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

Sistemas de recomendación colaborativos e híbridos

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

Adrian Norberto Marino

Buenos Aires, 2022

RESUMEN

Acá va un resumen del trabajo. Con el resumen se deberia poder tener una idea del trabajo en su totalidad. Desde los obetivos y los datos hasta los resultados y conclusiones. Se escibe al final. (aprox. 200 palabras).

Palabras claves: representativas del trabajo, los métodos, los datos (no menos de 5).

Índice general

1. INTRODUCCIÓN

Los sistemas de recomendación tienen por objetivo acercar a sus usuarios información productos, contenido (Textos, audio, videos), etc..relevantes a sus preferencias o necesidades, permitiendo a estos encontrar con mayor facilidad aquello que buscan.Formalizando esta definición podemos decir que: Los sistemas de recomendación apuntan a ayudar a un usuario o grupo de usuarios a seleccionar items de forma personalizada dado un de conjunto de items de gran extensión o un gran espacio de búsqueda.

Este objetivo puede cambiar según el contexto de cada negocio. Para un e-commerce de delivery de comidas, el objetivo es acercar a los usuarios el tipo de comida que quieren probar en ese mismo momento, a un precio que puedan pagar con tiempo de entrega aceptable. Para un e-commerce de venta de productos, se busca acercar al usuario aquellos productos que este necesitando en ese mismo momento, los cuales tienen un precio que el mismo puede pagar y por otro lado, asegurar una experiencia satisfactoria con el vendedor. En el negocio de visualización de contenido (Ya sea audio o video), el objetivo es acercar al usuario contenido a fin a sus gustos para mejorar su experiencia en la plataforma y así aumentar el engagement de sus usuarios.

Por otro lado, el objetivo de fondo siempre es el mismo, mejorar la conversión. Con esto nos referimos a aumentar el volumen de ventas para un e-commerce de venta de productos, la cantidad de deliveries mensuales, la cantidad de impresiones de publicidad en aplicaciones de visualización de contenido, aumentar el tiempo de permanecía en las plataformas de streaming de audio o video, etc.. Podemos encontrar muchos ejemplos distintos donde el objetivo común es mejorar la conversión y engagement de los usuarios.

Desde un punto de vista mas técnico, los sistemas de recomendación se utilizan para predecir el grado de preferencia de un usuario con respecto a un item. En general, se puede lograr aplicando de un algoritmo de optimización, el cual minimiza la diferencia entre el grado de preferencia esperado versus real. Otros enfoques hace uso de medidas de distancia para establecer este grado de preferencia.

1.1. Tipos de sistemas de recomendación

A continuación se puede ver un gráfico que describe las clasificaciones y subclasificaciones de los sistemas de recomendación:
8

1.1.1. Basados en Popularidad

Este tipo de recomendador toma algunas características de popularidad de los items en cuestión, como puede ser: cantidad de vistas, cantidad de compras, cantidad de reviews positivos, etc.. Luego resuelve el top K de los items mas populares según estos criterios. Si bien este tipo de recomendaciones tiene buenos resultandos para usuarios nuevos, de los cuales no se conocen sus preferencias, al carecer de personalización, sus recomendaciones no tienen en cuenta las preferencias de cada usuario particular, debido a que se basan en estadísticas comunes a todos los usuarios. Por esta cuestión no son considerados sistemas de recomendación per se.

1.1.2. Basados en Contenido

A diferencia de los recomendadores basados en filtros colaborativos, este tipo de recomendador necesita un trabajo previo de ingeniería de features sobre los items, donde se busca definir que features son los mas significativos para la tarea en cuestión, y cual es el grado de adecuación de cada items con los features definidos. Por otro lado, es necesario registrar las interacciones de los usuarios. Dada estas interacciones, se puede definir el grado de preferencia de los usuarios a cada feature definido para los items. Con esta información es posible encontrar tanto items como usuarios similares y realizar recomendaciones del tipo:

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

La principal desventaja de este enfoque es que es necesario realizar ingeniería de features para encontrar los features que produzcan las mejores recomendaciones. El modelo no encuentra estos features sino que deben ser definidos de antemano. Como ventaja, si se encuentra los features correctos se pueden lograr buenos resultados.

1.1.3. Basados en Filtrado Colaborativos

Estos modelos, a diferencia de los basados en contenido, no requiriere ingeniería de features, lo que los hace mas simples de implementar, ya que únicamente es necesario registrar las interacciones de los usuarios para con los items. Ejemplos interacciones podrían ser:

- El usuario A visualizo el item X el dia 2 de marzo de 2022.
- El usuario A compro el item X el dia 10 de marzo de 2022.
- El usuario A califico al item X con un 5 el dia 25 de marzo de 2022.

Por otro lado, estos modelos personalizan sus recomendaciones, es decir que ajustan las recomendaciones a cada usuario particular, en base a sus preferencias, al igual que los basados en contenido. De igual forma que los basados en contenido se puede encontrar usuario e items similares y recomendar items entre usuarios similares.

Estos modelos aprende un espacio latente de soluciones sin necesidad de recolectar datos y definir features, lo cual puede llevar a una solución sesgada. Por otro lado, no todo son rosas con estos modelos, ya que sufren un problema llamado cold start o arranque en frio. Si pensamos en una solución donde alimentamos al modelo con una ventana de interacciones de usuario-item filtrando 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 modelo pueda realizar recomendaciones acertadas, ya que con pocas interacciones no podrá realizar buenas recomendaciones.
- Usuarios con pocas interacciones: Por otro lado, tenemos a los usuarios que tienen una baja velocity en cuando a interacciones con el sistema o aplicación. Si pensamos en un e-commerce, hay usuario que compran con mucha frecuencia y otra compras muy de vez en cuando. Estos últimos en general tendrán pocas interacciones pudiendo caer por debajo del umbral mínimo que requiere el modelo. De esta forma, tendremos usuario que quedaran fuera del modelo en forma cíclica.
- Usuarios con muchas interacciones: Este es el caso ideal, donde el usuario tiene una gran cantidad de interacciones usuario-item. Para estos usuarios el modelo podrá ofrecer mejores recomendaciones, ya que cuanto mas interacciones se tengan, el modelo se ajusta con mas facilidad a sus preferencia.

1.1.4. Híbridos

Son aquellos modelos que combinan mas de una técnica de recomendación, también llamados ensambles de modelos.

1.1.5. Tipos dentro de los 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 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 se suele usar una cache para realizar las búsqueda una única vez. Otro problema es su limitación al tamaño máximo de la memoria con la que se cuente, es decir que el tamaño de la matrix depende de la memoria maxima. Esto puede mitigarse utilizando implementaciones de matrices esparzas, las cuales comprimen los datos en memoria guardando unicamente las celdas que tiene datos. Ademas, es posible utilizar un 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 buscan es encontrar una representación mas compacta de los items y usuarios sin perder información. Mas allá de tener que lidiar con esto problema de escalabilidad, los resultado obtenidos por estos modelos no están muy alejados de aquellos que se encuentra 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) en memoria. 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 las siguientes preguntas:

1.2.1. ¿Los modelos basado en filtro colaborativos que utilizan técnicos de deep learning obtienen mejores resultados?

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 cuente.

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?

Como ya se comentó en la introducción, los modelos de filtro colaborativos necesitan un número mínimo de interacciones usuario-item 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 ensamples de modelos basados en filtros colaborativos con otros modelo basados en contenidos. Estos ensamples puede diferir en sus técnicas dependiendo del dominio de los datos.

1.3. Trabajos previos

COMPLETAR

1.4. Objetivos

Como primer objetivos, se pretender comprender cuales son los fundamentos teóricos sobre los que se apoya cada técnica aplicada y bajo que escenarios puede ser con 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 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. Finalmente, se compararan esta técnicas mediante el 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 definidor 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 items(películas en este caso). Ademas, dado que se busca solucionar el problema de cold-start para el enfoque de filtros colaborativos, se necesitara 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 items(películas).

