# Diseño y evaluación de un recomendador de anime con MF y Two-Tower

Brian Alex Fuentes Acuña[1]

[1]Redes Neuronales, Facultad de Ingeniería — Universidad de Buenos Aires
[1]bfuentes@fi.uba.ar

## Resumen

Los **sistemas de recomendación** son fundamentales para mejorar la experiencia del usuario mediante sugerencias y tratar de retenerlo el mayor tiempo posible.

Se quiere generar un sistema de recomendación para títulos de anime (una serie o película de animación japonesa), los cuales se fueron popularizando en los últimos años.

Se va a implementar el modelo clásico de *matrix factorization* (MF) a fin de explicar un modelo más actual basado en redes neuronales con arquitectura recomendadora *Two-Tower* ($\mathrm{T^2rec}$) para predecir *ratings* y recomendar ítems.

El código fue realizado en PyTorch y se encuentra disponible en GitHub.

## 1 Introducción

Existen varios tipos de sistemas de recomendación, como los filtros colaborativos basados en contenido o usuario, hasta modelos avanzados basados en *deep learning*, en grafos, en aprendizaje por refuerzo y, recientemente, en modelos de lenguaje de gran tamaño (LLMs).

La base de datos usada es User Animelist Dataset, disponible públicamente en Kaggle. Consiste en una recopilación de distintas fuentes como el mismo Kaggle, GitHub, *web scraping* a la página MiAnimeList y similares; por lo que los datos están sucios y repetidos. Estos fueron filtrados y procesados previamente de forma independiente para MF (limitando los datos usados) y para $\mathbf{T^2rec}$; en este último se requirió el uso de Spark debido a la gran cantidad de datos.

Un sistema de recomendación puede representarse matemáticamente como una función $f$ que predice la utilidad de un ítem $i$ para un usuario $u$, denotada de la siguiente manera.

$$\hat{r}_{ui} = f(u, i; \Theta)$$

Esto se puede lograr de muchas maneras. A lo largo del tiempo se usaron métodos de clasificación sobre los *ratings*, como KNN, métodos bayesianos y muchos más, pero siempre trabajando sobre el espacio de los *ratings*. Luego se comenzó a trabajar en el espacio latente, en donde se usaban métodos como PCA y luego los métodos anteriormente mencionados, e incluso similitud coseno, para predecir los *ratings*.

Estos métodos son suficientes para un conjunto de datos típico, con pocos datos y pocas *features*. En este caso queremos predecir *ratings* para recomendar animes/ítems a un usuario en particular. Esto tiene una complejidad extra, ya que la cantidad de ítems es muchísima y muy variada: tanto en cantidad de géneros, subgéneros, episodios, autores o cualquier tipo de *feature*. Para un usuario que consume anime, una *feature* puede ser muy importante; por ello, la cantidad de información

a considerar es aún mayor. En las siguientes soluciones se consideran modelos que usan el espacio latente para las predicciones, por lo que es muy importante estimar su dimensión latente.

## 2 Matrix factorization

Para el primer caso se verá el funcionamiento de uno de los primeros modelos: un filtro colaborativo, un método del tipo *model-based*. Este modelo parte de una matriz de interacción **usuario–ítem** $R$, donde $r_{ui} \in [0, 10]$ si el usuario $u$ ha valorado el ítem $i$, $R \in \Re^{(n_i n_u)}$.

Se debe minimizar la función de costo $J(\Theta)$, donde $\Theta = [W, X]$ son los parámetros a entrenar del usuario y de los ítems, respectivamente; además, $W = [w, b]$.

Se puede modelizar la función $f$ de forma vectorial para predecir los *ratings* de la siguiente manera:

$$\hat{R} = XW^\top, \qquad X \in \mathbb{R}^{n_i \times d},\ W \in \mathbb{R}^{n_u \times d}. \quad (1)$$

En donde $n_u$, $n_i$, $n_d$ son las dimensiones de cantidad de usuarios, ítems y espacio latente, respectivamente; en general, el espacio latente es la cantidad de géneros relevantes de los ítems.

