Máster en Inteligencia Artificial Aplicada

Unidad: Systems Recomendations - Proyecto Aplicacion

Sistema de Recomendación de Películas

Basado en Metodos de Similitud de Coseno y SVD

Nombre: Patricio Galván

Fecha: 27 de Marzo 2025

Sistema de Recomendación de Películas – Análisis Comparativo

Introducción

Este proyecto tiene como objetivo implementar y comparar diferentes enfoques de sistemas de recomendación aplicados al dataset **MovieLens 100K**. A través de este análisis buscamos entender el comportamiento de distintos modelos en términos de precisión, diversidad y cobertura, así como su aplicabilidad en un escenario real de recomendación personalizada.

Métodos Utilizados

Se implementaron dos enfoques principales de recomendación:

1. **Filtrado Colaborativo Basado en Similitud del Coseno (Ítem-Ítem):**
   Este método calcula la similitud entre películas en función de las calificaciones de los usuarios y recomienda aquellas similares a lo que el usuario ha visto recientemente. Se agregó un algoritmo que impidiera que se recomendaran peliculas ya vistas por el usuario.
2. **Factorización de Matrices con SVD (Singular Value Decomposition):**
   Utilizamos la biblioteca `Surprise` para implementar un modelo SVD capaz de predecir calificaciones desconocidas y generar recomendaciones.
   Además, se evaluaron:
   - Un **modelo base SVD**
   - Un **modelo optimizado mediante Grid Search**
   - Un **modelo optimizado mediante Random Search**

Nota2: Importante mencionar que para los 2 metodos se consideró implementar la **Partida en Frio** con datos del **Metodo de Items por Popularidad**.

Dataset

Se trabajó con el dataset **MovieLens 100K**, que contiene:

- 100,000 calificaciones explícitas de 943 usuarios sobre 1,682 películas.
- Calificaciones en una escala de 1 a 5.
- Información adicional de cada película (título, fecha de lanzamiento).

Los datos fueron preprocesados para construir:

- Una matriz de usuario-película.
- Una matriz de similitud ítem-ítem.
- Conjuntos de entrenamiento y prueba para los modelos basados en SVD.

Evaluación de Modelos

Se evaluaron los modelos considerando múltiples dimensiones:

- **Precisión@5 y Recall@5**: métricas clave en sistemas de recomendación, que indican la relevancia y capacidad de recuperación de las sugerencias generadas dentro del top 5.
- **RMSE (Root Mean Square Error)** y **MAE (Mean Absolute Error)**: utilizados como indicadores secundarios de la calidad de predicción de calificaciones numéricas.
- **Popularidad y dispersión de las películas recomendadas**: para analizar la tendencia hacia contenido popular y la diversidad en las preferencias de los usuarios.
- **Cobertura**: porcentaje de usuarios para los que se pueden generar recomendaciones válidas sin colisión con contenido ya visto.

Resultados

- El modelo **SVD base**, sin optimización, presentó el mejor desempeño global tanto en precisión como en recall, superando incluso a las versiones optimizadas mediante Grid Search y Random Search.
- El modelo **SVD optimizado con Random Search** logró un rendimiento razonablemente cercano, pero no logró superar al modelo base, lo que sugiere que los hiperparámetros por defecto ya están bien adaptados al dataset.
- El modelo basado en **Similitud del Coseno** mostró una marcada preferencia por títulos más populares, con menor precisión y recall, y una capacidad limitada para personalizar las recomendaciones.
- En el análisis individual por usuario (por ejemplo, el usuario ID 156), **ninguno de los modelos recomendó películas ya vistas**, lo que valida su utilidad como motores de descubrimiento.

- No se observaron intersecciones en las recomendaciones entre SVD y Coseno, lo cual evidencia una **alta diversidad** entre ambos enfoques.

Conclusiones

- El modelo **SVD base** demostró ser el más efectivo en términos de relevancia, precisión y estabilidad de las recomendaciones, tanto a nivel agregado como individual.
- La falta de mejora por parte de los modelos optimizados mediante Grid y Random Search, aunque inusual, es un comportamiento posible y será objeto de análisis más profundo en una etapa posterior del proyecto.
- El enfoque de **Similitud del Coseno**, si bien limitado en rendimiento, puede ser útil como componente complementario, especialmente para promover contenido de alta popularidad.
- Dada la diversidad en los resultados generados por ambos modelos, su **combinación en un sistema híbrido** representa una oportunidad valiosa para equilibrar personalización, popularidad y descubrimiento.
- El pipeline desarrollado se ha diseñado con modularidad y escalabilidad en mente, y está preparado para adaptarse a versiones más amplias del dataset como MovieLens 1M o 10M.

1.0 Preparacion Entorno y Funciones de Apoyo

```
# 1.0 PREPARACION ENTORNO
# ======================
#!pip install surprise -q

# Hoy en dia exiten problemas de compatibilidad en COLAB entre la version de Surprise y N
# Por ello aplicaremos el sigeuinte procedimiento

# Instalamos NumPy 1.23.x, que era bastante estable
!pip install --no-cache-dir "numpy==1.23.5" -q

# Instalamos una versión concreta de scikit-surprise
!pip install --no-cache-dir "scikit-surprise==1.1.1" -q

# Reiniciamos el kernel para que los cambios en NumPy se apliquen correctamente
# Esto es importante para resolver el problema de importación
import IPython
IPython.Application.instance().kernel.do_shutdown(True)
# Despues del Reinicio solamente hay que continuar en la sigueinte celda
```

```
# 2.0 CARGA DE LIBRERIAS
import os
import urllib.request
import zipfile
```

```
import pickle

import numpy as np
import pandas as pd

from collections import defaultdict

import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display

from scipy.sparse import csr_matrix
from sklearn.metrics.pairwise import cosine_similarity

from surprise import SVD, Dataset, Reader, accuracy
from surprise.model_selection import train_test_split, cross_validate, GridSearchCV, Rand
```

2.0 Carga y Exploración de Datos

```
# 1 Caraga de datos
# ====================

# Definicion de Path
DATASET_URL = "http://files.grouplens.org/datasets/movielens/ml-100k.zip"
DATASET_PATH = "/content/ml-100k.zip"
EXTRACT_FOLDER = "/content/ml-100k"


# Descarga y extracción del dataset si no está presente
if not os.path.exists(EXTRACT_FOLDER):
    print("Descargando MovieLens 100k...")
    urllib.request.urlretrieve(DATASET_URL, DATASET_PATH)
    print("Descarga completa.")

    print("Extrayendo archivos...")
    with zipfile.ZipFile(DATASET_PATH, "r") as zip_ref:
        zip_ref.extractall("/content/")
    print("Extracción completa.")

# Carga de archivos
ratings_path = os.path.join(EXTRACT_FOLDER, "u.data")
movies_path = os.path.join(EXTRACT_FOLDER, "u.item")

# Cargar ratings
columns_ratings = ["user_id", "movie_id", "rating", "timestamp"]
ratings_df = pd.read_csv(ratings_path, sep="\t", names=columns_ratings, encoding="latin-1

# Cargar información de películas
columns_movies = ["movie_id", "title", "release_date", "video_release_date", "IMDB_URL",
                  "Adventure", "Animation", "Children", "Comedy", "Crime", "Documentary",
                  "Film-Noir", "Horror", "Musical", "Mystery", "Romance", "Sci-Fi", "Thri
movies_df = pd.read_csv(movies_path, sep="|", names=columns_movies, encoding="latin-1", u
```

```
# 2 Exploración de datos
# ========================

print("\n### Exploración Inicial de Datos ###\n")

## 1. Información general
print(" Cantidad de registros en cada dataset:")
print(f"✅ Ratings: {ratings_df.shape[0]} filas, {ratings_df.shape[1]} columnas")
print(f"✅ Películas: {movies_df.shape[0]} filas, {movies_df.shape[1]} columnas\n")

## 2. Primeras filas de cada dataset
print("🔍 **Ejemplo de datos en 'ratings_df' (calificaciones de usuarios a películas):**'
display(ratings_df.head())

print("\n🔍 **Ejemplo de datos en 'movies_df' (información de películas):**")
display(movies_df.head())

## 3. Estadísticas descriptivas de las calificaciones
print("\n **Estadísticas descriptivas de las calificaciones:**")
display(ratings_df["rating"].describe())

## 4. Cantidad de usuarios y películas únicas
print("\n **Cantidad de elementos únicos en el dataset:**")
print(f" Películas únicas: {ratings_df['movie_id'].nunique()}")
print(f" Usuarios únicos: {ratings_df['user_id'].nunique()}")

## 5. Distribución de calificaciones
print("\n **Distribución de calificaciones:**")

plt.figure(figsize=(8, 5))
ratings_counts = ratings_df["rating"].value_counts().sort_index()
plt.bar(ratings_counts.index, ratings_counts.values, edgecolor="black", alpha=0.7)
plt.xticks([1, 2, 3, 4, 5])
plt.xlabel("Rating")
plt.ylabel("Frecuencia")
plt.title("Distribución de Calificaciones en MovieLens 100k")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

## 6. Análisis de cantidad de calificaciones por usuario
ratings_per_user = ratings_df.groupby("user_id")["rating"].count()
print("\n **Cantidad de calificaciones por usuario:**")
display(ratings_per_user.describe())

## 7. Análisis de cantidad de calificaciones por película
ratings_per_movie = ratings_df.groupby("movie_id")["rating"].count()
print("\n **Cantidad de calificaciones por película:**")
display(ratings_per_movie.describe())
```