Dadas estas necesidades se decidió utilizar los siguientes datasets:

2.1.1. MovieLens 25M Dataset

Esta dataset prácticamente no tiene features para los items(películas) pero si tiene las calificaciones realizadas por los usuarios. También se cuenta con un conjunto de tags o palabras clase cargadas por los usuarios para cada item(película). Otro punto importante, es que todos los usuarios tienen al menos 20 interacciones, lo cual asegura que no habrá problemas de baja performance por falta de datos. De esta forma, este dataset sera muy util para entrenar modelos de recomendación basados en filtros colaborativos y ademas cuenta con columnas extras como tags, que serán útiles a la hora de entrenar modelos basados en contenido.

2.1.2. TMDB Movie Dataset

Este dataset no tiene calificaciones personalizadas de los items como si sucede con el dataset anterior, pero tiene varios features de los items que pueden ser muy útiles para modelos basados en contenido e inclusive modelos híbridos, los cuales buscamos explorar.

2.1.3. Preprocesamiento

Como parte inicial se la etapa de pre-procesamiento de datos se utilizo una base de datos MongoDB. Se utilizo MongoDB y no pandas debido a que pandas requiere cargar todo el dataset en memoria. Si bien este problema se puede lidiar aumentado el tamaño de la memoria swap(Linux) o la memoria virtual(windows) puede ocasionar caída de procesos y lentitudes innecesarias. En este caso se selecciono una base de datos de tipo document, 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 dataset contiene una varios archivos csv los cuales vamos a llamar tablas, de los cuales se utilizaron los siguientes:

- movie_metadata: Pertenece al TMDB dataset. Es la fuente de verdad de donde tomamos columnas con datos de películas.

- tags: Pertenece al dataset MovieLens. De esta tabla se tomaron los tags o palabras clase datas de alta por los usuarios para cada película.
- ratings: Pertenece al dataset MovieLens. De esta tabla se tomaron las calificaciones de los usuario para las películas que fueron calificadas.

De esta forma primero se hizo un merge o join de las tabla 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 hizo merge entre las tablas ratings_tags_v1 y movie_metadata utilizando la columnas imdb_id, la cual es el único identificador único de las películas en ambas tablas.

Finalmente se termino con dos tablas como resultado:

- 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 las tags para cada usuario y película.

Tabla de interacciones

La tabla ratings_tags_v1 tiene datos a nivel interacción usuario-item. De esta forma a nivel usuario-item se cuenta con la calificación de la película realizada por el usuario, ademas de los tags que el usuario cargo o ingreso para esa películas. Estos tags no son mas una lista de palabras que son representativas de la película en cuestión. Por ejemplo, para la película Toy Store 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.

- user_id: Existen 13.281 usuarios.
- movie_id: Existen 33.444 películas.
- timestamp: Fecha en la cual el usuario califico el item(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, 5.
- tags: Lista de palabras definidas por cada usuario para una película. Se cuenta con los tags a nivel usuario-película.

Tablas de metadata de películas

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

- 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 adiete la película.
- overview: Sinopsis de la película.
- poster: Enlaces al detalle de la película en imdb y the movie database. Estos enlaces permiten hace join con mas datos que se encuentren en la Descripción de estos sitios pero en este trabajo solo se utilizara la imagen pata de la película a modo de visualización.
- release: Fecha de lanzamiento.
- budget: Presupuesto destinado para realizar el film.
- popularity: Popularidad.
- vote_count: Cantidad de votos.
- vote_mean: Medio de votos por película.
- tags: Son los tags cargados por todos los usuario que interactuaron con la película. Es la mismos información que tenemos en la tabla de interacción pero ahora a nivel item.

Valores faltantes

Una vez generadas ambas tablas se procedió a buscar missing values. A continuación en la tabla 2.1 se pueden ver las columnas con missing values:

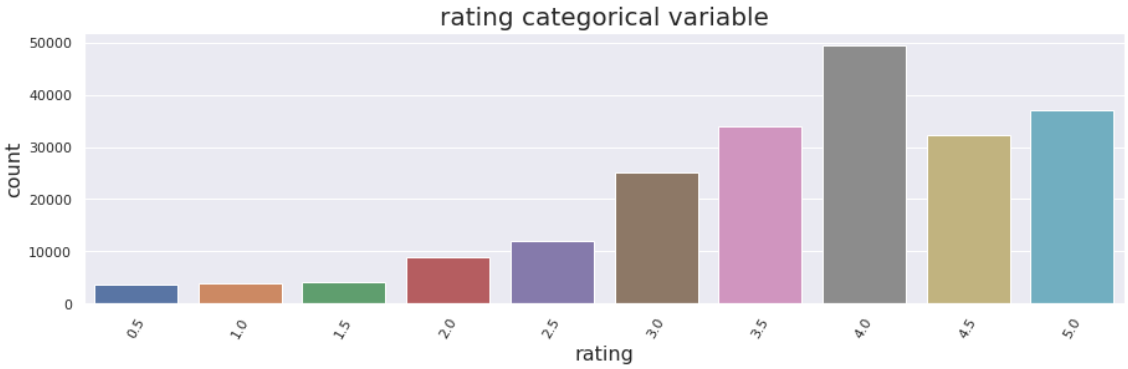
Columna	Porcentaje de missing values
budget	70
poster	0.00085
release	0.0085
popularity	0.00085
vote_mean	4.8
vote_count	4.6

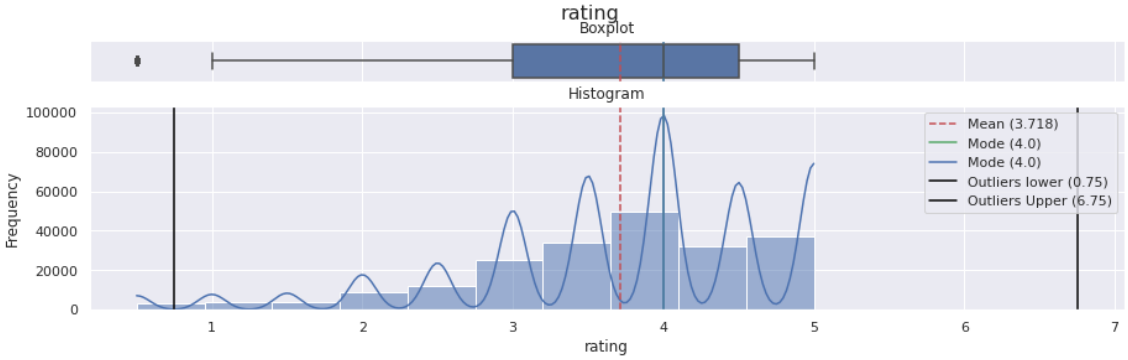
Tab. 2.1: Missing values en la tabla movies_v4.

