Sistemas Recomendadores: Informe Tarea 1

Tomás Trincado, Joaquín Peralta

13/05/2025

1. Introducción

El objetivo de este informe es evaluar y comparar distintos algoritmos de recomendación aplicados a un conjunto de datos de calificaciones de anime. Para ello, se aplicaron diversas técnicas de aprendizaje automático con el fin de predecir las calificaciones que los usuarios otorgarían a diferentes animes,como también, se propusieron listas de recomendaciones personalizadas. Adicionalmente, se analiza el rendimiento de estos algoritmos utilizando diversas métricas de evaluación, como precisión, eficiencia en el procesamiento, uso de memoria, tiempo de ejecución, novedad y diversidad de las recomendaciones, y la calidad general de cada una de ellas.

2. Análisis de datos

En esta sección se presenta un análisis exploratorio de los datos, describiendo su estructura, estadísticas básicas, y aspectos relevantes para los métodos de recomendación.

Este análisis incluye la distribución de ratings, comportamiento de los usuarios y popularidad de los ítems.

Partimos observando la distribución de los items según fueron rateados y de su limpieza

Métrica	**Train File**	**Validation File**
Filas totales	27,399	2,311
Ítems calificados	22,237	1,865
Ítems sin calificar (-1)	5,162	446
Porcentaje sin calificar	18.84 %	19.30 %
Tamaño antes de la limpieza	(27399, 3)	(2311, 3)
Tamaño después de la limpieza	(22237, 3)	(1865, 3)

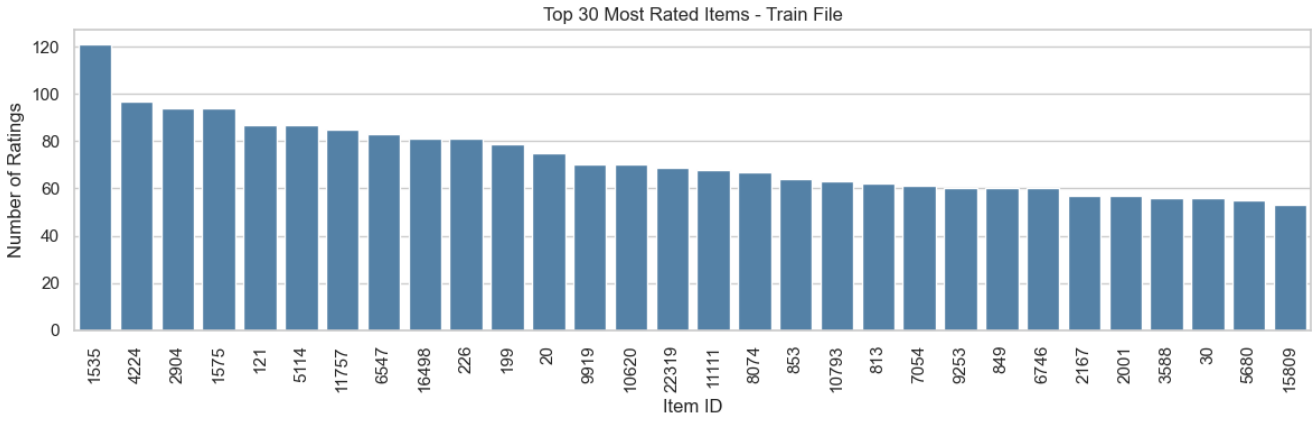
Archivo	**Ítem**	**Calificaciones Faltantes**	**Total de Apariciones**
Train	**Death Note**	25	146
	Death Note	121	146
File	**Death Note**	4	10
	Code Geass	11	12

2.1. Estadísticas de los ítems

Ahora describiremos la cantidad de ratings por ítem, identificando ítems populares (con alta cantidad de interacciones) y aquellos que presentan escasa participación. Se analizan también medidas de centralidad y dispersión.

Train File	**Valor**
Total de ítems únicos calificados	3543
Promedio de calificaciones por ítem	6.28
Mediana de calificaciones por ítem	3.0
Máxima cantidad de calificaciones para un solo ítem	121

Cuadro 1: Estadísticas de los ítems - Conjunto de Entrenamiento



Validation File	**Valor**
Total de ítems únicos calificados	1233
Promedio de calificaciones por ítem	1.51
Mediana de calificaciones por ítem	1.0
Máxima cantidad de calificaciones para un solo ítem	11

Cuadro 2: Estadísticas de los ítems - Conjunto de Prueba

2.2. Densidad del dataset (ítems por usuario)

Calculamos la densidad del dataset considerando la proporción de ítems que cada usuario ha calificado respecto al total disponible. Esta métrica permite evaluar el nivel de actividad y cobertura del sistema de recomendación desde la perspectiva del usuario.

Métrica	**Valor**
Promedio de ítems calificados por usuario	1.33
Total de ítems únicos	3543
Densidad (%)	0.04 %

Cuadro 3: Densidad del dataset de entrenamiento (ítems por usuario)