Exploración Inicial de Datos

Cantidad de registros en cada dataset:
✅ Ratings: 100000 filas, 4 columnas
✅ Películas: 1682 filas, 3 columnas

🔍 **Ejemplo de datos en 'ratings_df' (calificaciones de usuarios a películas):**

	user_id	movie_id	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

🔍 **Ejemplo de datos en 'movies_df' (información de películas):**

	movie_id	title	release_date
0	1	Toy Story (1995)	01-Jan-1995
1	2	GoldenEye (1995)	01-Jan-1995
2	3	Four Rooms (1995)	01-Jan-1995
3	4	Get Shorty (1995)	01-Jan-1995
4	5	Copycat (1995)	01-Jan-1995

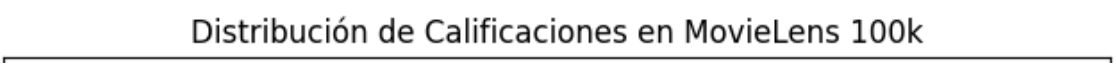
Estadísticas descriptivas de las calificaciones:

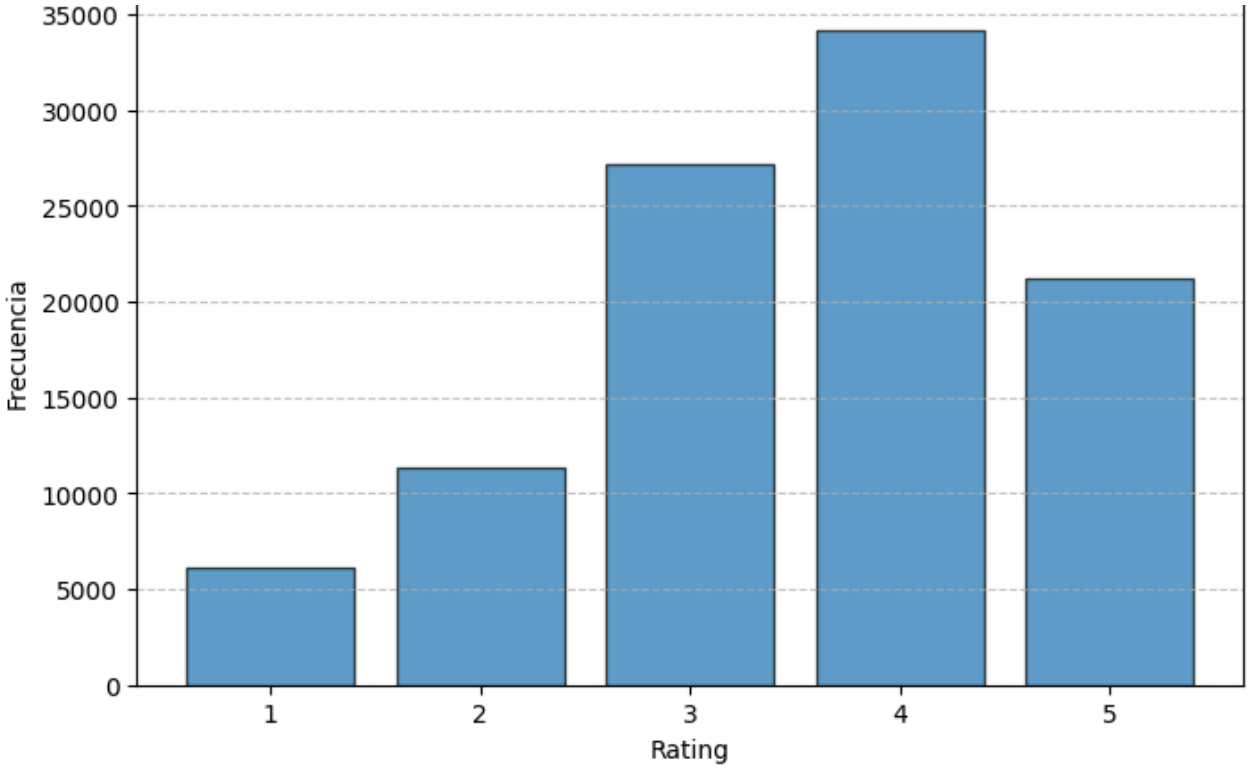
	rating
count	100000.000000
mean	3.529860
std	1.125674
min	1.000000
25%	3.000000
50%	4.000000
75%	4.000000
max	5.000000

dtype: float64

Cantidad de elementos únicos en el dataset:
Películas únicas: 1682
Usuarios únicos: 943

Distribución de calificaciones:





Cantidad de calificaciones por usuario:

	rating
count	943.000000
mean	106.044539
std	100.931743
min	20.000000
25%	33.000000
50%	65.000000
75%	148.000000
max	737.000000

dtype: float64

Cantidad de calificaciones por película:

	rating
count	1682.000000
mean	59.453032
std	80.383846
min	1.000000
25%	6.000000
50%	27.000000
75%	80.000000
max	583.000000

dtype: float64

3.0 Construcción de Matrices para los Métodos de Recomendación

Descripción

En esta sección, construimos las matrices fundamentales para los distintos enfoques de recomendación. Cada método requiere una estructura específica, por lo que preparamos tres representaciones clave:

1 Matriz de Popularidad (Para Partida en Frío)

- Se basa en el número de calificaciones recibidas por cada película.
- Permite recomendar películas a usuarios sin historial previo.
- La clasificación prioriza la cantidad de calificaciones y, en caso de empate, el promedio de rating.

2 Matriz Ítem-Ítem (Para Similitud del Coseno)

- Utiliza la similitud entre películas para generar recomendaciones.
- Se obtiene transponiendo la matriz usuario-película antes de aplicar la Similitud del Coseno.
- Optimizada con una representación dispersa (`csr_matrix`) para mejorar la eficiencia.

3 Matriz Usuario-Ítem (Para SVD)

- Representa las calificaciones de los usuarios sobre las películas.
- Utilizada en la factorización de matrices para identificar patrones latentes en las preferencias.
- `Surprise` maneja automáticamente los datos dispersos, por lo que no es necesario convertir la matriz a formato `sparse`.
- Se utiliza el formato `long format` (`user_id`, `movie_id`, `rating`) en lugar de una matriz pivotada para mayor eficiencia en `Surprise`.

```
# ======================================================
# 1 MATRIZ PARA METODO POR POPULARIDAD (Partida en frío)
# ======================================================

def compute_popular_movies(ratings_df, movies_df, n=10):
    """
    Calcula las películas más populares basadas en cantidad de calificaciones y promedio

    Parámetros:
    - ratings_df: DataFrame con calificaciones de usuarios.
    - movies_df: DataFrame con información de películas.
```

```
    - n: Número de películas populares a retornar.

    Retorna:
    - DataFrame con las películas más populares, incluyendo título y estadísticas de popu
    """
    # Calcular cantidad de calificaciones y promedio de rating
    popular_movies = (ratings_df.groupby("movie_id")
                      .agg(count=("rating", "count"), mean_rating=("rating", "mean"))
                      .sort_values(by=["count", "mean_rating"], ascending=[False, False])
                      .head(n))

    # Asociar nombre de la película
    popular_movies = popular_movies.merge(movies_df[['movie_id', 'title']], on="movie_id"

    return popular_movies

# Generar la matriz de popularidad
popular_movies_df = compute_popular_movies(ratings_df, movies_df)

# Mostrar películas populares
print("\n **Películas más populares (Para Partida en Frío):**")
display(popular_movies_df)
```