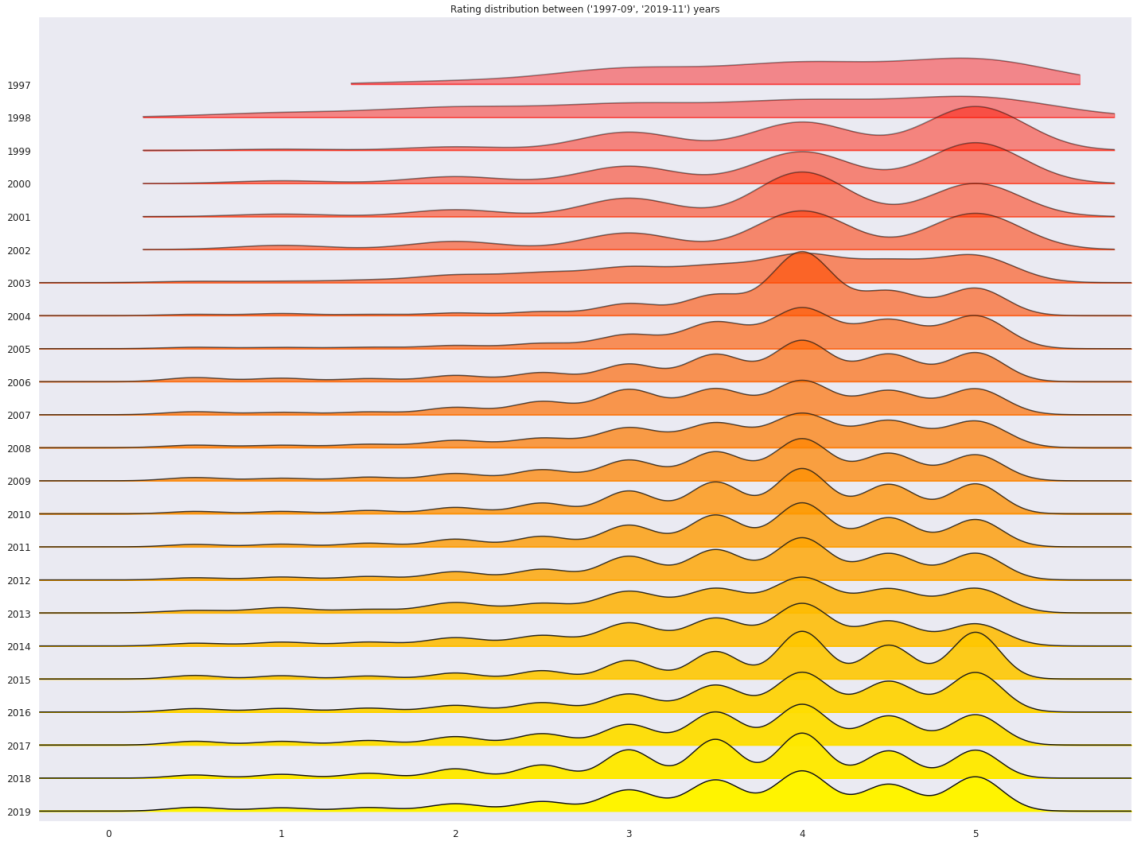
Luego se removieron las filas de la tabla para aquellas columnas del reporte anterior que tuvieran hasta 0.06 % de valores faltantes. A continuación se removió la columna budget por tener un porcentaje muy alto de valores faltantes lo que la volvió inutilizable. Por otro lado, la tabla ratings_tags_v1 no se modifico, ya que no tenia valides faltantes en ninguna de sus columnas.

2.2. Análisis exploratorio

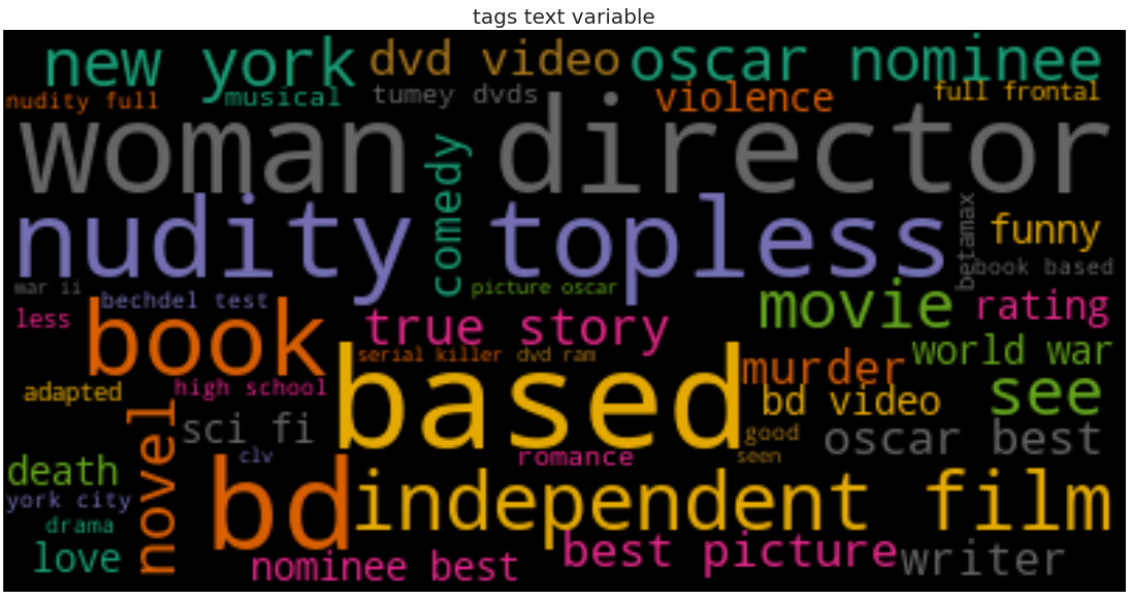
2.2.1. Variable Rating







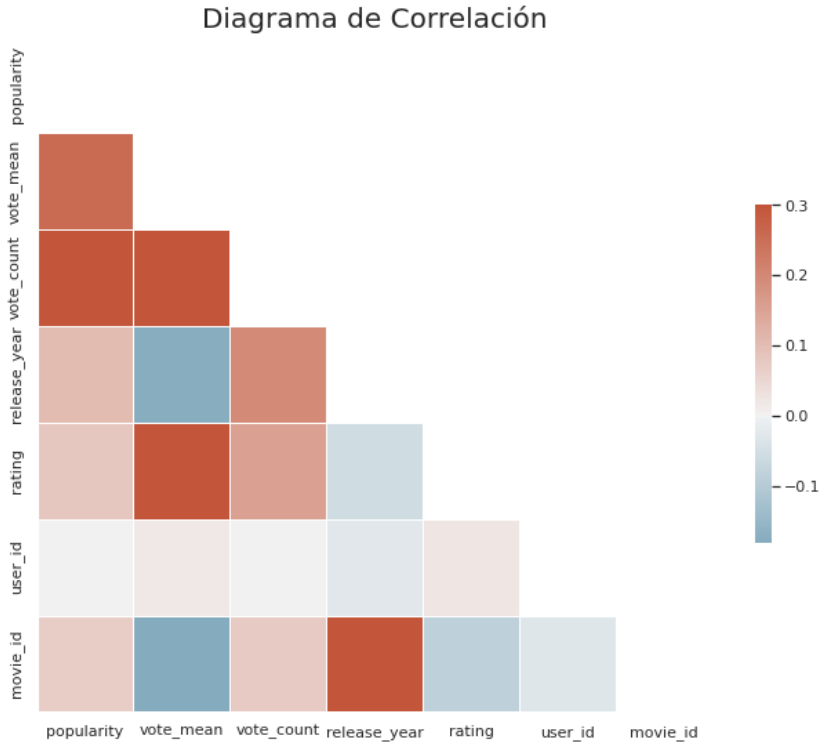
2.2.2. Variables de tipo texto



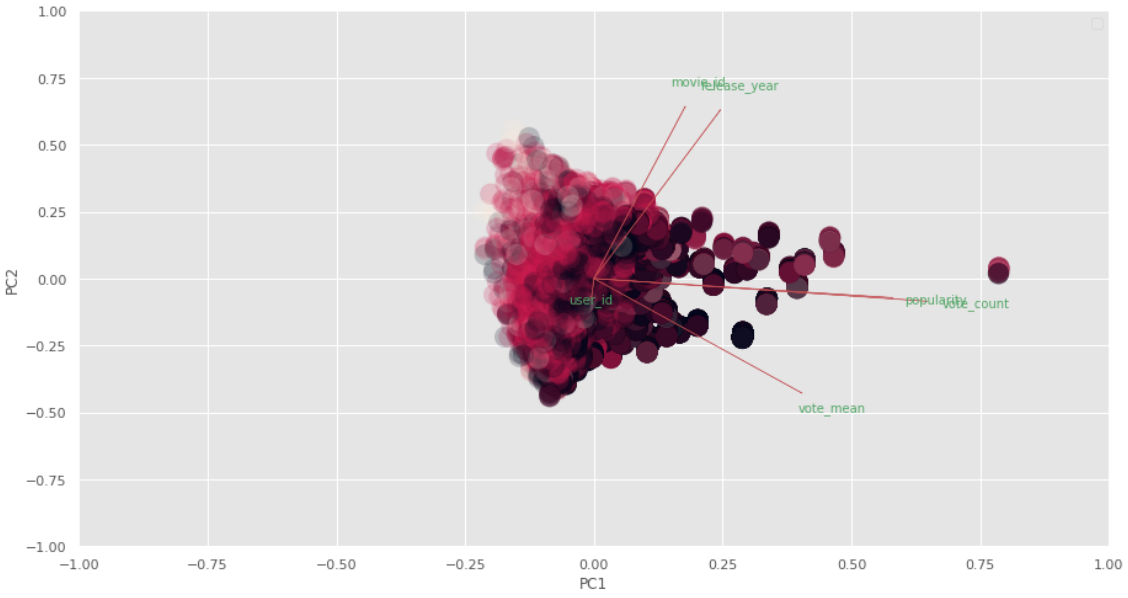
tags text variable

title text variable

2.2.3. Correlaciones



2.2.4. Componente principales



3. MÉTODOS

En este capitulo se describirán los modelos utilizados para realiza la predicción de las 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: Cuanto puede escalar, sus tiempos de entrenamiento y predicción, su implementación, exactitud de las predicciones, tendencia al overfitting, 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 y por el otro, modelos basados en deep learning. Los modelos basados en deep learning utilizan embedding en todos los casos, como una forma de reducir la dimensionalidad de las variables categóricas que se utilizan como entradas. Ademas, 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, entrenado con el mismo dataset en todos los casos. De esta forma podemos compara los resultados de cada modelo. K vecinos cercanos (KNN) fue tomado como baseline a partir del cual poder comparar los demás modelos.