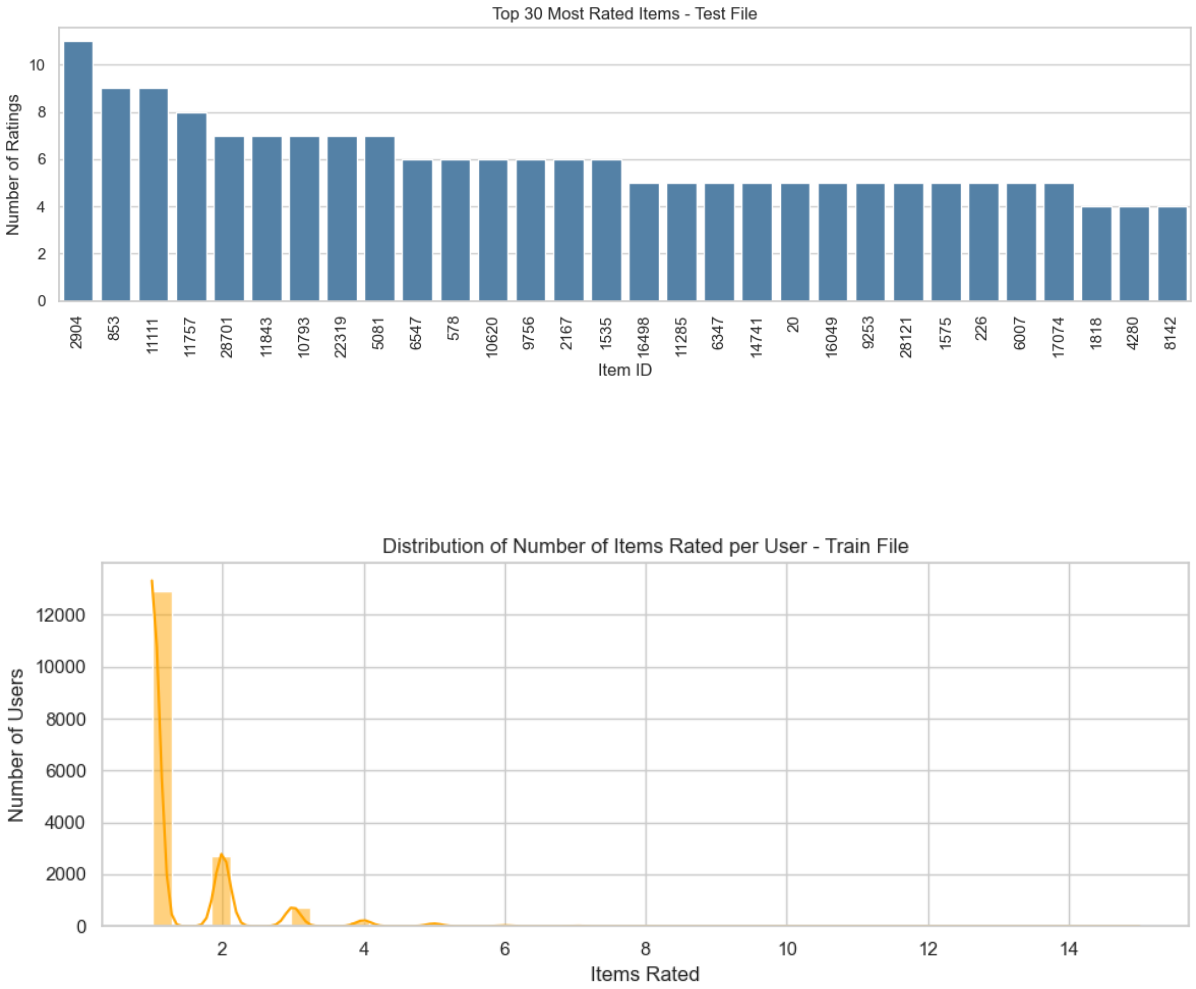


Figura 1: Enter Caption

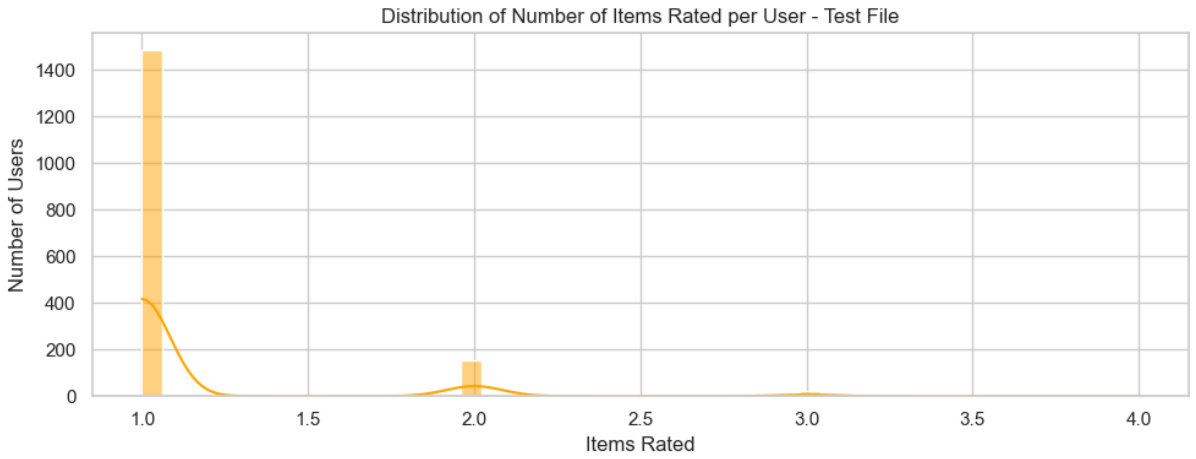


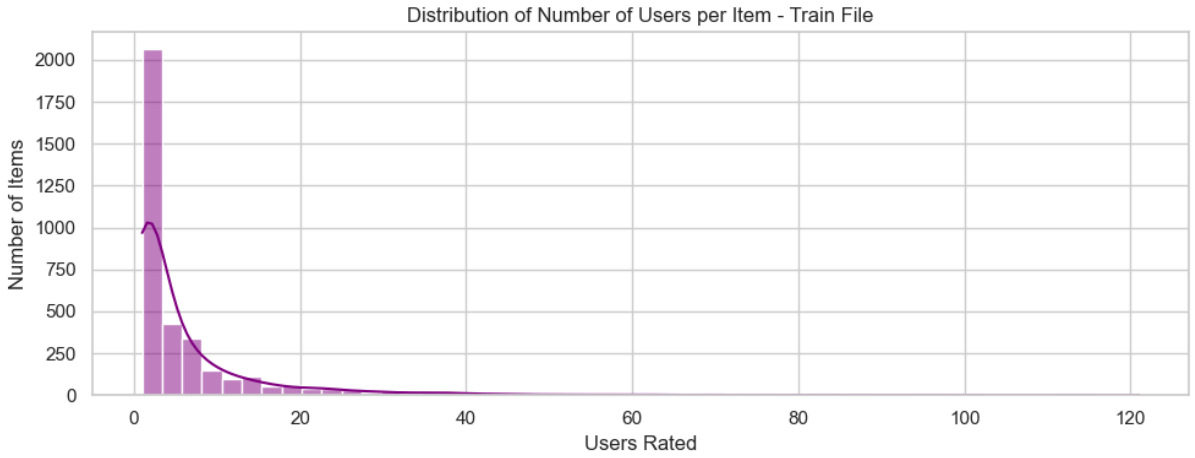
Figura 2: Enter Caption

Métrica	Valor
Promedio de ítems calificados por usuario	1.12
Total de ítems únicos	1233
Densidad (%)	0.09 %

Cuadro 4: Densidad del dataset de prueba (ítems por usuario)

2.3. Densidad del dataset (usuarios por ítem)

Análogamente, cuanto han calificado cada ítem los usuarios.