Películas más populares (Para Partida en Frío):

	movie_id	count	mean_rating	title
0	50	583	4.358491	Star Wars (1977)
1	258	509	3.803536	Contact (1997)
2	100	508	4.155512	Fargo (1996)
3	181	507	4.007890	Return of the Jedi (1983)
4	294	485	3.156701	Liar Liar (1997)
5	286	481	3.656965	English Patient, The (1996)
6	288	478	3.441423	Scream (1996)
7	1	452	3.878319	Toy Story (1995)
8	300	431	3.631090	Air Force One (1997)
9	121	429	3.438228	Independence Day (ID4) (1996)

```
# ======================================================
# 2 MATRIZ PARA METODO SIMILITUD COSENO ÍTEM-ÍTEM
# ======================================================

def create_ratings_matrix(ratings_df):
    """
    Construye la matriz de calificaciones usuario-película.
```

```
    Parámetros:
    - ratings_df: DataFrame con columnas ['user_id', 'movie_id', 'rating'].

    Retorna:
    - DataFrame con usuarios como filas, películas como columnas y calificaciones como va
    """
    ratings_matrix = ratings_df.pivot(index="user_id", columns="movie_id", values="rating
    return ratings_matrix

def compute_item_similarity_sparse(ratings_matrix):
    """
    Calcula la matriz de similitud Ítem-Ítem basada en la Similitud del Coseno optimizada

    Parámetros:
    - ratings_matrix: DataFrame usuario-película con calificaciones.

    Retorna:
    - DataFrame con la matriz de similitud entre películas.
    """
    filled_matrix = ratings_matrix.T.fillna(0)  # Transponer y reemplazar NaN con 0
    sparse_matrix = csr_matrix(filled_matrix)  # Convertir a formato disperso para optimi

    # Cálculo de similitud del coseno
    item_similarity = cosine_similarity(sparse_matrix)

    # Convertir a DataFrame con nombres de películas como índices y columnas
    return pd.DataFrame(item_similarity, index=ratings_matrix.columns, columns=ratings_ma

# Crear matriz usuario-película
ratings_matrix = create_ratings_matrix(ratings_df)

# Construcción de la matriz de similitud Ítem-Ítem optimizada
item_similarity_matrix = compute_item_similarity_sparse(ratings_matrix)

# Exploración inicial
print("\n **Matriz de Similitud Ítem-Ítem (Vista parcial de 5x5):**")
display(item_similarity_matrix.iloc[:5, :10])
```

```
**Matriz de Similitud Ítem-Ítem (Vista parcial de 5x5):**
```

movie_id	1	2	3	4	5	6	7	8
movie_id								
1	1.000000	0.402382	0.330245	0.454938	0.286714	0.116344	0.620979	0.481114
2	0.402382	1.000000	0.273069	0.502571	0.318836	0.000500	0.383403	0.337002
3	0.330245	0.273069	1.000000	0.324866	0.212957	0.106722	0.372921	0.200794
4	0.454938	0.502571	0.324866	1.000000	0.334239	0.090308	0.489283	0.490236
5	0.286714	0.318836	0.212957	0.334239	1.000000	0.037299	0.334769	0.259161

```
# ========================================================
# 3 MATRIZ PARA METODO SVD USUARIO - ÍTEM
# ========================================================

def create_user_item_matrix(ratings_df, fill_na=False):
    """
    Construye la matriz Usuario-Ítem que será usada en SVD.

    Parámetros:
    - ratings_df: DataFrame con calificaciones de usuarios.
    - fill_na: Si es True, reemplaza NaN con 0 para mejor visualización.

    Retorna:
    - DataFrame con usuarios como filas, películas como columnas y calificaciones como va
    """
    matrix = ratings_df.pivot(index="user_id", columns="movie_id", values="rating")

    if fill_na:
        matrix = matrix.fillna(0)  # Solo para visualización, no se usa en SVD

    return matrix

def prepare_surprise_data(ratings_df):
    """
    Prepara los datos en formato Surprise para SVD.

    Parámetros:
    - ratings_df: DataFrame con calificaciones de usuarios.

    Retorna:
    - Dataset de Surprise listo para entrenar con SVD.
    """
    reader = Reader(rating_scale=(1, 5))
    data = Dataset.load_from_df(ratings_df[["user_id", "movie_id", "rating"]], reader)
    return data

# Construcción de la matriz Usuario-Ítem para SVD (original y visual)
user_item_matrix = create_user_item_matrix(ratings_df, fill_na=False)  # Versión para SVD
user_item_matrix_filled = create_user_item_matrix(ratings_df, fill_na=True)  # Versión pa

# Conversión al formato de Surprise
surprise_data = prepare_surprise_data(ratings_df)

# Exploración inicial
print("\n📌 **Matriz Usuario-Ítem para SVD (Vista parcial de 5x5, con Nulos reemplazados
display(user_item_matrix_filled.head(5).iloc[:, :5])
```

📌 **Matriz Usuario-Ítem para SVD (Vista parcial de 5x5, con Nulos reemplazados para

movie_id	1	2	3	4	5
user_id					
1	5.0	3.0	4.0	3.0	3.0
2	4.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0
5	4.0	3.0	0.0	0.0	0.0

4.0 Metodo Recomendación por Similitud Coseno

En esta sección se implementa un sistema de recomendación basado en la **Similitud del Coseno entre películas**. Este enfoque identifica películas similares a una que el usuario haya visto recientemente, analizando patrones de calificación entre ítems (Ítem-Ítem).

Metodología

1 Construcción de la Matriz de Similitud Ítem-Ítem

- Se parte de la matriz usuario-película (ratings) transpuesta.
- Se aplica la función `cosine_similarity()` para calcular similitudes entre películas.
- El resultado se convierte en un `DataFrame` con índices `movie_id` para facilitar su uso posterior.

2 Motor de Recomendación Basado en Similaridad

- Dado un `movie_id` base, se obtienen las películas más similares según la matriz.
- Se excluye la película de referencia y las películas ya vistas por el usuario (si aplica).
- En caso de que no haya suficientes ítems similares no vistos, se completa con **películas populares no vistas** por el usuario.

3 Manejo de Partida en Frío (Usuarios Nuevos)

- Si el usuario no tiene historial, se genera una lista de películas populares ordenadas por cantidad de calificaciones y promedio de rating.

Resultados y Ventajas

- Se generan recomendaciones personalizadas que consideran tanto **similitud de contenido** como **historial del usuario**.
- El sistema evita recomendar contenido ya visto.

- En caso de falta de datos, ofrece un **fallback inteligente** basado en popularidad.
- La implementación es **eficiente, escalable y flexible**, compatible con cualquier cantidad de usuarios o películas.