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 ya desarrollados de forma oficial. De esta forma implementamos cada modelo desde cero para poder hacer uso de CPU y GPU de forma granular y realizar un usu mas eficiente de los recursos disponible.

3.1. Enfoque basados en memoria

3.1.1. KNN para predicción de la calificación de películas

Esta es la implementación clásica y mas intuitiva para realizar la recomendación de items. Una vez entrenado el modelo, se cuenta con una matriz de distancias que pueden ser distancias entre usuario o items, y otra matriz de calificaciones usuario-item. De esta forma, en la etapa de inferencia, el modelo tomo como entrada un usuario(id) y un item(id) y retorna la predicción de la calificación. Estas matrices se puede mantener en memoria o bien 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

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

Donde:

- u es un identificador 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 se no usar matrices esparzas.
- i es un identificador univoco y numérico de un item también secuencial. En nuestros caso los items son películas, pero podrían ser cualquier entidad identificable como productos, usuarios, comidas, etc..
- $r_{u,i}$ es la calificación otorgada al item i por parte del usuario u.
- n es la cantidad total de usuarios en el dataset de entrenamiento.
- m es la cantidad total de items 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 item o vise versa, y las celdas o valores de la misma contienen las calificaciones.

Matriz de calificaciones

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

Donde:

- $r_{u,i}$ es la calificación otorgada al item i por parte del usuario u.
- Cada vector fila F_u contiene todas la calificaciones realizadas por el usuario u para todos los items. Los items que aun no tiene calificación contiene el valor 0.
- Cada vector columna C_i contiene todas la calificaciones realizadas por todos los usuarios para el item i.

En el siguiente paso, se debe construir la matrix $Distancias_{u_a,u_b}$ 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

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

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 que cual utilizar. En trabajos anteriores donde se utilizo una muestra del mismo dataset, se encontró que las distancias que mejor ajustan a 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, ya con esta se obtuvieron buenos resultado en trabajos anteriores (Referencia).

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 menos es el angulo entre dos vectores mas similares son entre si. De forma contraria, cuando mayor es el angulo entre dos vectores menos similares son entre si.

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

Donde:

- au 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.
- $au \neq ub$, es decir que cada indice representa a un usuario distinto.
- I es la cantidad total de columnas de la matriz de $Calificaciones_{u,i}$.
- i el indice de una columna de la matriz de $Calificaciones_{u,i}$. Cada columna representa a un item y contiene todas las calificación realizaras por todos los usuarios sobre ese item.
- $0 <= Distancia\,Coseno_{ua,ub} <= 1$. Cuanto menor sea el valor de $Distancia\,Coseno_{u_a,u_b}$ mas similares seran los usuarios u_a y u_b.

Similitud Coseno

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

Donde:

- $0 <= Similitud\,Coseno_{ua,ub} <= 1$. Cuanto mayor sea el valor de $Similitud\,Coseno_{ua,ub}$ mas similares seran los usuarios 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 se explica el paso de inferencia o predicción para cada implementación elegida.

3.1.2. KNN basado en usuarios

En el apartado anterior se explico como calcular las matrices de $Calificaciones_{u,i}$ y $Distancias_{u_a,u_b}$.El calcula de esta matrices es parte del proceso de entrenamiento del modelo KNN. En este apartado se explicará el proceso de inferencia de una clasificación de un usuario para un item. En enfoque de K usuarios cercanos para calcular la calificación de un item se basa en la siguiente definición:

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

Donde:

- $Prediccion\,basada\,en\,usuarios\,_{u,i}$ es la predicción de la calificación del usuario u para el item i.
- $u \neq o$, es decir que cada indice representa a un usuario distinto.
- o pertenece al conjunto O de otros de usuarios. O es el conjunto de todos los usuario menos el usuario u.
- $w_{u,o}$ es la similitud entre los usuarios u y o. En nuestro casos se calcula mediante $Similitud\,Coseno_{u,o}$
- $\overline{r}_u$ es el promedio de todas las calificaciones realizadas por el usuario u.
- $r_{o,i} - \overline{r}_o$ es la diferencia entre la calificación del usuario o para el item 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 item. Algunos usuario tienden a calificar todo de forma optimista, otorgando calificaciones mas bien altas; otro usuario son mas pesimistas y tienden a poner calificaciones bajas. Al restar por la medio de calificación de cada usuario, estamos normalizando las calificaciones, haciéndolas mas o menos comparables, siendo esta una forma de disminuir este fenómeno de subjetividad al momento de calificar un item.

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

Ahora, por un tema 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 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 numero de vecinos a tener en cuenta para calcular la predicción, y mayor sera 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 traves de una optimización de hiper parámetros regida por una métricas que valida la exactitud del modelo al momento de predecir, obteniendo como resultado el K para el cual el modelo tiene el resultado mas exactos posibles.

3.1.3. KNN Item Base Prediction

Este modelo es muy similar al anterior, la diferencia radica en que la matriz de $Calificaciones_{i,u}$ tiene items como filas y usuarios como columnas, decir que es la matriz transpuesta de la matriz de $Calificaciones_{u,i}$ original. De forma la matriz de $Distancias_{i_a,i_b}$ mide las distancia entre vectores fila F_i los cuales representan a items. Dadas estas diferencias el calcula de la predicción de las calificaciones también difiere en su definición:

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

Donde:

- $Prediccion\,basada\,en\,items\,_{u,i}$ es la predicción de la calificación del usuario u para el item i.
- O es el conjunto de los vecinos cercanos o mas similares de i previamente seleccionado. o pertenece al conjunto O.
- $i \neq o$: los indices item i y o representan a items distintos.
- $w_{i,o}$ es la similitud entre los items i y o.

Finalmente la predicción, un promedio pesado de las calificaciones del usuario u para los items vecinos al item i, pesadas por la similitud de cada item o con i.

3.1.4. Ensample de modelos

Dado que contamos dos modelos basados en KNN se realizo un sample 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 del modelo KNN utilizados en este trabajo y las distintas implementaciones utilizadas. 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. One-Hot Encoding 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 ids secuenciales de usuarios e items. 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 native es realizar un one-hot encoding de la variable categórica, el 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 1 en la posición que concuerda con el valor representado y un 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 one-hot encoding obtenemos lo 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 usuario que tiene Google o Amazon ¿Que tamaño tendría el vector que representa a un solo usuario? ¿Por que usan un vector 99 % ralo para representar un valor? ¿No hay una forma mas compacta de realizar esta codificación?

La respuesta corta es si, en estos casos se utilizan Embeddings. ¿Pero que son los Embeddings y en que se diferencia 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 perdida 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 angulo 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 angulo menos 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 de los valores que son mas cercanos en términos de significado. Los días nublados y lluviosos son muy parecido entre si 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 ademas captan información que util para la clusterización de sus valores. La parte interesante es que los modelos que entrenan Embeddings captan esta información de forma automática en base a las observaciones usadas en el entrenamiento, generan estos espacios latentes llamados Embeddings.

3.2.2. Embedding Layer

En el ámbito del Deep Learning o Machine Learning se cuenta con la abstracción de **Capas** (Keras/Tensorflow) o **Módulos**(PyTorch), las cuales encapsulan el comportamiento esencial en un conjunto de bloque básicos utilizados para construir cualquier modelo. Los bloque que permiten que un modelos infiera o construya un embedding durante el entrenamiento son los bloques Embedding/EmbeddingBag en PyTorch o Embedding en Keras/Tensorflow. En ambos frameworks tienen el mismo comportamiento.