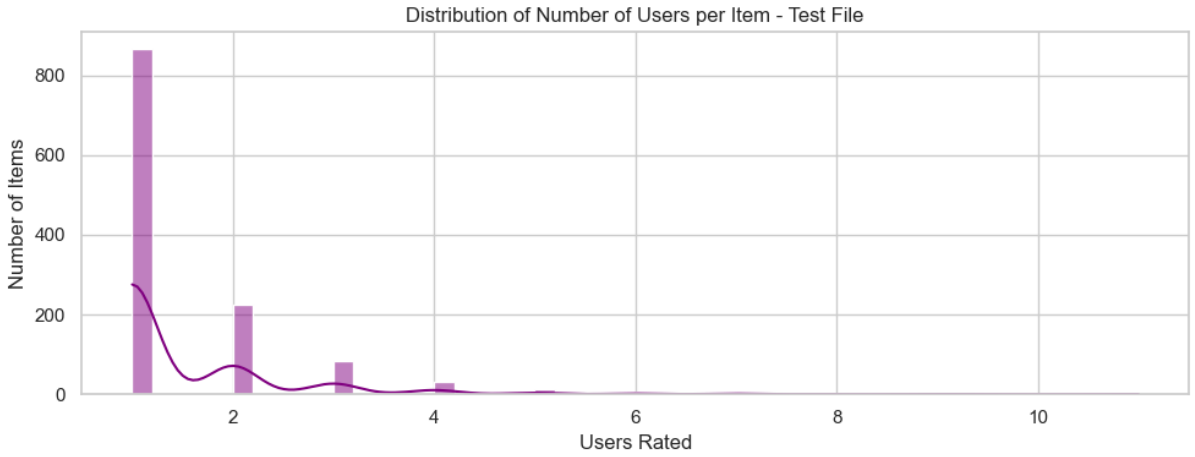


Métrica	Valor
Promedio de usuarios por ítem	6.28
Total de usuarios únicos	16710
Densidad (%)	0.04 %

Cuadro 5: Densidad del dataset de entrenamiento (usuarios por ítem)

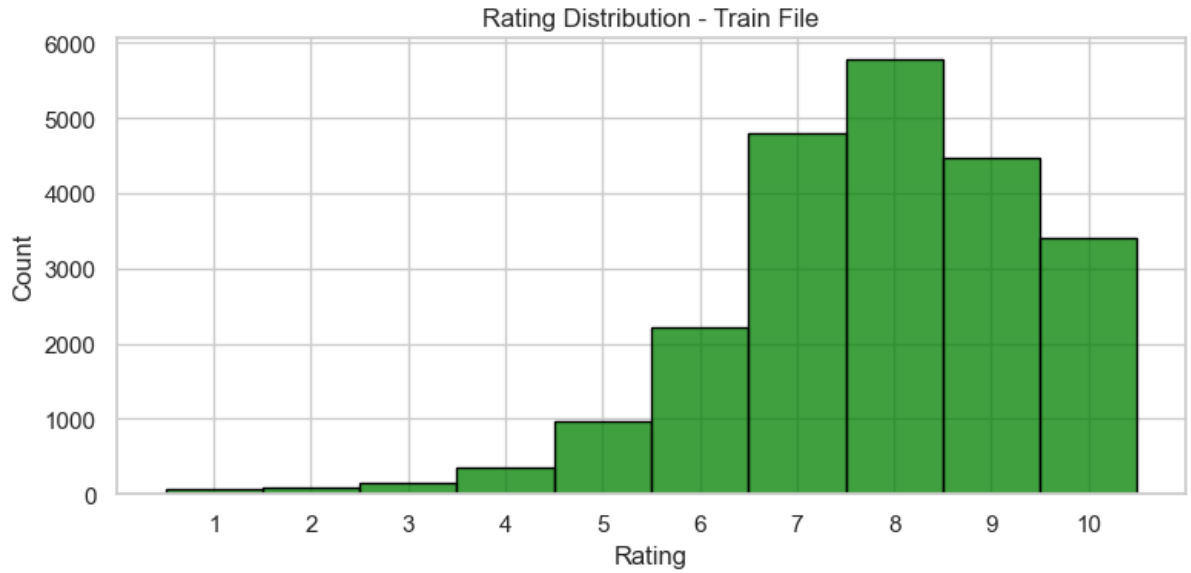
Métrica	Valor
Promedio de usuarios por ítem	1.51
Total de usuarios únicos	1661
Densidad (%)	0.09 %

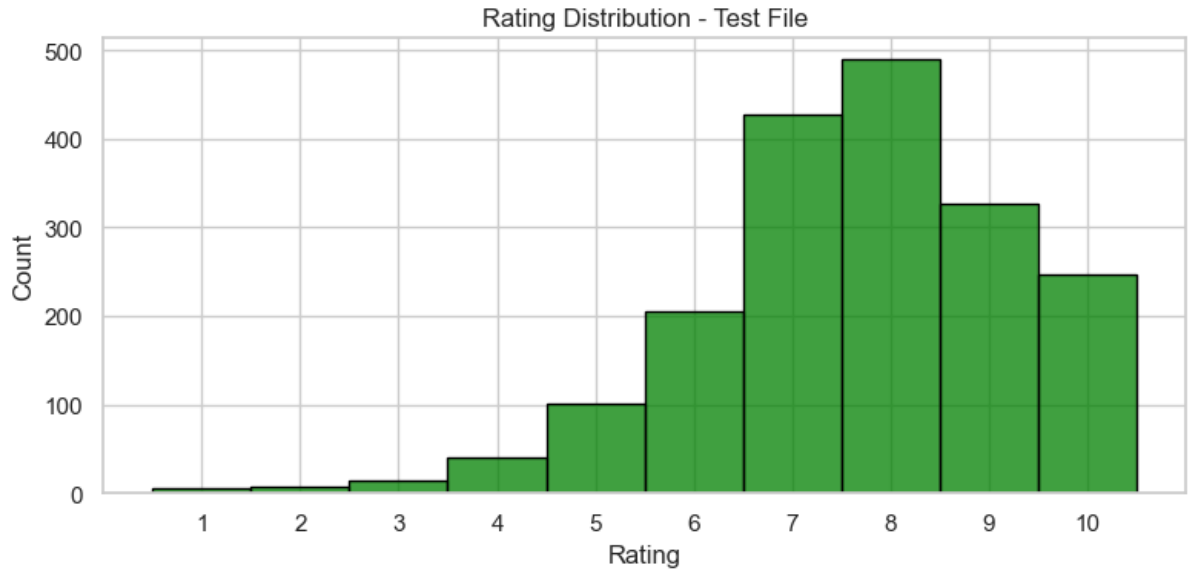
Cuadro 6: Densidad del dataset de prueba (usuarios por ítem)