Con esta arquitectura, el sistema es capaz de entregar recomendaciones efectivas tanto a **usuarios activos** como a **usuarios nuevos**, garantizando una experiencia robusta y adaptable.

```
# ======================================
# 1 CÁLCULO DE SIMILITUD ENTRE PELÍCULAS
# ======================================

def compute_precision_recall_at_k(similarity_matrix, ratings_df, movies_df, k=5):
    """
    Calcula la precisión y recall promedio para todos los usuarios usando recomendaciones
    """
    precision_list = []
    recall_list = []

    for user_id in ratings_df["user_id"].unique():
        # Películas que el usuario calificó como relevantes (rating >= 3)
        relevant_movies = set(ratings_df[(ratings_df["user_id"] == user_id) & (ratings_df

        # Películas que el usuario ha visto
        seen_movies = ratings_df[ratings_df["user_id"] == user_id]["movie_id"].tolist()

        if not seen_movies or len(relevant_movies) == 0:
            continue  # Saltar usuarios sin historial o sin relevantes

        # Tomar la última película vista como base para recomendación (ejemplo simple)
        last_movie = seen_movies[-1]

        # Obtener recomendaciones
        try:
            recommended_movies = set(
                [x[0] for x in recommend_movies(last_movie, similarity_matrix, movies_df,
            )
        except:
            continue  # Si la película no tiene similitud definida

        # Cálculo de precisión y recall
        hits = len(relevant_movies & recommended_movies)
        precision = hits / k
        recall = hits / len(relevant_movies) if relevant_movies else 0

        precision_list.append(precision)
        recall_list.append(recall)

    return np.mean(precision_list), np.mean(recall_list)

def compute_movie_similarity(sparse_matrix):
    """
    Calcula la similitud del coseno entre películas a partir de la matriz usuario-películ
```

```
    Parámetros:
    - sparse_matrix: Matriz dispersa en formato CSR.

    Retorna:
    - Matriz de similitud entre películas (numpy array).
    """
    similarity_matrix = cosine_similarity(sparse_matrix.T)  # Transponemos para calcular

    # Evitar que una película sea su propia recomendación
    np.fill_diagonal(similarity_matrix, 0)

    return similarity_matrix

# Convertir la matriz usuario-película en una matriz dispersa antes de calcular la simili
ratings_sparse = csr_matrix(ratings_matrix.fillna(0))
movie_similarity = compute_movie_similarity(ratings_sparse)

# Calculamos la matriz de similitud entre películas
movie_similarity = compute_movie_similarity(ratings_sparse)

# Verificación de la matriz de similitud
print("\nDimensiones de la matriz de similitud entre películas:", movie_similarity.shape)
print("Ejemplo de similitud entre las primeras películas:")
display(movie_similarity[:10, :5])
```

```
Dimensiones de la matriz de similitud entre películas: (1682, 1682)
Ejemplo de similitud entre las primeras películas:
array([[0.        , 0.40238218, 0.33024479, 0.45493792, 0.28671351],
       [0.40238218, 0.        , 0.27306918, 0.50257077, 0.31883618],
       [0.33024479, 0.27306918, 0.        , 0.32486639, 0.21295656],
       [0.45493792, 0.50257077, 0.32486639, 0.        , 0.33423948],
       [0.28671351, 0.31883618, 0.21295656, 0.33423948, 0.        ],
       [0.11634398, 0.08356281, 0.10672227, 0.09030829, 0.03729866],
       [0.62097859, 0.38340339, 0.37292069, 0.4892828 , 0.33476858],
       [0.48111389, 0.33700186, 0.20079389, 0.49023553, 0.25916097],
       [0.49628843, 0.25525203, 0.27366928, 0.41904357, 0.2724484 ],
       [0.27393511, 0.17108221, 0.15810426, 0.25256072, 0.05545322]])
```

```
# ======================================================================
# 2 MOTOR DE RECOMENDACIÓN BASADO EN SIMILITUD DE PELÍCULAS
# ======================================================================

def recommend_movies(movie_id, similarity_matrix, movies_df, ratings_df, top_n=5, user_id
    """
    Recomienda películas similares a una dada, evitando repetir películas ya vistas
    y completando con contenido popular si es necesario.

    Parámetros:
    - movie_id: ID de la película base para calcular similitud.
    - similarity_matrix: Matriz de similitud entre películas (DataFrame).
    - movies_df: DataFrame con información de películas.
    - ratings_df: DataFrame con calificaciones de usuarios.
    - top_n: Número de recomendaciones a entregar.
    - user_id: (opcional) ID del usuario para evitar recomendar películas ya vistas.
```

```
    - verbose: Si True, imprime mensajes cuando se aplica fallback.

    Retorna:
    - Lista de películas recomendadas como (movie_id, title).
    """
    # Convertir similarity_matrix a DataFrame si es un array NumPy
    if isinstance(similarity_matrix, np.ndarray):
        similarity_matrix = pd.DataFrame(similarity_matrix, index=movies_df['movie_id'],

    # Verificar que la película base está en el índice de la matriz
    if movie_id not in similarity_matrix.index:
        if verbose:
            print(f"⚠️ Película ID {movie_id} no está en la matriz de similitud.")
        return []

    # Obtener las puntuaciones de similitud
    similarity_scores = similarity_matrix.loc[movie_id]

    # Ordenar películas similares excluyendo la película base
    similar_movie_ids = similarity_scores.drop(index=movie_id).sort_values(ascending=Fals

    # Obtener títulos asociados
    recommended_movies = movies_df[movies_df["movie_id"].isin(similar_movie_ids)][["movie

    # Si no se especifica usuario, entregar las top_n más similares directamente
    if user_id is None:
        return recommended_movies[:top_n]

    # Filtrar películas ya vistas por el usuario
    user_seen_movies = set(ratings_df[ratings_df["user_id"] == user_id]["movie_id"])
    filtered = [m for m in recommended_movies if m[0] not in user_seen_movies]

    # Si hay suficientes recomendaciones no vistas, usar solo esas
    if len(filtered) >= top_n:
        return filtered[:top_n]

    # Fallback: completar con películas populares no vistas
    needed = top_n - len(filtered)
    popular_movies = compute_popular_movies(ratings_df, movies_df, n=100)
    popular_not_seen = [
        (row["movie_id"], row["title"]) for _, row in popular_movies.iterrows()
        if row["movie_id"] not in user_seen_movies and row["movie_id"] not in [m[0] for m
    ]

    if verbose:
        print(f"⚠️ Solo se encontraron {len(filtered)} recomendaciones por similitud. Co

    return filtered + popular_not_seen[:needed]


# 3 MANEJO DE PARTIDA EN FRÍO (USUARIOS NUEVOS)
# ================================================

def recommend_for_new_user(ratings_df, movies_df, top_n=5):
    """
```

```
    Recomienda películas basadas en popularidad si el usuario es nuevo.

    Parámetros:
    - ratings_df: DataFrame con calificaciones de usuarios.
    - movies_df: DataFrame con información de películas.
    - top_n: Número de películas recomendadas.

    Retorna:
    - Lista de películas populares recomendadas.
    """
    return compute_popular_movies(ratings_df, movies_df, n=top_n)


# 4 EJEMPLO DE RECOMENDACIÓN
# ============================

Pelicula = 50
movie_example = movies_df["movie_id"].iloc[Pelicula]  # Tomamos una película del dataset
recommended_movies = recommend_movies(movie_example, movie_similarity, movies_df, ratings

# Mostrar resultados
print("** PRUEBA DEL MOTOR SIMILITUD COSENO **")
print(50*"=")
print("\n🎬 Película base:", movies_df[movies_df["movie_id"] == movie_example]["title"].v
print("\n Películas recomendadas:")
print(25*"-")
for movie in recommended_movies:
    print(f"- {movie[1]} (ID: {movie[0]})")

# Simulación de usuario nuevo
print(50*"=")
print("\n **Simulación de usuario nuevo **")
print("\n Partida en Frio => Recomendaciones por popularidad\n")
new_user_recommendations = recommend_for_new_user(ratings_df, movies_df, top_n=5)
display(new_user_recommendations)
```

```
** PRUEBA DEL MOTOR SIMILITUD COSENO **
================================================

🎬 Película base: Legends of the Fall (1994)

 Películas recomendadas:
------------------------
- Toy Story (1995) (ID: 1)
- GoldenEye (1995) (ID: 2)
- Four Rooms (1995) (ID: 3)
- Get Shorty (1995) (ID: 4)
- Copycat (1995) (ID: 5)
================================================

 **Simulación de usuario nuevo **

 Partida en Frio => Recomendaciones por popularidad
```

	movie_id	count	mean_rating	title
0	50	583	4.358491	Star Wars (1977)
1	258	509	3.803536	Contact (1997)
2	100	508	4.155512	Fargo (1996)
3	181	507	4.007890	Return of the Jedi (1983)
4	294	485	3.156701	Liar Liar (1997)

5.0 Metodo Factorizacion de Matrices (SVD)

En esta sección, implementamos un sistema de recomendación basado en **Factorización de Matrices con SVD (Singular Value Decomposition)**. Este enfoque permite identificar patrones latentes en las preferencias de los usuarios y generar recomendaciones más personalizadas.

Pasos Implementados

1 Entrenamiento del Modelo SVD

- Se entrena un modelo SVD utilizando la biblioteca `Surprise`, optimizado con hiperparámetros ajustados.
- Se usa `build_full_trainset()` para entrenar con **todos los datos disponibles**, eliminando la necesidad de dividir en conjuntos de entrenamiento y prueba.
- El modelo se **guarda en un archivo (`pickle`)** para evitar reentrenamientos innecesarios.

2 Generación de Recomendaciones Optimizada

- Se obtiene la lista de películas que el usuario ha calificado **directamente desde el conjunto de entrenamiento (`trainset`)**, evitando consultas a `ratings_df`.

- Se predicen las calificaciones para las películas no vistas con `model.test()`, **acelerando drásticamente la recomendación** en comparación con el cálculo secuencial `model.predict()`.
- Se optimiza la búsqueda de películas vistas mediante `trainset.ur` y `trainset.to_raw_iid()`, evitando errores y mejorando la eficiencia.

3️⃣ **Manejo de Partida en Frío (Usuarios Nuevos)**

- Si un usuario no tiene historial de calificaciones, el sistema recomienda **películas populares** en lugar de intentar predecir ratings inexistentes.
- Se incorpora una verificación que previene errores si el `user_id` no está en `trainset`.

Resultados

- Se obtiene un **modelo SVD altamente eficiente**, con predicciones **mucho más rápidas** gracias a la vectorización con `model.test()`.
- La implementación optimizada **reduce significativamente el tiempo de ejecución** de la recomendación en comparación con versiones anteriores.
- Se maneja correctamente la **partida en frío**, permitiendo recomendaciones incluso para usuarios sin historial previo.