La función de costo a minimizar será el error cuadrático medio entre la predicción y la matriz de interacción.

$$\min_{W,X} J(W, X) = \min_{W,X} \left\| \left(XW^\top - R\right) \odot M \right\|^2. \quad (2)$$

Siendo $M$ una máscara que vale $m_{u,i} = 1$ si existe interacción entre usuario y *rating*, $m_{u,i} = 0$ en caso contrario.

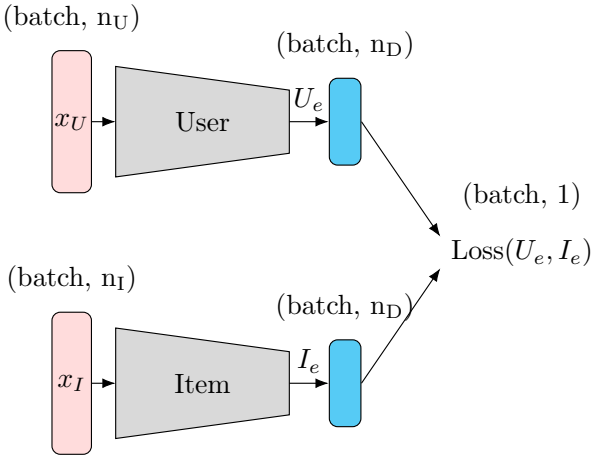
La función de costo es muy similar a una regresión lineal enmascarada, solo que esta tiene una minimización doble, tanto en la matriz de usuarios como en la de ítems, y es necesario (y muy importante) dimensionar bien el espacio latente.

## 3 Sistema de recomendación Two-Tower

El modelo MF es muy simple y requiere de una tabla de usuarios–ítems limitada a las interacciones $R$, pero no es la única información que se puede obtener de un usuario o un ítem. En general, de un usuario se puede obtener su edad, calificación por géneros, cantidad de clics y demás, dependiendo de la plataforma que recopile información. Del mismo modo, los ítems no son solo un ID: pueden tener año de estreno, géneros y subgéneros, y pueden tener autores y actores que hagan que el usuario prefiera un ítem respecto de otros. Todas estas características son ignoradas en MF; por ello surge el sistema de recomendación basado en MLP **Two-Towers**, que aún es *state of the art*. Este separa el procesamiento de usuarios e ítems en estructuras paralelas; así, puede capturar interacciones complejas y no lineales que modelos tradicionales como MF no podían capturar.

Ahora la idea es similar a MF, pero aprovechando el poder de las redes neuronales y generando un *embedding* por cada *feature* y usuario, a fin de encontrar el mejor *match* entre ambos.

$$\begin{aligned} U_e &= \mathcal{N}_{User}(x_U) \\ I_e &= \mathcal{N}_{Item}(x_I) \\ s(u, i) &= U_e^T I_e \end{aligned} \quad (3)$$




Tanto $\mathcal{N}_{User}$ como $\mathcal{N}_{User}(x_U)$ tienen la misma

arquitectura: un MLP de $[n_F, 256, 512, n_D]$, en donde la cantidad de *features* $n_f$ para usuarios e ítems son 320 y 323, respectivamente. La dimensión del *embedding* $n_D$ es de 256.

$s(u, i)$ es el vector de score generado a partir de un producto punto entre los embeddings, esto generá los rantings.

Cada capa interna está compuesta por una capa lineal seguida de una función de activación ReLU y, finalmente, *dropout* con $p = 0.5$.

Todos los pesos son inicializados con una distribución $\mathcal{N}(0, 0.01^2)$.

En términos prácticos de optimización, aunque la tarea es no convexa, se asegura que el algoritmo convergerá al menos a un punto estacionario bajo condiciones estándar de descenso de gradiente estocástico (SGD).