2.4. Figura 1: Distribución de ratings

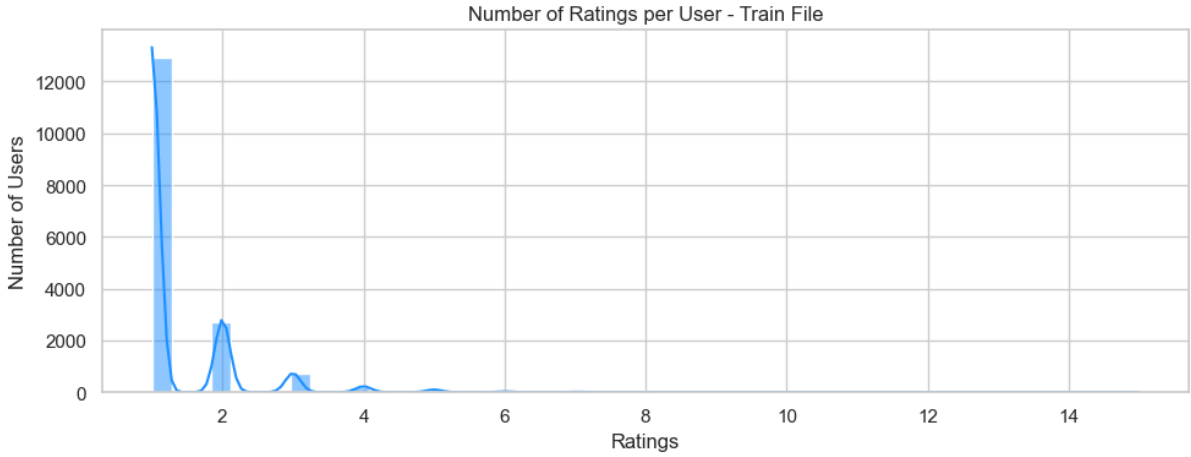
Mostramos como se comporta la distribución de las puntuaciones asignadas por los usuarios:

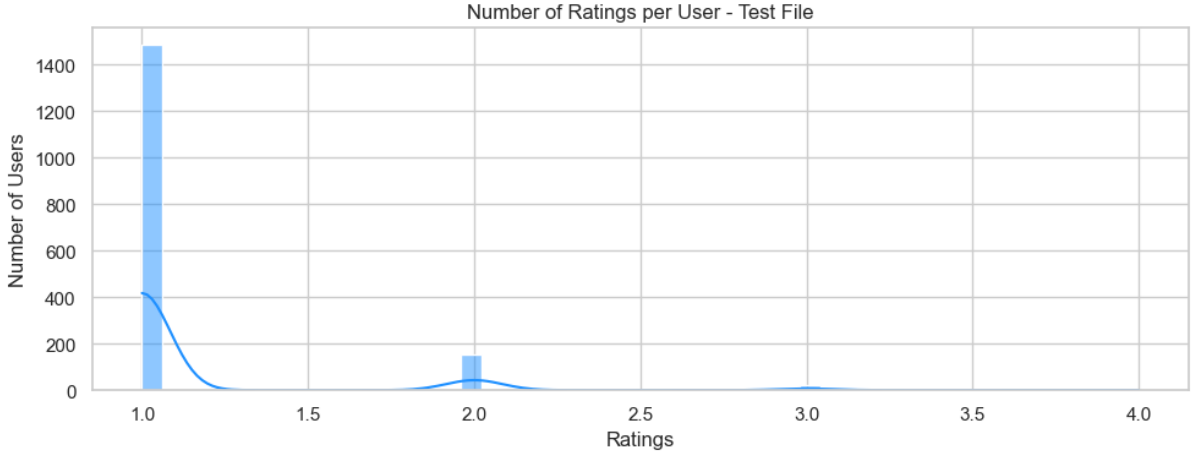




2.5. Figura 2: Cantidad de ratings vs número de usuarios

Esta figura muestra la relación entre la cantidad de ratings realizados y el número de usuarios, permitiendo visualizar cómo se distribuye la participación. Puede ayudar a identificar usuarios extremadamente activos o poco comprometidos.





3. Métodos De Predicción

3.1. User-based collaborative filtering

Para empezar, probamos distintas combinaciones de parámetros de número de vecinos y de tipo de correlación.

Primera prueba: $k = 5$, Pearson.

Una de las primeras cosas que notamos al analizar las predicciones individuales es que, para la mayoría de los casos, no se logra hacer una predicción válida. Por ejemplo:

```
# Input
myUserKnn.predict("68084", "6325")

# Output
Prediction(uid='68084', iid='6325', r_ui=None, est=6.1656,
details={'was_impossible': True, 'reason': 'Not enough neighbors.'})
```

Para entender por qué pasa esto, analizamos cuánta información tenemos tanto del ítem como del usuario:

- Cantidad de usuarios que calificaron el ítem `6325`: 20
- Cantidad de ítems calificados por el usuario `68084`: 3

Con esto, podemos observar que no hay suficiente información sobre los gustos del usuario ni sobre la popularidad del ítem como para que KNN encuentre vecinos confiables.

A pesar de esto, generamos la matriz completa de predicciones usando el anti-testset y filtramos sólo aquellas predicciones que sí fueron posibles. Acá encontramos que todas las predicciones válidas se realizaron utilizando solamente 1 vecino, lo cual refuerza aún más la escasez de datos:

```
[Prediction(uid='44929', iid='6', est=7.0, details={'actual_k': 1}),
 Prediction(uid='44929', iid='8314', est=6.0, details={'actual_k': 1}),
 Prediction(uid='44929', iid='18179', est=8.0, details={'actual_k': 1}),
 Prediction(uid='52016', iid='8675', est=8.0, details={'actual_k': 1}),
 Prediction(uid='39921', iid='11319', est=1, details={'actual_k': 1})]
```

Así, al calcular el RMSE notamos que este fue sorprendentemente bajo:

RMSE: 0.0009

Pero, en este caso, esta no una medida fiable. El RMSE sólo considera las predicciones que fueron posibles. Como la mayoría de los casos no pudieron ser predichos, esta métrica no representa el verdadero rendimiento del modelo.

Para realizar un análisis de mayor claridad, armamos esta tabla que muestra el contraste entre predicciones totales y válidas:

Métrica	Valor
Predicciones válidas	20
Total de predicciones posibles	78,455,796
Tasa de predicciones válidas	$2{,}55 \times 10^{-7}$

Cuadro 7: Proporción de predicciones válidas con $k = 5$, Pearson

Segunda prueba: $k = 50$, Pearson

Cambiamos a 50 vecinos, esperando que con un k más amplio el modelo logre más predicciones. El resultado fue prácticamente el mismo: un RMSE bajo y un número casi nulo de predicciones posibles. Se iteró probando incluso hasta con 200 vecinos, pero aun así el resultado fue el mismo.