Este modelo servirá como base para la comparación con otros enfoques, evaluando su **precisión, eficiencia y escalabilidad**.

```
%%time
# ================================
# 1 Entrenamiento del Modelo SVD
# ================================

def train_svd_model(data, save_path="model_svd.pkl"):
    """
    Entrena un modelo SVD con todos los datos y lo guarda en un archivo.

    Parámetros:
    - data: Dataset de Surprise con las calificaciones.
    - save_path: Ruta para guardar el modelo entrenado.

    Retorna:
    - Modelo SVD entrenado y conjunto de entrenamiento.
    """
    print("\n Entrenando modelo SVD con regularización...\n")

    # Configuración del modelo con hiperparámetros ajustados
    model_svd = SVD(n_factors=100, n_epochs=15, lr_all=0.01, reg_all=0.02)

    # Entrenar con todos los datos
    trainset = data.build_full_trainset()
    model_svd.fit(trainset)

    # Guardar modelo entrenado
```

```
    with open(save_path, "wb") as f:
        pickle.dump(model_svd, f)

    print("✅ Modelo SVD guardado en:", save_path,"\n")
    return model_svd, trainset


# ==========================================
# 2. Generación de Recomendaciones (OPTIMIZADA)
# ==========================================
def recommend_movies_svd(user_id, model, trainset, movies_df, top_n=5):
    """
    Recomienda películas a un usuario usando SVD de manera rápida.

    Parámetros:
    - user_id: ID del usuario para generar recomendaciones.
    - model: Modelo SVD entrenado.
    - trainset: Conjunto de entrenamiento usado en Surprise.
    - movies_df: DataFrame con información de películas.
    - top_n: Número de películas recomendadas.

    Retorna:
    - Lista de películas recomendadas ordenadas por calificación predicha.
    """
    try:
        inner_uid = trainset.to_inner_uid(user_id)
    except ValueError:
        print(f"\n Usuario {user_id} no encontrado. Recomendando películas populares...")
        return compute_popular_movies(ratings_df, movies_df, top_n)

    # Obtener películas vistas por el usuario
    seen_movies = {trainset.to_raw_iid(i[0]) for i in trainset.ur[inner_uid]}

    # Obtener películas no vistas
    unseen_movies = list(set(movies_df["movie_id"]) - seen_movies)

    # Predecir todas las calificaciones no vistas
    predictions = model.test([(user_id, movie, 0) for movie in unseen_movies])

    # Ordenar por calificación predicha
    predictions.sort(key=lambda x: x.est, reverse=True)

    # Seleccionar las mejores películas recomendadas
    recommended_movies = movies_df[movies_df["movie_id"].isin([x.iid for x in predictions

    return recommended_movies


def precision_recall_at_k(predictions, k=5, threshold=4):
    """
    Calcula precisión y recall para un modelo de Surprise basado en calificaciones predic

    Parámetros:
    - predictions: Lista de predicciones generadas por Surprise.
    - k: Número de recomendaciones consideradas.
```

```
    - threshold: Calificación mínima para considerar una película como relevante.

    Retorna:
    - Precisión y Recall promedio.
    """
    user_est_true = defaultdict(list)

    # Agrupar predicciones por usuario
    for uid, _, true_r, est, _ in predictions:
        user_est_true[uid].append((est, true_r))

    precisions = []
    recalls = []

    for uid, user_ratings in user_est_true.items():
        # Ordenar por calificación predicha y seleccionar top-k
        user_ratings.sort(key=lambda x: x[0], reverse=True)
        top_k = user_ratings[:k]

        # Contar aciertos
        relevant = sum((true_r >= threshold) for (_, true_r) in user_ratings)
        hits = sum((true_r >= threshold) for (_, true_r) in top_k)

        # Calcular precisión y recall
        precisions.append(hits / k if k > 0 else 0)
        recalls.append(hits / relevant if relevant > 0 else 0)

    return np.mean(precisions), np.mean(recalls)


# ==============================
# 3. Cargar y Evaluar el Modelo
# ==============================

# Cargar datos en formato Surprise
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(ratings_df[["user_id", "movie_id", "rating"]], reader)

# definir trainset y testset
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)

reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(ratings_df[["user_id", "movie_id", "rating"]], reader)

# Entrenar y guardar el modelo
svd_model, trainset = train_svd_model(data)

print("=" * 50)
print("\n Evaluación del modelo SVD base con conjunto de test\n")

# Realizar predicciones
predictions = svd_model.test(testset)

# Calcular métricas
rmse = accuracy.rmse(predictions, verbose=False)
```

```
mae = accuracy.mae(predictions, verbose=False)
precision, recall = precision_recall_at_k(predictions, k=5, threshold=4)

# Mostrar resultados
print(f"✅ RMSE: {rmse:.4f}")
print(f"✅ MAE: {mae:.4f}")
print(f"✅ Precision@5: {precision:.4f}")
print(f"✅ Recall@5: {recall:.4f}")
```

```
 Entrenando modelo SVD con regularización...

✅ Modelo SVD guardado en: model_svd.pkl

==================================================

 Evaluación del modelo SVD base con conjunto de test

✅ RMSE: 0.5377
✅ MAE: 0.4239
✅ Precision@5: 0.8347
✅ Recall@5: 0.6030
CPU times: user 7.34 s, sys: 52.7 ms, total: 7.4 s
Wall time: 7.5 s
```

```
%%time
# ========================
# Generar recomendaciones
# ========================
Usuario = 50
user_example = ratings_df["user_id"].iloc[Usuario]
recommended_movies_svd = recommend_movies_svd(user_example, svd_model, trainset, movies_d

# Mostrar resultados
print("** PRUEBA DEL MOTOR SVD **")
print(50*"=")
print("\n Recomendaciones para el usuario:", user_example,"\n")
for movie in recommended_movies_svd:
    print(f"- {movie[1]} (ID: {movie[0]})")

# Simulación de usuario nuevo (Partida en Frío)
user_new = -1
print(50*"=")
print("\n **Simulación de usuario nuevo **")
print("\n Partida en Frio => Recomendaciones por popularidad\n")

new_user_recommendations = recommend_movies_svd(user_new, svd_model, trainset, movies_df,
display(new_user_recommendations)
```

```
** PRUEBA DEL MOTOR SVD **
================================================

 Recomendaciones para el usuario: 251

- Silence of the Lambs, The (1991) (ID: 98)
- Godfather, The (1972) (ID: 127)
- Leaving Las Vegas (1995) (ID: 276)
- Schindler's List (1993) (ID: 318)
- Casablanca (1942) (ID: 483)
================================================

 **Simulación de usuario nuevo **

 Partida en Frio => Recomendaciones por popularidad


 Usuario -1 no encontrado. Recomendando películas populares...
```

	movie_id	count	mean_rating	title
0	50	583	4.358491	Star Wars (1977)
1	258	509	3.803536	Contact (1997)
2	100	508	4.155512	Fargo (1996)
3	181	507	4.007890	Return of the Jedi (1983)
4	294	485	3.156701	Liar Liar (1997)

```
CPU times: user 56.1 ms, sys: 953 µs, total: 57 ms
Wall time: 130 ms
```

6.0 Optimización del Modelo SVD con Grid Search y Random Search

En esta sección, optimizamos el modelo de **Factorización de Matrices con SVD**, buscando los mejores hiperparámetros para mejorar la precisión, recall y eficiencia de las recomendaciones.

Estrategias de Búsqueda

Para encontrar la mejor configuración del modelo, utilizamos dos enfoques:

1. **Grid Search** (Búsqueda Exhaustiva)
   - Prueba **todas las combinaciones posibles** de hiperparámetros dentro de un espacio definido.
   - Es **más preciso**, pero puede ser **más lento** si el número de combinaciones es muy grande.
2. **Random Search** (Búsqueda Aleatoria)
   - Explora un **subconjunto aleatorio** del espacio de hiperparámetros.

- Es **más rápido** y escalable en **datasets grandes**, sacrificando algo de precisión en la búsqueda óptima.

Hiperparámetros Optimizados

- `n_factors` : Número de factores latentes (dimensión de embeddings).
- `n_epochs` : Número de épocas de entrenamiento.
- `lr_all` : Tasa de aprendizaje.
- `reg_all` : Parámetro de regularización para evitar overfitting.

Implementación

- **Ejecutamos `GridSearchCV` y `RandomizedSearchCV`** para encontrar los mejores valores de hiperparámetros.
- **Evaluamos los modelos con `RMSE`, `MAE`, `Precision@5` y `Recall@5`**, permitiendo una comparación equilibrada entre precisión y relevancia en recomendaciones.
- **Comparamos los tiempos de ejecución** de cada estrategia para medir la eficiencia computacional.
- **Guardamos los mejores modelos** (`model_grid.pkl` y `model_random.pkl`) para futuras predicciones sin necesidad de reentrenar.

Resultados

- **Grid Search** encuentra la configuración óptima con **alta precisión**, pero **requiere más tiempo de cómputo**.
- **Random Search** logra una calidad de modelo **similar**, pero en **menos tiempo**, siendo una opción viable para escenarios con limitaciones computacionales.
- Ambas estrategias **superan al modelo base de SVD**, validando la importancia de la optimización de hiperparámetros.