Se deben considerar muchos supuestos de los cuales se cumplen para asegurar la convergencia, aun así se puede considerar que llega a un punto estacionario y esta es una de las razones del exito del modelo en grandes corporaciones como YouTube.

# 4 Problema del arranque en frío

Cuando un nuevo usuario llega, no se tiene ningún tipo de información sobre el cual basarse para dar recomendaciones; es decir, no tiene historial.

Pasa lo mismo cuando aparece un ítem nuevo: al no haber interacciones con ese ítem, no va a ser recomendado hasta que las haya.

Esto es un problema frecuente en sistemas de recomendación y es crítico para atraer nuevos clientes.

MF ha "parchado" esto agregando a la predicción una ponderación con las recomendaciones promedio; es decir, $\hat{R} = p(xW^T) + (1 - p)\langle R \rangle$. Sin embargo, esto pierde generalidad según $p$, no soluciona el problema de ítems y no aprovecha los posibles *features* de los que se disponga del usuario.

Por otro lado, T$^2$rec. es más robusto ante este problema, dado que no depende solo del usuario o el ítem en específico, sino de sus *features*. Con tener esos *features* puede generar un vector de características tanto para un usuario nuevo como para un ítem nuevo.

# 5 Entrenamiento

**MF** es un modelo muy simple; de hecho, es lineal y su *loss* solo necesita de una época para llegar a resultados aceptables. También se aplica regularización L2 a los parámetros de entrenamiento usando $\lambda_{reg} = 1e{-}4$ y un *learning rate* $\nu = 1e{-}3$, y el optimizador Adam con parámetros por defecto.

Las dimensiones las matrices de parametros son $n_d = 15$ y tambien $n_u = 1001$, $n_i = 3096$.

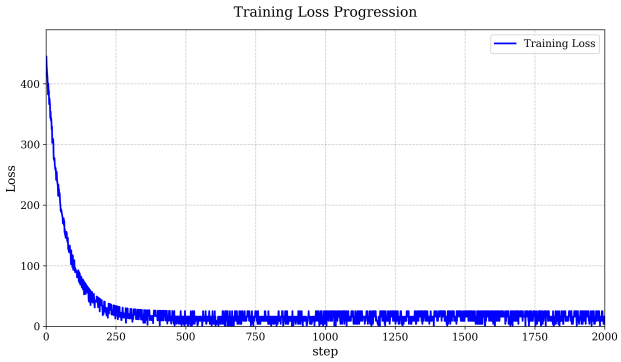



Figure 1: Loss a lo largo de una epoca

**T$^2$rec.** requiere de una función de costo un poco mas elavorada, debido a problemas presentados con el set de datos, es mucho mas grande respecto al anterior, esto por que ya no es necesario una matriz sparce y el modelo puede ser tan complejo como uno quiera.

El costo utilizado es una ponderacion el MSE ponderado por la frecuencia de los ratings en todo el set de entrenamiento, esto debido al desbalance en los ratings del set de datos, introduce sesgo, como se ve en la figura 2 la proporcion de ratings mayor a 7 son por mucho la mas dominante, si no se tratara el modelo aprenderia a devolver un valor medio entre esos ratings para minimizar la

funcion de costo.

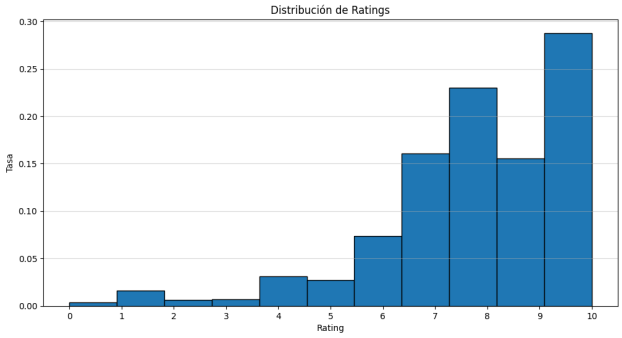



Figure 2: Distribución de ratings en el conjunto de datos

Se usa entonces la primer funcion de costo.