Por un lado, podemos elegir el tamaño de los vectores Embedding, el cual como ya adelantamos en un hiper-parámetro mas a optimizar. Por otro lado, debemos definir la cantidad de vectores embeding que debe contener la capa. Esta es siempre igual al número total de valores que puede tomar variable categórica.

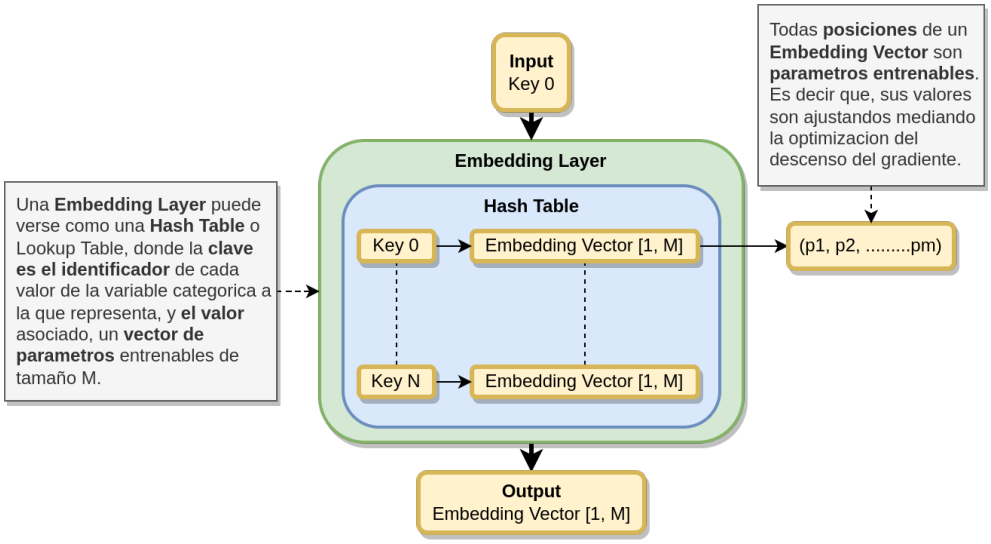
De esta forma, para crear una capa o modulo Embedding para la variable **Estado del Tiempo** podríamos crear una capa Embedding de tamaño 3, ya que cuenta con 3 posible valores, con un tamaño de vector siempre menos a 3, ya que de lo contrario tendríamos la misma dimensionalidad que tenemos al usa la codificación one-hot, con la diferencia de que una capa Embedding capta la similitud entre los valores de la variable categórica un la codificación one-hot no.

Luego el modo de funcionamiento de la capa es muy simple. Esta se puede pensar como una tabla Hash donde las claves los posible valores de la variable categórica cosificados a números y los valores son los vectores embedding. Cave aclarar que en general estos vectores son inicializados con valores aleatorio. Luego el modelo ira ajustando sos valores durante el entrenamiento.

En el froward pass, como entrada se pasa un valor codificado a números de la variable categórica. Para nuestra variable **Estado del Tiempo** podríamos codificar sus valores como sigue:

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

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



Finamente, tengamos en cuenta que el proceso de back-propagation sera encargado de ir ajustando los valores o también llamados pesos de los vectores embeddings de cuerdo a lo que requiera en la salida del modelos durante el proceso de optimización de descenso del gradiente.

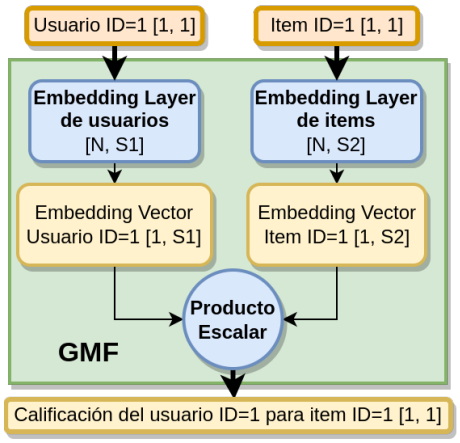
3.2.3. Arquitecturas Utilizadas

En el apartado anterior se explica uno de las componente básicos y mas usando en modelos de recomendación basados en modelo de Deep Learning. Des de qui se describirán las arquitecturas utilizadas en este trabajo.

3.2.4. General Matrix Factorization

Esta es una arquitectura clásica en sistemas de recomendación basados en filtros colaborativos. El algoritmo de factorización de matrices funciona desacoplando la matriz de interacciones usuario-item 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 en 2006. La idea principal del algoritmo es representar a usuario e items en un espacio latente de baja dimensionalidad. A partir del trabajo inicial realizado por *Funk* en 2006, se han propuesto multiples enfoque de factorización de matrices para sistemas de recomendación, siento este el modelo de mas simple y efectivo.

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



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

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

Donde:

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

En términos matriciales podemos verlo de la siguiente manera:

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

Donde:

- $U \in \mathbb{R}^{u \times f}$ es la matriz de vectores embedding de usuarios, la cual tiene tantas filas u como usuarios y el la dimensión del numero de columnas (también llamada factor latente) corresponde al tamaño seleccionado para los vectores embedding.
- $I \in \mathbb{R}^{f \times items}$ es la matriz de vectores embeddings de items, la cual tiene tantas filas como factores latente en los vectores embedding, y tantas columnas como items se tenga.
- $\tilde{R} \in \mathbb{R}^{usuarios \times items}$ es la matriz de calificaciones, donde cada fila corresponde a un usuario y columna a un item.

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 que realizar factorización de matrices con un factor latente de tamaño 1 es equivalente a un modelo de recomendación por popularidad, es decir que recomienda los items mas

populares sin tener en cuenta la personalización de las recomendaciones. Luego, a medida que vamos incrementando el tamaño del factor latente estar recomendaciones serán cada ves mas 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 comentara a empeorar. Para solucionar este se suelen agregar términos de regularización en la función de error a minimizar:

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

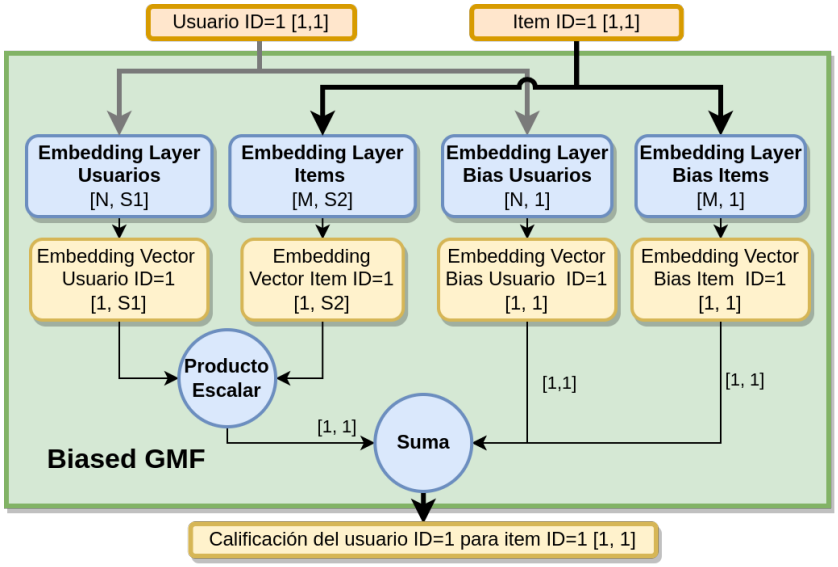
Donde:

- $\|.\|_{\mathrm{F}}$ se define como [[norma matricial]] mientras que las otras normas pueden ser matricial u otro tipo de normal dependiendo del sistema de recomendación.

3.2.5. Biased General Matrix Factorization

El modelo *GMF* de *Simon Funk* realiza recomendaciones de muy buena calidad, pero tiene como limitación que solo utilizar interacciones usuario-item que tiene 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 (Me gusta, Compras, Vistas, Favoritos, etc..). Para solucionar este nuevo problema donde es necesario usar cualquier tipo de interacción usuario item explicita o implícita se agrega un Bias o sesgo para los usuarios y otro para los items.

