2 de junio de 2022. [Online]. Available: https://en.wikipedia.org/wiki/Netflix_Prize

[8] F. G. Dietmar Jannach, Lukas Lerche, "What recommenders recommend – an analysis of accuracy, popularity, and sales diversity effects," 2022, fecha de acceso: 2 de junio de 2022. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-38844-6_3

[9] J. Wittenauer, "Deep learning with keras: Recommender systems," 2019, fecha de acceso: 2 de febrero de 2021. [Online]. Available: https://www.johnwittenauer.net/deep-learning-with-keras-recommender-systems

[10] D. M. R. Gintare Karolina Dziugaite, "Neural network matrix factorization," 2015, fecha de acceso: 2 de abril de 2022. [Online]. Available: https://arxiv.org/pdf/1511.06443.pdf

[11] H. Z. Xiangnan He, Lizi Liao, "Neural collaborative filtering," 2017, fecha de acceso: 2 de febrero de 2021. [Online]. Available: https://arxiv.org/pdf/1708.05031.pdf

[12] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Factorization machines," 2021, fecha de acceso: 20 de junio de 2022. [Online]. Available: https://medium.com/qloo/popular-evaluation-metrics-in-recommender-systems-explained-324ff2fb427d#:~:text=Precision%20and%20recall%20are%20evaluation,user%20query%20in%20our%20case.

[13] ——, "Dive into deep learning," *arXiv preprint arXiv:2106.11342*, 2021.

[14] S. Rendle, "Factorization machines," 2010, fecha de acceso: 2 de febrero de 2022. [Online]. Available: https://www.csie.ntu.edu.tw/~b97053/paper/Rendle2010FM.pdf

[15] Y. Y. Huifeng Guo, Ruiming Tang, "Deepfm: A factorization-machine based neural network for ctr prediction," 2021, fecha de acceso: 2 de febrero de 2021. [Online]. Available: ttps://arxiv.org/pdf/1703.04247.pdf

[16] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Deep factorization machines," 2021, fecha de acceso: 20 de junio de 2022. [Online]. Available: https://d2l.ai/chapter_recommender-systems/deepfm.html

[17] J. H. HengTze Cheng, Levent Koc, "Wide and deep learning for recommender systems," 2016, fecha de acceso: 2 de marzo de 2021. [Online]. Available: https://arxiv.org/pdf/1606.07792.pdf

[18] A. U., "How mean average precision at k (map@k) can be more useful than other evaluation metrics," 2020, fecha de acceso: 20 de abril de 2022. [Online]. Available: https://medium.com/@misty.mok/how-mean-average-precision-at-k-map-k-can-be-more-useful-than-other-evaluation-metrics-6881e0ee21

[19] R. Brideau, "Precision@k: The overlooked metric for fraud and lead scoring models," 2021, fecha de acceso: 20 de abril de 2022. [Online]. Available: https://towardsdatascience.com/precision-k-the-overlooked-metric-for-fraud-and-lead-scoring-models-fabad2893c01

[20] G. Papachristoudis, "Popular evaluation metrics in recommender systems explained," 2019, fecha de acceso: 19 de abril de 2022. [Online]. Available: https://medium.com/qloo/popular-evaluation-metrics-in-recommender-systems-explained-324ff2fb427d#:~:text=Precision%20and%20recall%20are%20evaluation,user%20query%20in%20our%20case.