Tercera prueba: $k = 20$, Cosine

Finalmente, cambiamos la métrica de similitud a Cosine con 20 vecinos. Si bien se modificaron algunos detalles menores en las predicciones válidas, la situación general se mantuvo. El modelo simplemente no cuenta con suficiente información como para hacer predicciones razonables.

En conclusión, UserKNN depende fuertemente de una matriz de ratings densa para funcionar bien. En nuestro caso, la cantidad de calificaciones por usuario e ítem es extremadamente baja, lo que hace que el modelo no encuentre suficientes vecinos para hacer recomendaciones válidas.

La implementación del modelo fue de una baja dificultad, aunque la generación de las predicciones tomó bastante tiempo, con un promedio de 13 minutos por cada combinación de parámetros, lo cual es considerable para datasets grandes, sobre todo si se quiere hacer una validación cruzada más extensa o afinar hiperparámetros.

Por último, consideramos este modelo como costoso en memoria. Esto se debe a que se necesita almacenar la matriz de similitud completa entre usuarios, lo cual escala cuadráticamente. En nuestro caso, el consumo fue manejable, pero podría volverse un problema a mayor escala, en especial con matrices más completas y, por ende, realizando mayores cómputos para el cálculo de predicciones.

3.2. Item-based collaborative filtering

Similarmente a userKNN, probamos distintos casos para ver cómo se comporta el modelo.

Primera prueba: $k = 7$, Pearson

Inmediatamente, notamos que la situación que ocurrió en userKNN se repite. Obtenemos un RMSE bajo: 0.0162, pero nuevamente lo tomamos como una métrica no fiable.

Métrica	Valor
Predicciones válidas	1097
Total de predicciones posibles	59,177,752
Tasa de predicciones válidas	$1{,}8537 \times 10^{-5}$

Cuadro 8: Proporción de predicciones válidas con $k = 7$, Pearson

Por otro lado, calculamos el top 10 para recomendar y obtenemos las siguientes métricas:

Métrica	Valor
MAP@10	0.0010
NDCG@10	0.0019
Recall@10	0.0037
Diversity	1.0000
Novelty	8.4038

Cuadro 9: Métricas de evaluación para el modelo ItemKNN (k=7, Pearson)

En general, las métricas nos confirman que el modelo no está logrando generar buenas recomendaciones. Tiene una precisión y recall extremadamente bajos, lo que indica que no acierta lo que le gusta al usuario, ni siquiera en parte. La diversidad y novedad altas pueden parecer positivas a primera vista, pero en este contexto reflejan un modelo que no tiene suficiente información para hacer recomendaciones sólidas, y termina lanzando ítems poco frecuentes y muy variados.

Esto se alinea completamente con lo que observamos antes: el dataset es demasiado disperso, y faltan ratings para que ItemKNN funcione bien.

Para corroborar que no es coincidencia, realizamos otra prueba:

Segunda prueba: $k = 5$, Cosine

Métrica	Valor
Predicciones válidas	373,449
Total de predicciones posibles	59,177,752
Tasa de predicciones válidas	$4{,}4 \times 10^{-3}$

Cuadro 10: Proporción de predicciones válidas con $k = 7$, Pearson

Notamos que el total de predicciones válidas aumentó considerablemente. A pesar de seguir siendo muy bajo en comparación al número total de predicciones posibles, notamos que cosine aumentó las predicciones válidas en aprox un x370, con un nuevo RMSE de 0.2124. A pesar de ser más alto, vemos que nuestra hipótesis está siendo aceptada, y a pesar de tener un "peor rendimiento", este es más fiel a lo que en realidad está ocurriendo.

Métrica	Valor
MAP@10	0.0014
NDCG@10	0.0027
Recall@10	0.0052
Diversity	0.9995
Novelty	7.9690

Cuadro 11: Métricas de evaluación para el modelo ItemKNN (k=20, Cosine)

Con valores muy parecidos a la prueba anterior, concluimos que efectivamente para que estos modelos tengan un buen rendimiento se debe tener una matriz más completa para poder realizar mejores predicciones. Se repite lo mismo para la memoria que en userKNN y los tiempos de ejecución fueron 8 y 12 minutos.

3.3. Random

Acerca de este modelo se tiene que toda recomendación se realiza de forma aleatoria. Esto nos provee con los siguientes resultados:

Metric	Value
MAP@10	0.0013
NDCG@10	0.0026
Recall@10	0.0045
Diversity	0.9998
Novelty	7.9534

Cuadro 12: Métricas de evaluación para el modelo de recomendaciones aleatorias (Random).

Las métricas obtenidas para el modelo aleatorio reflejan que, como se esperaba, este enfoque no produce recomendaciones significativas para los usuarios, ni mucho menos personalizadas. En profundidad, podemos explicar cada uno y relacionarlo con la aleatoriedad del modelo:

- **MAP@10**: El valor extremadamente bajo indica que las recomendaciones aleatorias rara vez se alinean con las preferencias reales de los usuarios, lo que resulta en una baja precisión.
- **NDCG@10**: Similar al MAP, el valor bajo de NDCG sugiere que las recomendaciones aleatorias no logran ordenar adecuadamente los ítems según la relevancia para los usuarios.
- **Recall@10**: El recall también es bajo, lo que significa que aunque algunas recomendaciones pueden ser relevantes, en general el modelo aleatorio no cubre bien las preferencias de los usuarios.
- **Diversity**: Dado que las recomendaciones son aleatorias, la diversidad es bastante alta, lo que refleja que los ítems recomendados son variados, pero sin una conexión real con las preferencias del usuario.

- **Novelty**: El valor de novedad es moderado. Aunque las recomendaciones son aleatorias, algunas pueden incluir ítems menos populares, lo que aumenta su novedad.

Por otro lado, su implementación fue de muy baja dificultad puesto a que solamente había que aleatorizar las predicciones. Este modelo solo ocupa las memorias para guardar las matrices, pero no debe guardar para relacionar ni guardar factores latentes, y tuvo unos tiempos de ejecución casi instantáneos, del orden de 1 segundo.