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



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

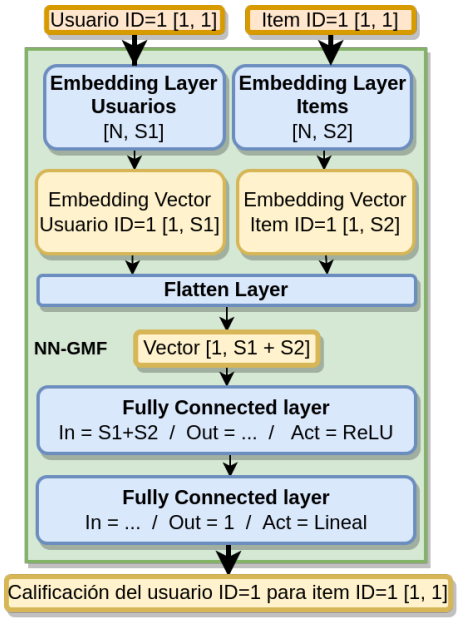
3.2.6. Neural Network Matrix Factorization

Con los enfoques anteriormente vistos (*GFM* y *Biased GFM*) dada dos matrices de baja dimensionalidad se realiza un producto escalar y se suman sesgos dependiendo de caso para calcular o inferir la calificación del usuario para un item dado. Estos modelos como ya se conto aprender los pesos o parámetros de los vectores embeddings en el proceso se entrenamiento.

El enfoque de Neural Network Matrix Factorization (NN MF) 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 su entrenamiento. En particular se remplaza el producto escalar mas 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 y en definitiva cual es la mejor función para predecir las calificaciones del usuario, como cualquier otro tipo de interacción.

Utilizar rede neuronales abre el panorama, ya que ahora podemos usar mas variable que sea relevante a nuestro problema de predicción y no solo las variable categóricas y usuario e items; aquí es donde recibe el mayor potencial de este enfoque.

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.



Entonces, como entradas tenemos los identificadores de usuarios e items en entradas independientes (Escalares). Cone sto identificadores cada *Embedding Layer* resuelve el vector embedding asociado. Acto siguiente el bloque *Flatter* toma ambos vectores y devuelve un nuevo vector el cual es la concatenación de los do anteriores. En el siguiente paso este nuevo vector es al entrada de la red multi capa, es decir que la red multi capa tendrá tanta entradas como posiciones tenga este vector. La cantidad de capas y neuronas por capas de la red so hiper-parámetros vana ir cambiando en el proceso de optimización de hiper-parámetros, pro esa cuestión no se especifican un numero de capas. Cada capa menos la ultima tiene una función de activación ReLU y la ultima capa por supuesto un activación

Lineal al igual que una regresión lineal, ya que queremos predecir las calificaciones que tienen un rango de valores reales entre 0.5 y 5. Cave aclarar que se esta pensando como segundo enfoque usar una activación *Softmax* en vez de una *Lineal* en la ultima capa. De esta manera, se podría abordar como un problema de clasificación ya que los valores de las calificaciones son números reales pero también son discretos.

3.2.7. Maquinas de Factorización

Antes de introducir el modelo *Deep Factorization Machine* vamos a comenzar explicando un de lso componente principales de este: Las *Máquinas de factorización.*

Las Maquinas de Factorización propuestas por *Steffen Rendle* en 2010,son algoritmos supervisados que se puede utilizar para tareas de clasificación, regresión y tareas de ranking como sucede en el ámbito de recomendaciones. Rápidamente se convivieron 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 *Maquina de soporte vectorial(SVM)* que utiliza un kernel polinomial.

Formalmente, si tenemos:

- $x \in \mathbb{R}^d$ es un vector de features donde cada una de sus componentes representa a una variable del dataset, siendo d la cantidad de variables de dataset (excluyendo la columna de labels). En nuestro caso $x \in \mathbb{R}^2$ ya que tenemos dos variables, usuarios e items.
- $y \in \mathbb{R}$ es la variable target a predecir, en nuestro caso es la calificación del usuario.

podemos definir el modelo para una *máquina de factorización* de grado dos de ls siguiente forma:

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

Donde:

- $\mathbf{w}_0 \in \mathbb{R}$ es el bias global.
- $\mathbf{w} \in \mathbb{R}^d$ es el peso asociado a la variable i^{th}.
- $\mathbf{V} \in \mathbb{R}^{d \times k}$ representa a un vector embeddings asociado al la variable i^{th}.
- $\mathbf{v}_i$ representa a la i^{th} fila de la matriz $\mathbf{V}$.
- k es la dimensionalidad del factor latente o tamaño de lso vectores embedding.
- $\langle \cdot, \cdot \rangle$ es el producto interno de dos vectores.
- $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ modelos la interacción entre i^{th} y j^{th} variable.

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 item y la variable j a un usuario, el tercer término es el producto escalar entre los vectores embedding de usuario u y item 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 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 insuficiencia, podemos reorganizar el tercer término del método, lo que podría reducir en gran medida el costo de cálculo, lo que lleva a una complejidad de tiempo de orden lineal $\mathcal{O}(kd)$. A continuación se describe los pasas para bajar el nivel de complejidad del método:

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

Con esta re-formulación de 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 expresion del metodo aplicada esta re-formulación queda como sigue:

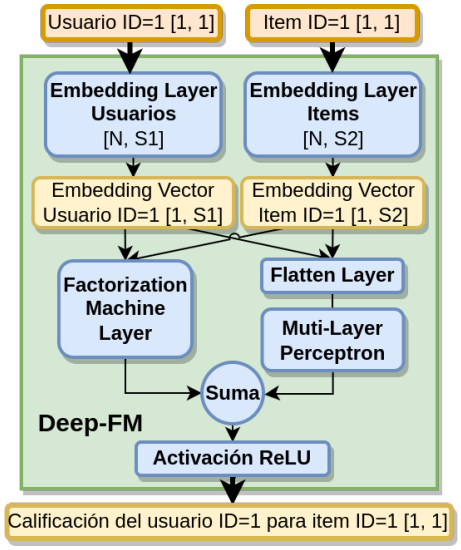
$$
\hat{y}(x) = \mathbf{w}_0 + \sum_{i=1}^{d} \mathbf{w}_i x_i + \frac{1}{2} \sum_{l=1}^{k} \big((\sum_{i=1}^{d} \mathbf{v}_{i,l} x_i)^2 - \sum_{i=1}^{d} \mathbf{v}_{i,l}^2 x_i^2 \big)
$$

3.2.8. Deep Factorization Machine

Hasta aquí, a grandes rasgos, todo los modelos vistos tratan de captar el comportamiento de las interacciones o correlación usuario-items ya sean implícita 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 Deep Factorization Machine (DeepFM) 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 de Google, el cual es un ensample de dos modelos: uno lineal, que captura las interacciones o correlaciones de alto orden y una MLP (Multi-Layer Perceptron) la cual captura correlación de mas bajo orden, o aquellas mas complejas.

A continuación de puede visualizar un diagrama de bloques de alto nivel de modelos:



Donde se puede apreciar que las entradas del modelo son las variables categóricas correspondiente a usuarios e items, como en los modelos previamente visto. dado un id de usuario y item se resuelve sus correspondientes vectores embedding, los cuales se convierten en entradas para los siguientes dos bloques. Uno de los bloques no es mas que una red neuronal multi capa con capas densa o fully connected. Por el otro lado ambos vectores se toman como entrada a la *machina de factorización*. las salida de ambos bloques son volares escalares los cuales se suman y se pasan por una activación ReLU ya que en nuestro caso las calificaciones son valores mayos a cero.