$$\mathcal{L}_1(\hat{y}, y) = \mathbb{E}\big[w(y)\,|\hat{y} - y|^2\big]$$

En la implementacion esto no es mas que la siguiente función de costo

$$\mathcal{L}_1(Ue, Ie) = \frac{1}{w_{freq}}.MSE(s, r_{u,i}) \qquad (4)$$

Tal que $\frac{1}{w_{freq}}$ es la inversa de la frecuencia del rating en el set de entrenamiento, multiplicar por la inversa para cada rating evitar el overfitting en caso de que los ratings no esten balanceados.

Para miticar las diferencias groseras en la prediccion de ratings se usa tambien **MarginLoss** el cual castiga la direncia de aquellos ratings conecutivos que se alejen mucho de los verdaderos, esto mediante un treshold **m**.

$$\mathcal{L}(.)_2 = \begin{cases} \max\big(0,\, m - (\hat{y}_i - \hat{y}_j)\big), & \text{si } y_i - y_j > m \\ 0, & \text{en otro caso.} \end{cases}$$

Finalmente, se suman los dos costos

$$\mathcal{L}(\cdot) = \lambda_1\,\mathcal{L}_1(\hat{y}, y) + \lambda_2\,\mathcal{L}_2(\hat{y}_i, \hat{y}_j, y_i, y_j) \qquad (5)$$

Para el entrenamiento se scalan todos los features utilizando MinMaxScaler, se puede ver mas en detalle este paso en el apendice A.

Los hiper parametros usados son un margen $m = 0.5$, optimizador Adam con $lr = 1x10^{-3}$ y 32 epocas, se vio que a partir de la quinta epoca no hay mejora apreciable.

# 6 Resultados

Los modelo fueron evaluados en principio con metricas de precision numerica las cuales miden las diferncia entre las predicciones y los targets.

| Metrica | MF | T²rec. |
|---|---|---|
| MSE | 13.727 | 0.219 |
| RMSE | 3.705 | 0.058 |
| MAE | 3.530 | 0.051 |

Estos valores son considerando que los scores $r$ se encuentran en el rango 0-10.

Es de notar la mejora resultante, teniendo en cuenta que ahora se puede poner ma datos, es un modelo mas complejo y no es necesario reentrenar el modelo para un nuevo usuario.

No obstante, estas metricas no evaluan la calidad de los rankings ni el rendimiento de **T²rec.** por ello tambien se evaluan con metricas basados en clasificacion/ranking (para ver en detalle las metricas ver Apendice B).

| Metrica | T²rec. |
|---|---|
| NDCG@10 | 0.998 |
| Precision@10 | 0.639 |
| Recall@10 | 0.889 |
| HitRate@10 | 0.999 |

Cabe recalcar que estas son pruebas *offline*; en la industria quedan más pruebas por realizar para terminar de desplegar el modelo: pruebas con personas reales que testeen las recomendaciones, *A/B testing*, por ejemplo.

Existen otros métodos de testeo, como asignar un usuario a un agente cuyo "cerebro" sea GPT, Gemini, Claude, etc. Si bien estos son relativamente más baratos que contratar verdaderos testers, según mi propia interpretación, sufren de

falta de generalidad: van a generar interacciones que no necesariamente son las de un humano. Sin embargo, es una buena primera aproximación para una PoC que no excluye las debidas pruebas.

# 7 Conclusión

En este trabajo se presentó el diseño y la implementación de un sistema de recomendación de títulos de anime, comparando un enfoque clásico basado en *matrix factorization* (MF) con un modelo moderno de redes neuronales tipo *Two-Tower* ($T^2$rec). Mientras que MF modela la interacción usuario–ítem únicamente a partir de la matriz de *ratings*, el enfoque Two-Tower permite incorporar *features* ricas tanto del usuario como del ítem, aprendiendo representaciones latentes (*embeddings*) que capturan relaciones no lineales y mejoran la capacidad de generalización.