3.4. BPR

Para este modelo, probamos los siguientes 3 casos y evaluamos sus diferencias:

Primera prueba: 100 factores, 10 iteraciones
Segunda prueba: 250 factores, 30 iteraciones
Tercera prueba: 500 factores, 5 iteraciones

Métrica	**MAP@10**	**NDCG@10**	**Recall@10**	**Diversity**	**Novelty**
Modelo 100 factores	0.000098	0.0010	0.0010	0.0005	9.90
Modelo 250 factores	0.000049	0.00049	0.00049	0.00054	9.79
Modelo 500 factores	0.000147	0.00148	0.00148	0.00059	8.77

Cuadro 13: Métricas de evaluación para el modelo BPR (Bayesian Personalized Ranking)

A diferencia de los modelos basados en vecinos como itemKNN o userKNN que vimos anteriormente, el modelo BPR permite captar relaciones más complejas entre los elementos, aunque, como se observa en el cuadro 13, las métricas obtenidas son considerablemente bajas en esta configuración y dataset.

En la primera prueba (100 factores, 10 iteraciones), el modelo logra su mejor balance en cuanto a precisión relativa (MAP@10: 0.000098, NDCG@10: 0.0010, Recall@10: 0.0010), con una novelty particularmente alta (9.90), indicando que tiende a recomendar ítems menos populares. Sin embargo, la diversity en esta configuración es extremadamente baja (0.0005), lo que sugiere que recomienda un conjunto muy limitado y repetitivo de ítems a los usuarios.

En la segunda prueba (250 factores, 30 iteraciones), las métricas de precisión bajan aún más (MAP@10: 0.000049), y aunque la diversity mejora levemente, sigue siendo muy baja (0.00054), manteniéndose la novelty alta (9.79). Esto sugiere que el modelo, al tener mayor complejidad, podría estar sobreajustando o sufriendo de problemas de convergencia con el número de iteraciones y factores elegidos.

La tercera prueba (500 factores, 5 iteraciones) presenta el mejor resultado en precisión relativa (MAP@10: 0.000147, NDCG@10: 0.00148, Recall@10: 0.00148), y la mayor diversity entre los tres ensayos (0.00059). Sin embargo, esto sigue siendo muy bajo en comparación con otros modelos como ItemKNN o incluso Random. Además, la novelty cae levemente a 8.77, aunque sigue siendo elevada.

En todos los casos, el tiempo de ejecución fue casi instantáneo, incluso en configuraciones con mayor número de factores, lo cual resalta la eficiencia del algoritmo implementado por la librería implicit. Sin embargo, la implementación de BPR es más compleja que otros modelos, requiriendo preprocesamiento de los datos, mapeo de IDs, y manejo explícito de matrices dispersas. Además, como modelo latente, la interpretabilidad de las recomendaciones es limitada, ya que no se puede conocer fácilmente por qué se recomienda un ítem a un usuario específico.

3.5. FunkSVD

FunkSVD es una técnica basada en factorización de matrices que descompone la matriz de interacciones usuario–ítem en dos matrices latentes más pequeñas, capturando patrones implícitos de preferencia.

Para su implementación, utilizamos la librería `pyreclab`, la cual facilitó la configuración de los hiperparámetros de entrenamiento, como número de factores latentes (*factors*), número máximo de iteraciones (*maxiter*), tasa de aprendizaje (*learning rate*) y regularización (*lambda*).

A continuación se detallan las combinaciones de hiperparámetros evaluadas:

Caso	**Factors**	**MaxIter**	**Learning Rate (lr)**	**Regularización (λ)**
1	100	50	0.005	0.1
2	100	100	0.01	0.1
3	200	150	0.05	0.1
4	200	50	0.01	0.1

Cuadro 14: Parámetros utilizados en cada ejecución del modelo FunkSVD

Los resultados se evaluaron utilizando tanto métricas de predicción (RMSE) como métricas de ranking: Recall@10, MAP@10, NDCG@10, Diversity y Novelty. El modelo fue capaz de generar predicciones para todos los usuarios e ítems, y mostró una mejora notable respecto a los métodos de vecinos (UserKNN y ItemKNN), especialmente considerando la alta dispersión del dataset.