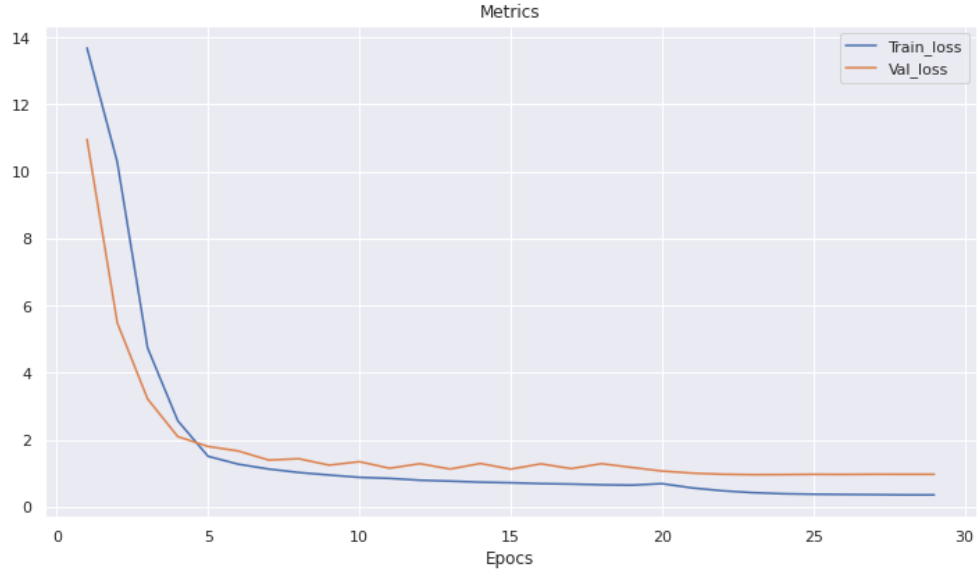
3.3. Métricas

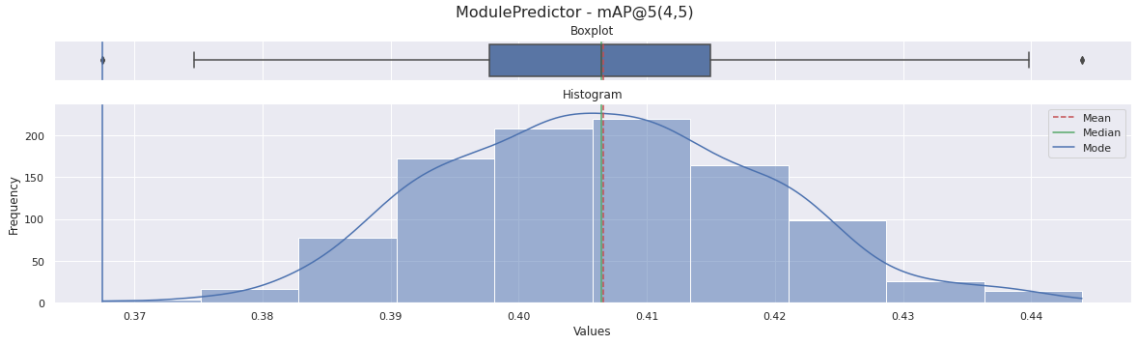
4. EXPERIMENTOS

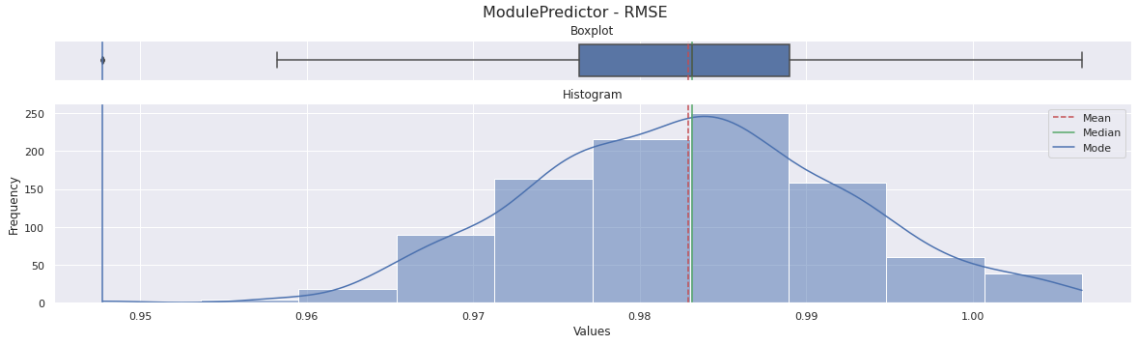
Pueden ser comparaciones entre los métodos utilizados, por ejemplo.