Los resultados obtenidos muestran una mejora marcada al pasar de MF a $T^2$rec en métricas numéricas (MSE, RMSE y MAE), coherente con el aumento de expresividad del modelo y con el aprovechamiento de información adicional. Asimismo, al evaluar el rendimiento desde la perspectiva de ranking con métricas como NDCG@10, Precision@10, Recall@10 y HitRate@10, el modelo Two-Tower exhibe un desempeño sólido en escenarios *offline*, sugiriendo que puede producir recomendaciones relevantes en el *top-k*.

Finalmente, se discutió el problema del *arranque en frío*, donde los modelos basados solo en interacciones (como MF) tienden a fallar ante nuevos usuarios o ítems. En contraste, $T^2$rec es naturalmente más robusto, ya que puede generar representaciones a partir de *features* aun cuando no existan interacciones previas.

Como trabajo futuro, además de refinar el preprocesamiento y la selección de *features*, sería necesario validar el sistema con evaluaciones *online* (por ejemplo, A/B testing) para medir impacto real en usuarios y asegurar que las mejoras *offline* se traduzcan en mejor experiencia y engagement.

Tambien, se puede modificar la funcion de decision $S(u, i)$ el cual recibe embeddings, existen metodos que pruebas con ver el espacio lantente de los items y segmentar a partir de ellos, podrian ser parte incluso de un modelo mas grande basado en transformers de los cuales estan siendo explorados en los ultimos años debido a la gran cantidad de datos que hay disponibles.

# 8 Apéndice

## 8.1 Apéndice A

Los datos de ítems tienen los siguientes *features*: *"score"*, *"episodes"*, *"year"*, seguido de 320 géneros conformados por géneros principales y géneros más específicos.

Los géneros tienen un uno si el ítem pertenece a ese género y cero en caso contrario (*one-hot encoding*), por lo que no requiere ningún tratamiento adicional. Por otro lado, los *features year* y *episodes* presentan un problema, dado que su rango es de 1907 a 2026 y de 0 a 3057, respectivamente, y haría que el modelo *overfittee* rápidamente.

Estos atributos pueden ser muy importantes a la hora de calificar un ítem (anime), por lo que no se descartan y se opta por usar MinMaxScaler, decidiendo cuáles van a ser los valores máximos y mínimos a utilizar.

| **Feature** | **Rango original** | **Rango usado** |
|---|---|---|
| year | $1907 - 2026$ | $1900 - 2030$ |
| episodes | $0 - 3057$ | $0 - 4000$ |

Esto quiere decir que el modelo se puede ir actualizando con nuevos items hasta el 2030 en donde va a ser necesario volver a transforar todos los items y además que no se va a considerar items con mas de 4000 episodios, son muy pocos los que hay en ese extremo pero son muy populares.

Por otro lado el rango de los *score* es de 0-10 el cual no es muy grosero pero por ser el target se lo va a dejar sin cambios, este será scalado

durante el entrenamiento en el rango que se crea conveniente.

## 8.2 Apéndice B

La precisión y el recall son las métricas más populares y probablemente las más intuitivas que puedes calcular. El recall mide qué porcentaje de los ítems que le gustaron al usuario fueron recomendados, mientras que la precisión calcula qué porcentaje de los ítems recomendados formaban parte de los ítems que le gustaron al usuario.

$$\text{Precision}_k = \frac{|\{\text{Liked Items}\}| \cap |\{\text{Rec. Items}\}|}{k}$$

$$\text{Recall}_k = \frac{|\{\text{Liked Items}\}| \cap |\{\text{Rec. Items}\}|}{|\{\text{Liked Items}\}|}$$

**Ganancia acumulada de descuento normalizada (nDCG)** las anteriores métricas que asumen que la relevancia del elemento es binaria, es decir, o bien algo es relevante para el usuario o no. Pero muchas veces, en aplicaciones reales, no solo sabemos si un elemento es relevante, sino que también tenemos información sobre cuán relevante es para el usuario. el objetivo será recomendar elementos que tengan un alto grado de relevancia para el usuario en posiciones superiores en la salida de recomendación. Para entender si estamos logrando eso, necesitamos una métrica como nDCG.