Esta optimización nos permite **seleccionar el mejor modelo para la evaluación comparativa final**, asegurando recomendaciones más precisas y eficientes.

```
# ============================================================
# 6. Ajuste de Hiperparámetros (Grid Search y Random Search)
# ============================================================

# ------------------------------------------------------------
# Funciones para Evaluar Modelos (SVD, Grid, Random)
# ------------------------------------------------------------

def load_model(file_path):
    """
    Carga un modelo o diccionario de métricas desde un archivo pickle.

    Parámetros:
    - file_path: Ruta del archivo pickle.
```

```
    Retorna:
    - Objeto cargado desde el pickle (puede ser un modelo o un diccionario de métricas).
    - Si el archivo no existe, retorna None.
    """
    if os.path.exists(file_path):
        with open(file_path, "rb") as f:
            return pickle.load(f)
    return None

def save_model(obj, file_path):
    """
    Guarda un objeto en un archivo pickle.

    Parámetros:
    - obj: Objeto a guardar.
    - file_path: Ruta donde guardar el archivo pickle.
    """
    with open(file_path, "wb") as f:
        pickle.dump(obj, f)

def train_and_evaluate(model, trainset, testset, model_name="Modelo"):
    """
    Entrena un modelo de Surprise y lo evalúa en términos de RMSE, MAE, Precisión y Recal

    Parámetros:
    - model: Modelo de Surprise a entrenar.
    - trainset: Conjunto de entrenamiento de Surprise.
    - testset: Conjunto de prueba de Surprise.
    - model_name: Nombre del modelo para la impresión de resultados.

    Retorna:
    - Modelo entrenado.
    - RMSE, MAE, Precisión@5 y Recall@5.
    """
    print(f"\n🚀 Entrenando y Evaluando {model_name}...")

    # Entrenar el modelo
    model.fit(trainset)

    # Realizar predicciones
    predictions = model.test(testset)

    # Calcular métricas
    rmse = accuracy.rmse(predictions, verbose=False)
    mae = accuracy.mae(predictions, verbose=False)
    precision, recall = precision_recall_at_k(predictions, k=5, threshold=4)

    print(f"📊 {model_name}: RMSE = {rmse:.4f}, MAE = {mae:.4f}, Precision@5 = {precisior

    return model, rmse, mae, precision, recall

# ----------------------------------------------------------
# Función General para Evaluar Modelos (SVD, Grid, Random)
# ----------------------------------------------------------
def evaluate_surprise_model(model, testset, model_name="Modelo"):
```

```
    """
    Evalúa un modelo de Surprise en términos de RMSE y MAE.

    Parámetros:
    - model: Modelo entrenado de Surprise.
    - testset: Conjunto de prueba.
    - model_name: Nombre del modelo para impresión.

    Retorna:
    - RMSE y MAE del modelo.
    """
    predictions = model.test(testset)
    rmse = accuracy.rmse(predictions, verbose=False)
    mae = accuracy.mae(predictions, verbose=False)
    precision, recall = precision_recall_at_k(predictions, k=5, threshold=4)

    print(f"\n📊 Evaluación de {model_name}: RMSE = {rmse:.4f}, MAE = {mae:.4f}, Precisio

    return rmse, mae, precision, recall



%%time
# ============================================================
# Cargar métricas previas o inicializar un nuevo diccionario
# ============================================================
metrics_results = load_model("metrics.pkl") or {}

# Asegurar que Grid Search y Random Search existen en metrics_results
default_metrics = {"RMSE": None, "MAE": None, "Precision": None, "Recall": None, "Coverag
metrics_results.setdefault("Grid Search", default_metrics.copy())
metrics_results.setdefault("Random Search", default_metrics.copy())

# ============================================================
# 6 Grid Search (Búsqueda Exhaustiva)
# ============================================================
print("\n🔍 Ejecutando Grid Search...")
param_grid = {
    'n_factors': [50, 100, 150],  # Se amplía el rango de factores latentes
    'n_epochs': [20, 30, 40],  # Se aumenta el número de épocas para asegurar convergenci
    'lr_all': [0.002, 0.005, 0.01],  # Se incluyen valores conservadores de learning rate
    'reg_all': [0.02, 0.05, 0.1]  # Regularización para evitar sobreajuste
}

# Ejecutar Grid Search
gs = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=3)
gs.fit(data)
best_model_grid = gs.best_estimator['rmse']

# Entrenar y evaluar el mejor modelo encontrado con Grid Search
best_model_grid, rmse_grid, mae_grid, precision_grid, recall_grid = train_and_evaluate(be

# Guardar modelo entrenado y métricas
save_model(best_model_grid, "model_grid.pkl")
metrics_results["Grid Search"].update({
```

```
    "RMSE": rmse_grid,
    "MAE": mae_grid,
    "Precision": precision_grid,
    "Recall": recall_grid
})

print(" Mejores parámetros con Grid Search:", gs.best_params['rmse'])
print(f"📊 RMSE: {rmse_grid:.4f} | MAE: {mae_grid:.4f}")
```

```
🔍 Ejecutando Grid Search...

🚀 Entrenando y Evaluando Grid Search...
📊 Grid Search: RMSE = 0.7118, MAE = 0.5648, Precision@5 = 0.8077, Recall@5 = 0.5885
 Mejores parámetros con Grid Search: {'n_factors': 150, 'n_epochs': 40, 'lr_all': 0.0
📊 RMSE: 0.7118 | MAE: 0.5648
CPU times: user 34min 24s, sys: 3.24 s, total: 34min 28s
Wall time: 34min 40s
```

```
%%time
# ============================================================
# 6 Random Search (Búsqueda Aleatoria)
# ============================================================
import random
random.seed(42)
np.random.seed(42)


print("\n Ejecutando Random Search...")
param_distributions = {
    'n_factors': [50, 75, 100, 125, 150, 175, 200],
    'n_epochs': [10, 20, 30, 40, 50, 60],
    'lr_all': [0.001, 0.002, 0.005, 0.007, 0.01, 0.015, 0.02],
    'reg_all': [0.01, 0.02, 0.03, 0.05, 0.07, 0.1, 0.15, 0.2]
}

rs = RandomizedSearchCV(
    SVD,
    param_distributions,
    n_iter=20,
    measures=['rmse', 'mae'],
    cv=3,
    random_state=42,
    n_jobs=1                        # Para reproducibilidad total
)
rs.fit(data)
best_model_random = rs.best_estimator['rmse']

# Entrenar y evaluar el mejor modelo encontrado con Random Search
best_model_random, rmse_random, mae_random, precision_random, recall_random = train_and_e

# Guardar modelo entrenado y métricas
save_model(best_model_random, "model_random.pkl")
metrics_results["Random Search"].update({
```

```
    "RMSE": rmse_random,
    "MAE": mae_random,
    "Precision": precision_random,
    "Recall": recall_random
})

print(f"\n✅ Mejores parámetros con Random Search: {rs.best_params['rmse']}")
print(f" RMSE: {rmse_random:.4f} | MAE: {mae_random:.4f}")

# -----------------
# Guardar métricas en Pickle
# -----------------
save_model(metrics_results, "metrics.pkl")
print("\n ✅ Todas las métricas guardadas correctamente.\n")
```

```
 Ejecutando Random Search...

🚀 Entrenando y Evaluando Random Search...
📊 Random Search: RMSE = 0.6862, MAE = 0.5426, Precision@5 = 0.8143, Recall@5 = 0.592

✅ Mejores parámetros con Random Search: {'n_factors': 150, 'n_epochs': 40, 'lr_all'
 RMSE: 0.6862 | MAE: 0.5426

 ✅ Todas las métricas guardadas correctamente.

CPU times: user 12min 19s, sys: 2.2 s, total: 12min 21s
Wall time: 12min 25s
```

7.0 Evaluacion Comparativa de Modelos

En esta sección se evalúan cuantitativamente los distintos modelos de recomendación implementados mediante un conjunto de métricas estándar:

- **RMSE (Root Mean Squared Error)**: mide la precisión de las predicciones numéricas de calificación.
- **MAE (Mean Absolute Error)**: evalúa el error promedio absoluto.
- **Precisión@5 y Recall@5**: miden la relevancia de las recomendaciones dentro del Top-5 sugerido.

Se evalúan los siguientes modelos:

- **SVD Base**: modelo con configuración inicial sin ajuste fino.
- **SVD Optimizado (Grid Search y Random Search)**: modelos ajustados mediante búsqueda de hiperparámetros.
- **Similitud del Coseno**: evaluado con métricas de ranking (Precisión y Recall), ya que no genera predicciones numéricas.

Los resultados se consolidan en una tabla comparativa y se visualizan mediante gráficos para analizar el comportamiento relativo de cada modelo en términos de **precisión, relevancia y**

estabilidad.