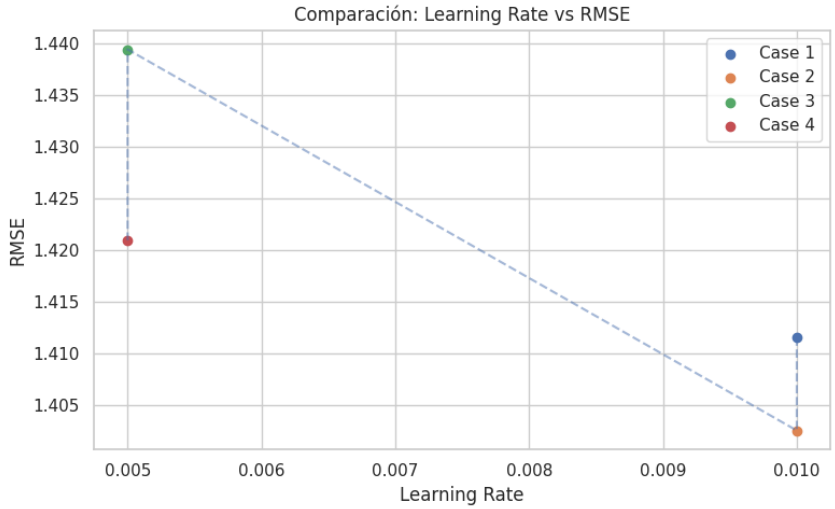


Figura 3: Relación entre tasa de aprendizaje (lr) y RMSE

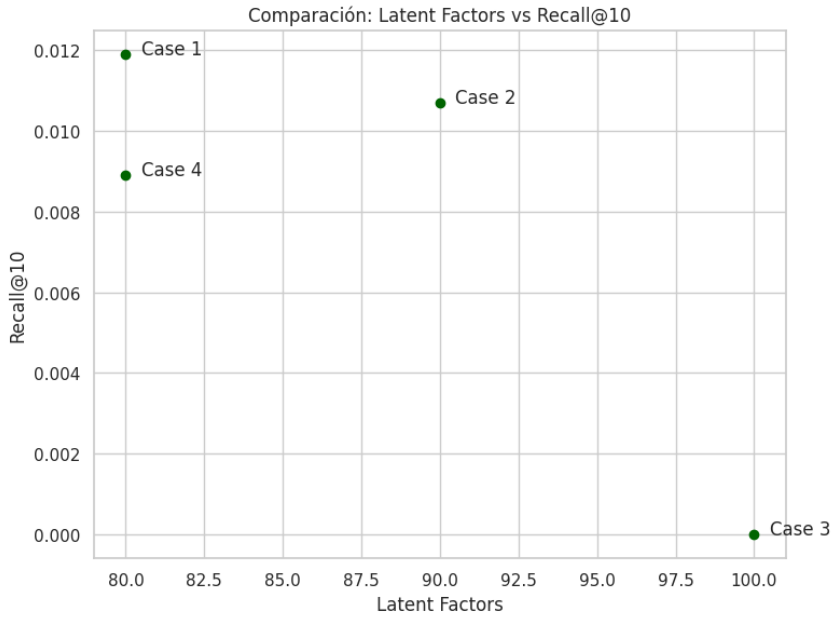


Figura 4: Impacto del número de factores sobre el Recall@10

Caso	RMSE	MAP@10	NDCG@10	Recall@10	Diversity	Novelty
1	1.4115	0.0051	0.0065	0.0119	22.26	6.72
2	1.4025	0.0045	0.0060	0.0111	21.90	6.69
3	1.4072	0.0031	0.0044	0.0079	21.54	6.58
4	1.4076	0.0043	0.0063	0.0109	21.79	6.66

Cuadro 15: Comparación de resultados del modelo FunkSVD bajo diferentes combinaciones de parámetros

De acuerdo con los resultados, el Caso 1 obtuvo un equilibrio adecuado entre error de predicción y calidad de recomendaciones. Aunque su RMSE fue ligeramente superior al

del Caso 2, logró mejores resultados en Recall@10 y MAP@10, lo que sugiere una mayor relevancia en las recomendaciones entregadas.

El tiempo de entrenamiento para cada configuración fue de aproximadamente 4 segundos, lo que convierte a FunkSVD en un modelo eficiente en comparación con UserKNN o ALS. Además, al no requerir el almacenamiento de matrices de similitud completas, su uso de memoria fue significativamente más bajo. La implementación fue sencilla gracias a la interfaz de Pyreclab, y permitió integrar las métricas adicionales como novelty y diversity utilizando datos de géneros y popularidad del dataset.

Finalmente, destacamos que FunkSVD fue uno de los modelos más robustos ante la baja densidad del dataset, siendo capaz de realizar predicciones razonables y personalizadas con bajo costo computacional.

3.6. ALS (Alternating Least Squares)

Alternating Least Squares (ALS) es un algoritmo basado en factorización de matrices, que alterna entre la optimización de las matrices de usuarios e ítems para minimizar el error cuadrático. A diferencia de otros enfoques, ALS permite trabajar directamente con *feedback implícito*, lo que lo convierte en una alternativa robusta cuando se dispone de pocos ratings explícitos o cuando los valores no son confiables.

Para esta implementación utilizamos la librería `implicit`, que optimiza el modelo sobre una matriz de preferencias binarizada (considerando como relevantes los ratings mayores o iguales a 5). Se entrenaron cuatro configuraciones diferentes del modelo, variando los hiperparámetros como el número de factores y la regularización.

Caso	**Factores latentes**	**Regularización (λ)**	**Iteraciones**
1	100	0.01	15
2	200	0.05	10
3	300	0.01	20
4	300	0.005	25

Cuadro 16: Parámetros utilizados para cada configuración del modelo ALS

A continuación, se muestran las métricas para cada caso

Caso	**MAP@10**	**NDCG@10**	**Recall@10**	**Diversity**	**Novelty**
1	0.0002	0.0007	0.0007	20.39	4.33
2	0.0001	0.0007	0.0020	21.20	3.10
3	0.0003	0.0013	0.0027	20.35	5.99
4	0.0003	0.0012	0.0017	20.28	7.58

Cuadro 17: Comparación de resultados del modelo ALS bajo distintas configuraciones

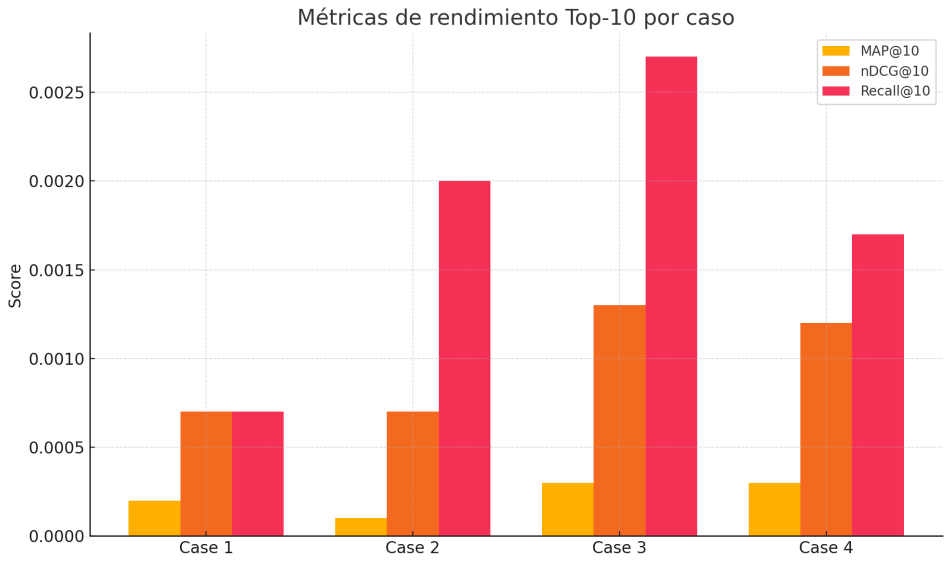


Figura 5: Comparación de MAP@10, NDCG@10 y Recall@10 entre los distintos casos

Como se puede observar, el **Caso 3** obtuvo los mejores valores en métricas de ranking, destacándose en *Recall@10* y *NDCG@10*. Sin embargo, sus valores absolutos siguen siendo bajos, lo cual es esperable dado el alto grado de dispersión del dataset. Aun así, ALS logra superar en precisión a modelos más simples como Random o ItemKNN.

Además, se midieron las métricas de *Diversity* y *Novelty*, las cuales permiten evaluar si las recomendaciones entregadas son variadas y novedosas para el usuario. A continuación se presenta un gráfico que muestra esta comparación.