Sea $S_i$ Sé la puntuación de relevancia del elemento en la posición $i$ en nuestra lista de items recomendados. Entonces el DCG se calcula como:

$$DCG_k = \sum_{i=1}^{k} \frac{s_i}{\log_2(i+1)}$$

Luego se normaliza dividiendo por la mejor puntuación posible de DCG.

$$nDCG_k = \frac{DCG_k}{IDCG_k}$$

**Hit Rate (Hitrate)** El hitrate mide si el sistema logró recomendar al menos un ítem relevante dentro del top-k. Para cada usuario vale 1 si hubo al menos un acierto y 0 si no. El valor final es el promedio entre todos los usuarios.

$$\text{HitRate}_k(u) = \begin{cases} 1 & \text{si } L_u \cap R_{u,k} \neq \varnothing, \\ 0 & \text{si } L_u \cap R_{u,k} = \varnothing. \end{cases}$$

Luego se calcula la metrica global,

$$\text{HitRate}_k = \frac{1}{|U|} \sum_{u \in U} \text{HitRate}_k(u)$$

## 8.3 Apéndice C

### Descripción del dataset y preprocesamiento (T²rec.)

#### 8.3.1 Datos Iniciales

**Archivos Fuente:**

- `animes.csv`:
  - `animeID`, `title`, `type`, `year`
  - `score`, `episodes`, `genres`, `genres_detailed`
  - `mal_url`, `sequel`, `image_url`, `alternative_title`
- `ratings.csv`:
  - `userID`, `animeID`, `rating`

#### 8.3.2 Procesamiento Realizado

**1. Limpieza Inicial:**

- Eliminación de columnas no relevantes: `mal_url`, `image_url`, `alternative_title`
- Conversión de tipos de datos (años, puntuaciones)
- Manejo de valores nulos

**2. Procesamiento de Géneros:**

- Unión de `genres` y `genres_detailed`
- Filtrado de géneros poco frecuentes (percentil 90)
- Normalización (minúsculas, eliminación de espacios)

**3. Filtrado de Usuarios:**

- Eliminación de usuarios con <8 o >117 calificaciones
- Conservación de usuarios con rango de calificaciones $\geq 3$

#### 8.3.3 Conjuntos Finales

**1. Fuatures de Usuarios:**

- `userID`, `n_ratings`, `avg_rating`
- Puntuaciones normalizadas por género
- Preferencias de género ponderadas

**2. Features de Animes:**

- `animeID`, `title`, `type`, `year`
- `score`, `episodes`
- Vector de géneros (one-hot encoding)

**3. Interacciones :**

- `userID`, `animeID`, `rating`
- Conjuntos de entrenamiento (80%) y prueba (20%)

#### 8.3.4 Optimizaciones

- Uso de Spark para procesamiento distribuido
- Persistencia en memoria intermedia
- Muestreo estratificado para mantener distribución de calificaciones

## References


[1] Giorgi. *Offline Metrics for Recommender Systems.* `https://giorgi.tech/blog/offline-metrics-for-recommender-systems/`.

[2] Aryan Jadon & Avinash Patil. *A Comprehensive Survey of Evaluation Techniques for Recommendation Systems.* `https://arxiv.org/html/2312.16015v2`

[3] *Towards a Theoretical Understanding of Two-Stage Recommender Systems* `https://arxiv.org/html/2403.00802v1`.