Esta evaluación permite identificar el modelo más robusto y justificar la elección del sistema de recomendación final.

```
# ============================================================
# Función General para Evaluar Modelos (SVD, Grid, Random)
# ============================================================
def evaluate_surprise_model(model, testset, model_name="Modelo"):
    """
    Evalúa un modelo de Surprise en términos de RMSE y MAE.

    Parámetros:
    - model: Modelo entrenado de Surprise.
    - testset: Conjunto de prueba.
    - model_name: Nombre del modelo para impresión.

    Retorna:
    - RMSE y MAE del modelo.
    """
    predictions = model.test(testset)
    rmse = accuracy.rmse(predictions, verbose=False)
    mae = accuracy.mae(predictions, verbose=False)
    precision, recall = precision_recall_at_k(predictions, k=5, threshold=4)

    print(f"\n Evaluación de {model_name}: RMSE = {rmse:.4f}, MAE = {mae:.4f}, Precision@

    return rmse, mae, precision, recall
```

```
%%time

# 1Cargar métricas previas
metrics_results = load_model("metrics.pkl") or {}

# -----------------------
# 2. Evaluar modelos
# -----------------------
print("\n🚀 Evaluando Modelos de Recomendación...\n")

results = {}

models_to_evaluate = {
    "SVD Base": svd_model,
    "Grid Search": best_model_grid,
    "Random Search": best_model_random
}

for name, model in models_to_evaluate.items():
    rmse, mae, precision, recall = evaluate_surprise_model(model, testset, name)
    results[name] = {
        "RMSE": rmse,
        "MAE": mae,
        "Precisión @5": precision,
```

```
        "Recall @5": recall
    }

# Evaluación para Similitud del Coseno
precision_cosine, recall_cosine = compute_precision_recall_at_k(
    movie_similarity, ratings_df, movies_df, k=5
)
results["Similitud del Coseno"] = {
    "RMSE": None,
    "MAE": None,
    "Precisión @5": precision_cosine,
    "Recall @5": recall_cosine
}

# ====================================
# 3. Convertir a DataFrame
# ====================================
comparison_df = pd.DataFrame(results).T.reset_index().rename(columns={"index": "Modelo"})

print("\n📊 Comparación de Modelos:\n")
display(comparison_df)

# ====================================
# 4. Visualización elegante con seaborn
# ====================================
sns.set(style="whitegrid", palette="pastel", font_scale=1.1)

fig, axes = plt.subplots(1, 2, figsize=(14, 6))
fig.suptitle("Comparación de Modelos de Recomendación", fontsize=16, fontweight="bold")

# A. Errores RMSE / MAE
df_error = comparison_df[["Modelo", "RMSE", "MAE"]].dropna()
df_error_melted = df_error.melt(id_vars="Modelo", var_name="Métrica", value_name="Valor")
sns.barplot(data=df_error_melted, x="Modelo", y="Valor", hue="Métrica", ax=axes[0])
axes[0].set_title("Error de Predicción (RMSE y MAE)")
axes[0].set_ylabel("Error")
axes[0].tick_params(axis='x', rotation=30)

# B. Precisión y Recall
df_pr = comparison_df[["Modelo", "Precisión @5", "Recall @5"]]
df_pr_melted = df_pr.melt(id_vars="Modelo", var_name="Métrica", value_name="Valor")
sns.barplot(data=df_pr_melted, x="Modelo", y="Valor", hue="Métrica", ax=axes[1])
axes[1].set_title("Precisión y Recall @5")
axes[1].set_ylabel("Score")
axes[1].tick_params(axis='x', rotation=30)

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

🚀 Evaluando Modelos de Recomendación...

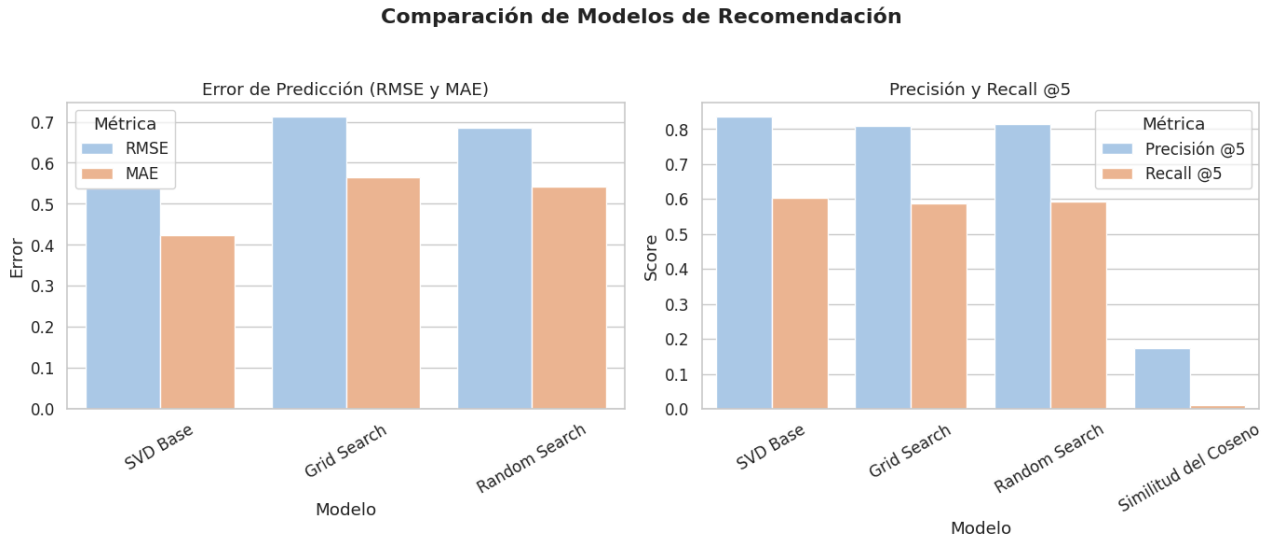
Evaluación de SVD Base: RMSE = 0.5377, MAE = 0.4239, Precision@5 = 0.8347, Recall@

Evaluación de Grid Search: RMSE = 0.7118, MAE = 0.5648, Precision@5 = 0.8077, Reca

Evaluación de Random Search: RMSE = 0.6862, MAE = 0.5426, Precision@5 = 0.8143, Re

📊 Comparación de Modelos:

	Modelo	RMSE	MAE	Precisión @5	Recall @5
0	SVD Base	0.537707	0.423921	0.834681	0.603018
1	Grid Search	0.711818	0.564840	0.807660	0.588538
2	Random Search	0.686211	0.542575	0.814255	0.592397
3	Similitud del Coseno	NaN	NaN	0.173277	0.009841



CPU times: user 4.8 s, sys: 291 ms, total: 5.09 s
Wall time: 4.8 s

Análisis Comparativo de Modelos

A continuación se presenta una comparación entre los diferentes modelos evaluados, considerando métricas de error (RMSE y MAE) como referencia, y métricas de evaluación específicas de sistemas de recomendación (Precisión y Recall en el top 5), que son las más relevantes en este contexto.

Modelo	RMSE	MAE	Precisión @5	Recall @5
SVD Base	0.537707	0.423921	0.834681	0.603018
Grid Search	0.711818	0.564840	0.807660	0.588538
Random Search	0.686211	0.542575	0.814255	0.592397
Similitud del Coseno	NaN	NaN	0.173277	0.009841

Observaciones

1. **Importancia de Precisión y Recall**:
    - En sistemas de recomendación, las métricas clave para evaluar la utilidad real del modelo son **Precisión@5** y **Recall@5**, ya que reflejan cuán relevantes son las recomendaciones entregadas al usuario.
    - El modelo **SVD Base** muestra el mejor desempeño en ambas métricas, lo que indica una excelente capacidad para seleccionar recomendaciones acertadas dentro del top 5.
    - **Random Search** obtiene un rendimiento relativamente cercano en precisión y recall, aunque sin superarlo, mientras que **Grid Search** presenta una leve degradación en ambas métricas.
2. **Modelos optimizados con Grid y Random Search**:
    - A pesar de aplicar técnicas estándar de optimización de hiperparámetros, **ninguno de los modelos optimizados logró superar al modelo SVD base**.
    - Este resultado, aunque inesperado, es posible. Puede deberse a una configuración subóptima del espacio de búsqueda o a que los valores por defecto del algoritmo ya están muy bien ajustados al dataset utilizado.
    - Este fenómeno justifica un análisis más profundo de la superficie de error y de las combinaciones de parámetros exploradas, lo cual se dejará como tarea para una etapa posterior.
3. **Métricas de error (RMSE y MAE)**:
    - Estas métricas sirven como referencia para evaluar la capacidad de predicción de ratings, pero **no son las más indicadas para medir la calidad de las recomendaciones** en términos de relevancia para el usuario.
    - Aun así, se observa un aumento en el error en los modelos optimizados, lo cual coincide con la ligera caída en las métricas de recomendación.

4. **Similitud del Coseno (basado en ítems)**:
    - Este enfoque no genera predicciones numéricas, por lo que no reporta RMSE ni MAE.
    - Su rendimiento en precisión y recall es muy bajo, evidenciando que **no es competitivo** frente a los modelos basados en factorización. Actúa como un punto de comparación básico, pero no resulta útil en entornos de recomendación exigentes.

Conclusión

El análisis evidencia que el modelo SVD base es, en este conjunto de datos, el más efectivo tanto en términos de precisión como de recuperación de ítems relevantes. El hecho de que los procesos de optimización no hayan mejorado su rendimiento requiere una revisión más detallada del espacio de búsqueda, así como de los criterios de evaluación utilizados durante la optimización. La prioridad, en sistemas de recomendación, debe centrarse en **maximizar la precisión y el recall**, ya que son métricas directamente asociadas a la experiencia del usuario.

8.0 DEMOSTRACIÓN DE USO DE LOS MOTORES DE RECOMENDACIÓN

Esta sección muestra el uso práctico de los motores de recomendación desarrollados para generar sugerencias personalizadas a partir del ID de un usuario.

Se utiliza un usuario aleatorio del dataset y se generan dos conjuntos de recomendaciones:

- 🔷 **Modelo SVD Optimizado**: recomienda películas basadas en patrones latentes aprendidos durante el entrenamiento.
- 🔷 **Similitud del Coseno (Ítem-Ítem)**: recomienda películas similares a la última vista por el usuario, evitando repetir contenido ya visto.

La salida incluye los títulos recomendados por cada método, permitiendo comparar rápidamente sus comportamientos. Esta demostración valida que ambos motores funcionan correctamente y pueden integrarse en una interfaz de recomendación real.

Si el usuario no tiene historial (partida en frío), el sistema entrega recomendaciones basadas en popularidad.