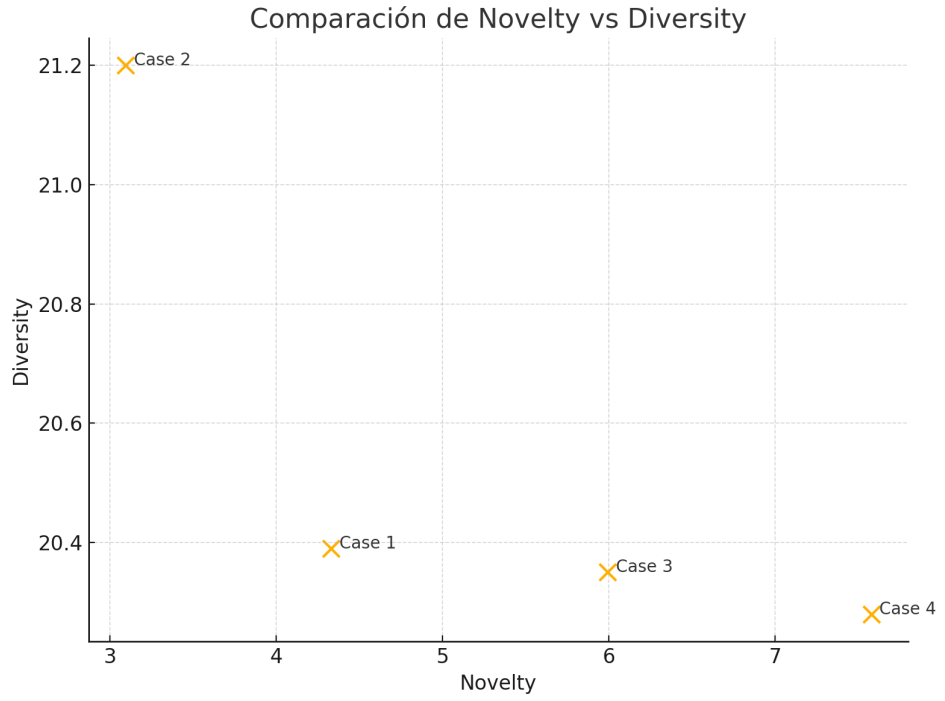


Figura 6: Relación entre novelty y diversity para los distintos casos de ALS

Este gráfico sugiere que el **Caso 4** entrega recomendaciones más novedosas (mayor

novelty), aunque con una pérdida leve en diversidad. Por otro lado, el **Caso 2** se caracteriza por ser el más diverso, pero con menor capacidad de acierto.

En conclusión, si bien los valores globales aún son bajos, el modelo ALS es consistente y balanceado, logrando adaptarse mejor al carácter implícito y disperso del dataset. Considerando tanto precisión como diversidad, el Caso 3 representa la mejor opción entre las configuraciones probadas. Sin embargo, los valores obtenidos no son suficientes como para emplear este modelo recomendador en un caso real, dado que los usuarios no recibirían recomendaciones adecuadas.

3.7. Factorization Machines (FM)

Las Factorization Machines son modelos de recomendación que permiten modelar interacciones entre pares de variables, generalizando técnicas como matrix factorization e incorporando información adicional de forma eficiente, por lo que es una modelación híbrida que considera tanto lo explícito, como lo implícito en los datos. En esta tarea utilizamos la librería `LightFM`, que permite incluir features de ítems como los géneros de los animes.

Se entrenaron dos versiones del modelo: una sobre un subconjunto sampleado del dataset y otra sobre el dataset completo. En ambos casos se consideraron 20 factores latentes y 15 iteraciones.

Versión	Recall@10	NDCG@10	MAP@10	Diversidad	Novedad
FM Sampleado	0.0075	0.0983	0.0050	6.16	8.6253
FM Completo	0.0077	0.1020	0.0049	8.00	7.2329

Cuadro 18: Comparación del rendimiento del modelo FM en dos versiones

Aunque los valores obtenidos aún son bajos en términos absolutos, se observa que el modelo completo mejora consistentemente las métricas de ranking y diversidad respecto a la versión reducida. Esto sugiere que el modelo se beneficia de una mayor cantidad de interacciones y features disponibles.

No obstante, dado que se utilizaron pocos factores e iteraciones, consideramos que FM tiene un potencial aún no explotado completamente. Sería recomendable realizar una búsqueda más exhaustiva de hiperparámetros y explorar regularizaciones y optimizadores alternativos para obtener un modelo final más competitivo.

4. Conclusiones Sobre Modelos de Predicción

A partir de la evaluación de todos los modelos implementados, se observa una notable diferencia en el rendimiento de las distintas estrategias. La Tabla 19 resume los resultados más representativos obtenidos para cada uno de los modelos considerados:

Método	RMSE	Recall@10	NDCG@10	MAP@10	Diversidad	Novedad	Tiempo
iKNN	–	0.0052	0.0027	0.0014	0.9995	7.9690	12 min
uKNN	–	–	–	–	–	–	13 min
FunkSVD	1.411	0.0119	0.0065	0.0051	22.2553	6.7241	4 s
ALS	–	0.0032	0.0015	0.0030	20.5600	6.0065	26.6 s
BPR	–	0.0015	0.0015	0.0002	0.0006	8.7700	1 s
FM	–	0.0077	0.1020	0.0049	8.0000	7.2329	30 s
Random	–	0.0045	0.0026	0.0013	0.9998	7.9534	1 s

Cuadro 19: Resultados comparativos de los modelos de recomendación

A partir de estos resultados, se puede concluir que el modelo **FunkSVD** presentó el mejor desempeño general, con valores superiores en todas las métricas de calidad de recomendación (Recall@10, MAP@10 y NDCG@10), además de mantener altos niveles de diversidad en las recomendaciones. Este modelo también demostró ser eficiente en términos de tiempo de entrenamiento y consumo de recursos.

Por otro lado, el modelo **Factorization Machines (FM)** también mostró resultados prometedores, especialmente en *NDCG@10*, aunque aún no logra superar a FunkSVD en precisión general. Su desempeño podría verse potenciado si se optimizan hiperparámetros como la cantidad de factores y las iteraciones.

El modelo **ALS**, aunque competitivo, se ve superado por FunkSVD en todas las métricas, aunque sigue siendo una alternativa sólida especialmente para datasets implícitos.

Modelos como **iKNN**, **BPR** y **Random** presentaron resultados considerablemente más bajos, evidenciando una menor capacidad de personalización y precisión, particularmente en un contexto de datos altamente dispersos como el utilizado en esta tarea.

Aunque FunkSVD destaca como el mejor modelo en este escenario, los resultados obtenidos aún se encuentran lejos de ser óptimos. Las métricas de evaluación reflejan una dificultad estructural en el dataset, como su alta dispersión y baja cantidad de interacciones por usuario. Por lo tanto, se concluye que será necesario seguir iterando sobre los modelos actuales, probar nuevas configuraciones de hiperparámetros y considerar enfoques híbridos o de aprendizaje profundo para encontrar un modelo predictor realmente competitivo y robusto.