4.1. GFM

COMPLETAR

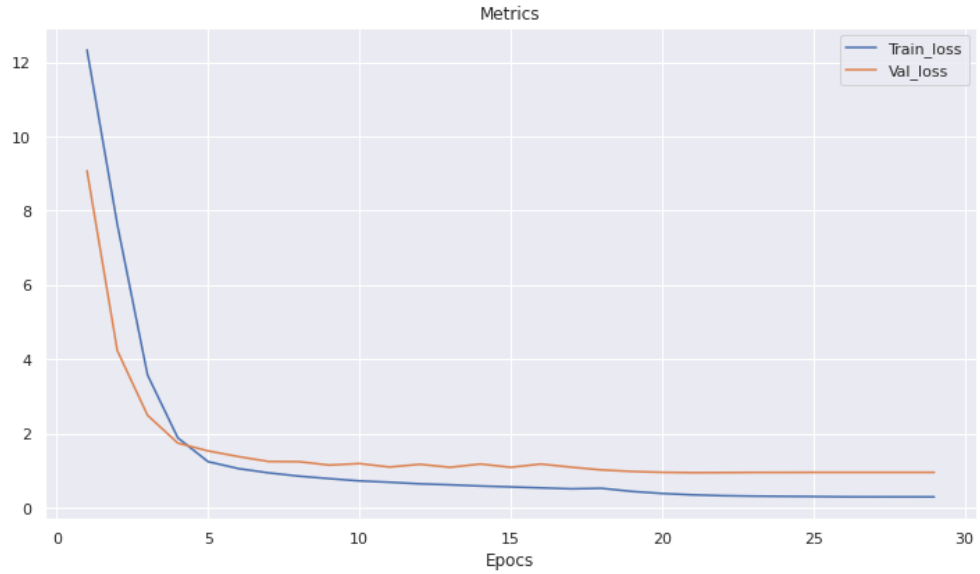


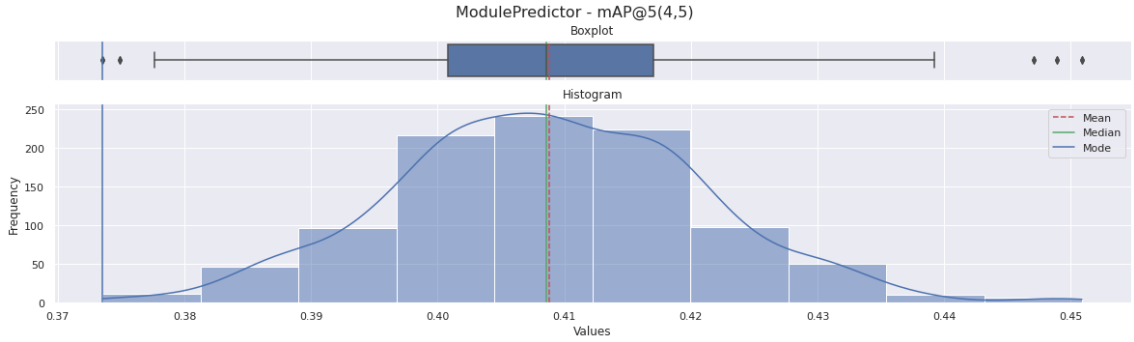


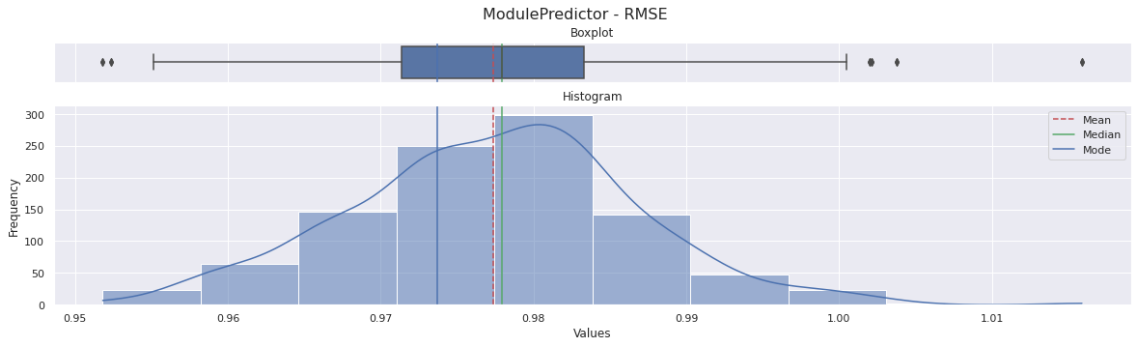


4.2. GFM Biased

COMPLETAR

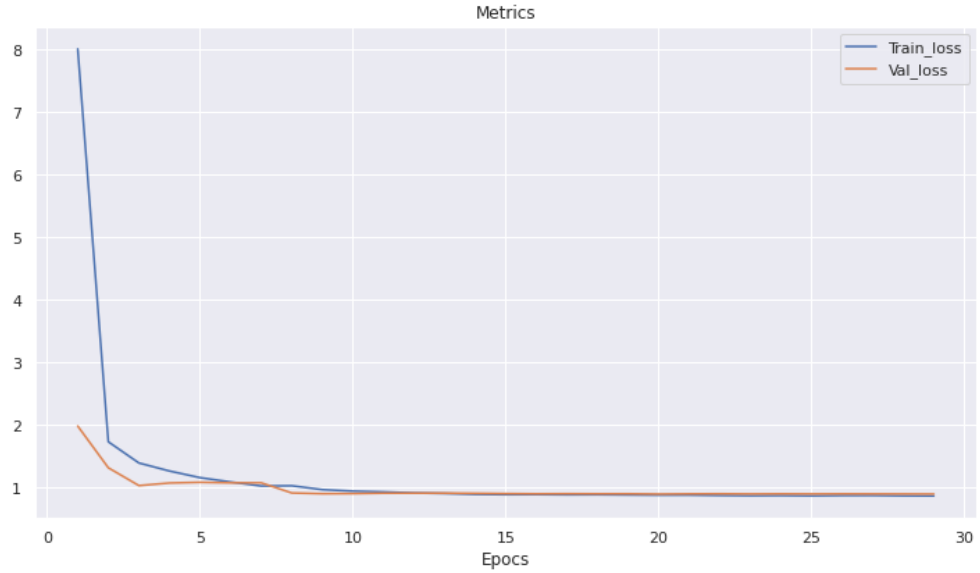


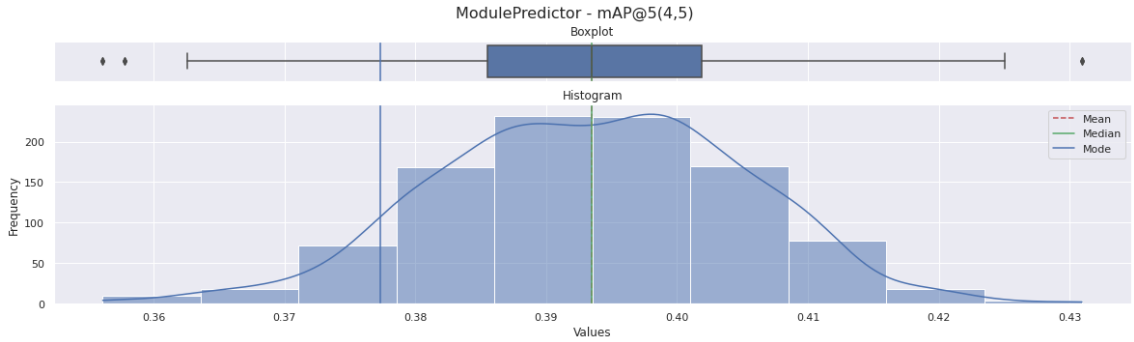


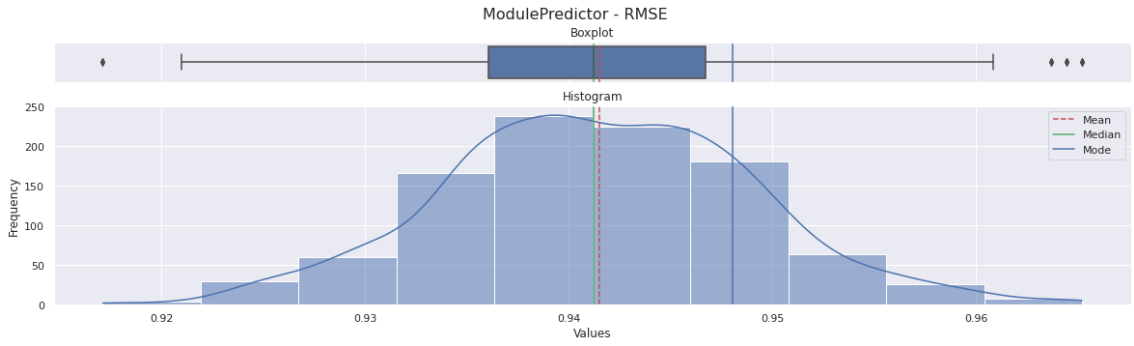


4.3. NN FM

COMPLETAR

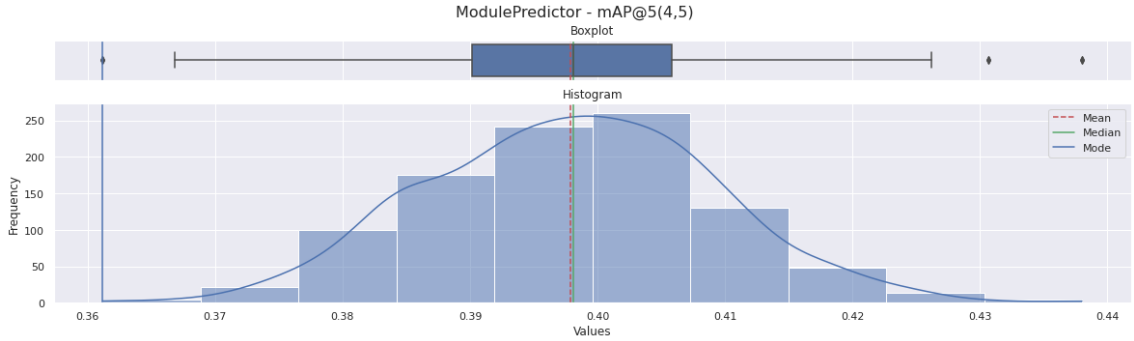


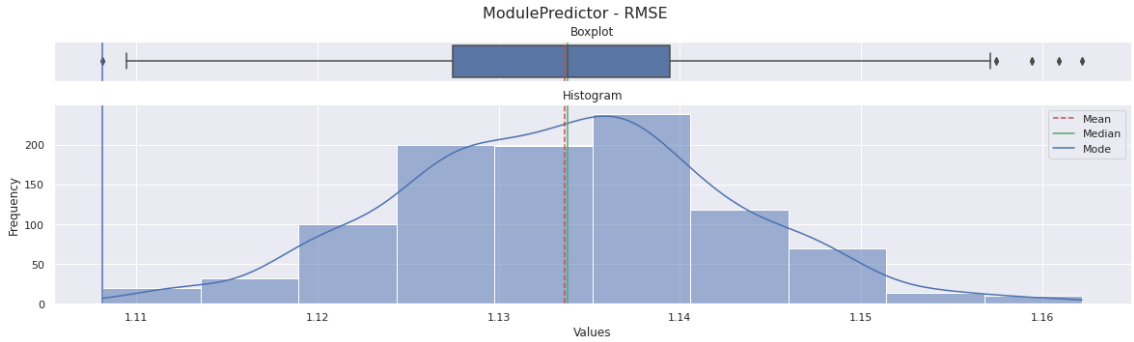




4.4. Deep FM

COMPLETAR


ModulePredictor - mAP@5(4,5)
Boxplot
Histogram
Mean
Median
Mode
Frequency
Values
0.36
0.37
0.38
0.39
0.40
0.41
0.42
0.43
0.44


ModulePredictor - RMSE
Boxplot
Histogram
Mean
Median
Mode
Frequency
Values
1.11
1.12
1.13
1.14
1.15
1.16

5. RESULTADOS

Pueden ser preliminares. Es obligatorio tener algo hecho.

6. CONCLUSIONES

Pueden ser preliminares. Si no hay mucho hecho se pueden discutir las dificultades a futuro.