```
def show_recommendations(user_id, svd_model, trainset, similarity_matrix, ratings_df, mov
    """
    Muestra recomendaciones de películas para un usuario, usando SVD y Similitud del Cose

    Parámetros:
    - user_id: ID del usuario.
    - svd_model: Modelo SVD entrenado.
    - trainset: Conjunto de entrenamiento Surprise.
```

```
    - similarity_matrix: Matriz de similitud entre películas (DataFrame).
    - ratings_df: Calificaciones.
    - movies_df: Información de películas.
    - top_n: Número de recomendaciones.
    """

    print("=" * 70)
    print(f"\n🎯 Recomendaciones para el Usuario {user_id}\n")

    # ====================================
    # 1️⃣ Recomendaciones SVD
    # ====================================
    recommended_svd = recommend_movies_svd(user_id, svd_model, trainset, movies_df, top_n

    print("🔷 Recomendaciones usando SVD:")
    if recommended_svd:
        for movie in recommended_svd:
            print(f"- {movie[1]} (ID: {movie[0]})")
    else:
        print("- Sin recomendaciones disponibles.")

    # ====================================
    # 2️⃣ Recomendaciones por Similitud del Coseno
    # ====================================
    user_ratings = ratings_df[ratings_df["user_id"] == user_id].sort_values(by="timestamp

    if not user_ratings.empty:
        last_movie_id = user_ratings.iloc[0]["movie_id"]
        recommended_cosine = recommend_movies(
            movie_id=last_movie_id,
            similarity_matrix=similarity_matrix,
            movies_df=movies_df,
            ratings_df=ratings_df,
            user_id=user_id,
            top_n=top_n
        )
        print("\n🔷 Recomendaciones usando Similitud del Coseno:")
        if recommended_cosine:
            for movie in recommended_cosine:
                print(f"- {movie[1]} (ID: {movie[0]})")
        else:
            print("- Sin recomendaciones disponibles.")
    else:
        print("\n🔷 Recomendaciones usando Similitud del Coseno:")
        print("- Usuario sin historial. No se pueden generar recomendaciones basadas en c

    print("=" * 70)

# ====================================
# 👉 Ejemplo
# ====================================

random_user = np.random.choice(ratings_df["user_id"].unique())
show_recommendations(random_user, svd_model, trainset, movie_similarity, ratings_df, movi
```

```
======================================================================

🎯 Recomendaciones para el Usuario 379

🔷 Recomendaciones usando SVD:
- Wizard of Oz, The (1939) (ID: 132)
- Henry V (1989) (ID: 190)
- Contact (1997) (ID: 258)
- Chasing Amy (1997) (ID: 268)
- Sense and Sensibility (1995) (ID: 275)
- Schindler's List (1993) (ID: 318)
- Close Shave, A (1995) (ID: 408)
- Manchurian Candidate, The (1962) (ID: 657)
- City of Lost Children, The (1995) (ID: 919)
- Some Folks Call It a Sling Blade (1993) (ID: 963)

🔷 Recomendaciones usando Similitud del Coseno:
- Four Rooms (1995) (ID: 3)
- Copycat (1995) (ID: 5)
- Shanghai Triad (Yao a yao yao dao waipo qiao) (1995) (ID: 6)
- Richard III (1995) (ID: 10)
- Seven (Se7en) (1995) (ID: 11)
- Mighty Aphrodite (1995) (ID: 13)
- Postino, Il (1994) (ID: 14)
- Mr. Holland's Opus (1995) (ID: 15)
- French Twist (Gazon maudit) (1995) (ID: 16)
- From Dusk Till Dawn (1996) (ID: 17)
======================================================================
```

9.0 ANÁLISIS COMPARATIVO DE MOTORES DE RECOMENDACIÓN

En esta sección se realiza un análisis cualitativo entre los motores de recomendación implementados:

- **SVD Optimizado (Random Search)**
- **Similitud del Coseno Ítem-Ítem**

Para un usuario dado, se comparan las recomendaciones generadas por ambos modelos en cuanto a:

1. **Diversidad de resultados**: número de películas coincidentes entre ambos modelos.
2. **Repetición de contenido**: detección de películas ya vistas por el usuario dentro de las recomendaciones.
3. **Popularidad promedio**: número de calificaciones que ha recibido cada película recomendada (como proxy de visibilidad).
4. **Dispersión de calificaciones**: desvío estándar de los ratings para medir estabilidad y consenso en la percepción de las películas.

Además, se presentan gráficos comparativos que muestran visualmente las diferencias en **popularidad** y **variabilidad** de las películas recomendadas por cada motor.

Este análisis permite evaluar no solo la precisión técnica, sino también la **diversidad, originalidad y estabilidad** del sistema de recomendación, aspectos clave para la experiencia del usuario.

```
def analyze_recommendations(user_id, svd_model, similarity_matrix, ratings_df, movies_df,
    """
    Analiza y compara las recomendaciones entregadas por SVD (modelo optimizado) y Simili

    Parámetros:
    - user_id: ID del usuario.
    - svd_model: Modelo SVD optimizado (ej: Random Search).
    - similarity_matrix: Matriz de similitud entre películas.
    - ratings_df: DataFrame con calificaciones.
    - movies_df: DataFrame con información de películas.
    - trainset: Objeto Surprise con datos de entrenamiento.
    - top_n: Número de recomendaciones.

    Retorna:
    - Análisis comparativo impreso y gráficos.
    """

    print(f"\n🔷 Análisis de Recomendaciones para el Usuario {user_id} 🔷\n")

    # 1️⃣ Recomendaciones con SVD (modelo optimizado)
    recommended_svd = recommend_movies_svd(user_id, svd_model, trainset, movies_df, top_n

    # 2️⃣ Recomendaciones por Similitud del Coseno
    user_ratings = ratings_df[ratings_df["user_id"] == user_id].sort_values(by="timestamp

    if not user_ratings.empty:
        last_movie_id = user_ratings.iloc[0]["movie_id"]
        recommended_cosine = recommend_movies(
            movie_id=last_movie_id,
            similarity_matrix=similarity_matrix,
            movies_df=movies_df,
            ratings_df=ratings_df,
            user_id=user_id,
            top_n=top_n,
            verbose=False
        )
    else:
        recommended_cosine = []

    # Convertir listas a DataFrame
    svd_df = pd.DataFrame(recommended_svd, columns=["movie_id", "title"])
    cosine_df = pd.DataFrame(recommended_cosine, columns=["movie_id", "title"])

    #  Diversidad (intersección)
    common = set(svd_df["movie_id"]) & set(cosine_df["movie_id"])
    print(f" Películas recomendadas en ambos modelos: {len(common)} de {top_n}")
```

```
    #  Películas ya vistas
    seen_movies = set(user_ratings["movie_id"])
    svd_seen = len(set(svd_df["movie_id"]) & seen_movies)
    cosine_seen = len(set(cosine_df["movie_id"]) & seen_movies)
    print(f" Películas ya vistas recomendadas por SVD: {svd_seen}")
    print(f" Películas ya vistas recomendadas por Coseno: {cosine_seen}")

    #  Popularidad
    popularity = ratings_df["movie_id"].value_counts()
    svd_df["popularity"] = svd_df["movie_id"].map(popularity)
    cosine_df["popularity"] = cosine_df["movie_id"].map(popularity)
    print(f"\n Promedio de popularidad:")
    print(f"- SVD: {svd_df['popularity'].mean():.2f} calificaciones")
    print(f"- Coseno: {cosine_df['popularity'].mean():.2f} calificaciones")

    #  Variabilidad
    rating_std = ratings_df.groupby("movie_id")["rating"].std()
    svd_df["rating_std"] = svd_df["movie_id"].map(rating_std)
    cosine_df["rating_std"] = cosine_df["movie_id"].map(rating_std)
    print(f"\n📊 Dispersión de calificaciones:")
    print(f"- SVD: {svd_df['rating_std'].mean():.2f}")
    print(f"- Coseno: {cosine_df['rating_std'].mean():.2f}")

    #  Visualización
    fig, axes = plt.subplots(1, 2, figsize=(12, 5))
    full_df = pd.concat([
        svd_df.assign(Modelo="SVD Optimizado"),
        cosine_df.assign(Modelo="Similitud Coseno")
    ])

    sns.barplot(data=full_df, x="Modelo", y="popularity", ax=axes[0])
    axes[0].set_title("Popularidad Promedio de Recomendaciones")
    axes[0].set_ylabel("Número de Calificaciones")

    sns.barplot(data=full_df, x="Modelo", y="rating_std", ax=axes[1])
    axes[1].set_title("Dispersión de Calificaciones en Recomendaciones")
    axes[1].set_ylabel("Desvío Estándar")

    plt.tight_layout()
    plt.show()

# ====================================

random_user = np.random.choice(ratings_df["user_id"].unique())
analyze_recommendations(random_user, best_model_random, movie_similarity, ratings_df, mov
```

🔷 Análisis de Recomendaciones para el Usuario 507 🔷

Películas recomendadas en ambos modelos: 0 de 5
Películas ya vistas recomendadas por SVD: 0
Películas ya vistas recomendadas por Coseno: 0

Promedio de popularidad:
- SVD: 268.60 calificaciones
- Coseno: 193.60 calificaciones

📊 Dispersión de calificaciones:
- SVD: 0.87
- Coseno: 1.00