Clase 216 — Filtrado colaborativo: user-based e item-based

Parte: 6 — Sistemas de Recomendación · Fuente: Aggarwal, Recommender Systems: The Textbook (Springer, 2016) cap. 2 + Linden, Smith, York (Amazon, 2003) Item-to-Item Collaborative Filtering. Duración estimada: 75 min.

Clase 216 — Filtrado colaborativo: user-based e item-based

> *Parte: 6 — Sistemas de Recomendación · Fuente: Aggarwal, Recommender Systems: The Textbook (Springer, 2016) cap. 2 + Linden, Smith, York (Amazon, 2003) Item-to-Item Collaborative Filtering. Duración estimada: 75 min.*

Objetivo

Construir el recomendador más antiguo y todavía usado: filtrado colaborativo basado en vecinos (kNN). Calcular similitudes user-user e item-item sobre una matriz usuario-item dispersa, generar top-N recomendaciones, y entender por qué Amazon publicó en 2003 que item-based gana a user-based en escala (computar similitudes item-item es offline y estable).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Representar las interacciones usuario-item en una scipy.sparse.csr_matrix (típicamente >99% sparse).
- Calcular similitud coseno, Pearson y Jaccard entre filas (users) o columnas (items).
- Generar top-N recomendaciones user-based: predicted_rating = Σ sim(u, v) * rating(v, i) / Σ sim(u, v).
- Generar top-N recomendaciones item-based: score(u, i) = Σ sim(i, j) * interaction(u, j).
- Reconocer límites: sparsity → similitudes ruidosas; cold-start → items/users nuevos sin recomendaciones (Clase 221).

Temas

#	Tema	Por qué importa
1	Matriz usuario-item: dense vs sparse	1M users × 100K items → 100B celdas; 99.9%
2	Similitud coseno, Pearson, Jaccard	La elección define qué entiende el modelo
3	User-based kNN	Intuitivo; no escala (similitudes user-use
4	Item-based kNN	El paper de Amazon: similitudes item-item
5	Implicit vs explicit feedback	Rating 1-5 vs "vio/no vio". Cambia loss y
6	Mean centering	Restar user_mean antes de calcular similit

Definiciones y características

- Matriz usuario-item R[u, i]: rating del user u sobre item i. Para implicit: 1 si interactuó, 0 si no.
- Sparse matrix: solo almacenan los valores no-cero. csr_matrix (Compressed Sparse Row) — eficiente para acceso por fila. csc_matrix por columna.
- Similitud coseno: cos(u, v) = (u · v) / (|u| × |v|). Insensible a la magnitud — un usuario que rateó muchas pelis ≈ uno que rateó pocas si los patrones coinciden.
- Correlación Pearson: similar a coseno pero resta la media antes — controla el sesgo del usuario (algunos califican 5 todo, otros 3 todo).
- Jaccard: para datos binarios. |A ∩ B| / |A B|. Útil con implicit feedback.
- User-based prediction: r̂(u, i) = Σ_v sim(u, v) × r(v, i) / Σ_v |sim(u, v)| con v vecinos del top-K más

similares a u que rateó i.
- Item-based prediction: r̂(u, i) = Σ_j sim(i, j) × r(u, j) / Σ_j |sim(i, j)| con j items más similares a i que u rateó.
- Implicit feedback: el dato es "interactuó/no interactuó" (vio, clickeó, compró). No hay rating explícito; el modelo no debe asumir 0 = dislike.

Dataset / recursos

- Dataset: MovieLens 100K (ml-100k, ~1 MB, ~100K ratings de 943 users × 1682 movies). El clásico para empezar.
- Librerías: scipy.sparse, numpy, pandas, scikit-learn (para cosine_similarity).

Ejercicios

1. Sparse matrix: cargar MovieLens 100K. Construir R como csr_matrix shape (n_users, n_items). Verificar sparsity: (1 - R.nnz / np.prod(R.shape)).
2. User similarity: sim_users = cosine_similarity(R) — devuelve (n_users, n_users). Encontrar los 5 más similares a user_id=42.
3. User-based top-10: para user_id=42, predecir score para items no vistos como R.T @ sim_users[42] (broadcasting). Recomendar top-10 que aún no vio.
4. Item-based top-10: sim_items = cosine_similarity(R.T) (ahora (n_items, n_items)). Para user_id=42, score = R[42] @ sim_items. Mostrar top-10.
5. Pearson + mean centering: R_centered = R - user_means.reshape(-1, 1) (cuidado con sparse — usar sklearn). Comparar recomendaciones con coseno vanilla.

Homework verificable

Notebook con:

1. Cargar MovieLens 1M (10× más grande que 100K).
2. Implementar 2 recomendadores: user-based kNN (k=30) e item-based kNN (k=30).
3. Split train/test con leave-one-out por user (el último rating de cada user va a test).
4. Reportar MAP@10, NDCG@10, recall@10 para ambos sobre test (Clase 220).
5. Comparativa: tiempo de entrenamiento, tiempo de predicción, calidad. Discutir por qué item-based suele ganar en producción.

Criterio de aceptación: ambos recomendadores entrenan en <5 min sobre 1M, recall@10 > 0.05, y el estudiante justifica la elección item-based para producción.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
MemoryError al construir n_users × n_users	Estás densificando. Fix: usar sklearn.metr
Recomendaciones son siempre los items popu	Sin top-K filter de neighbors, dominan ite
Recall=0.0 sobre test	Estás recomendando items que el user ya vi
Pearson sobre rows con poca overlap da nan	Si 2 users tienen <3 items en común, Pears
Item-based sim_matrix de 100K × 100K	Densa = 80 GB. Fix: top-K nearest neighbor
Implicit feedback con coseno se sesga a it	Coseno normaliza, da peso a items raros. F

Preguntas frecuentes

¿User-based o item-based?

Item-based en producción (Amazon, 2003). Razones:

- Similitudes item-item cambian lento (un item es estable; un nuevo rating no mueve mucho). Pre-computable offline.
- Similitudes user-user cambian con cada rating nuevo del user.
- N items << N users en muchos casos (productos vs millones de clientes).

¿cosine_similarity o pearson_correlation?

Coseno por default. Pearson cuando el sesgo absoluto del usuario importa (algunos califican alto todo, otros bajo todo) — frecuente en sistemas de rating 1-5 explícito.

¿Implicit feedback se trata igual?

No. Con implicit, R[u, i] = 0 NO significa "dislike" — puede significar "no lo vio aún". Estrategias: BPR (Bayesian Personalized Ranking), ALS implicit con confidence weighting (Clase 217), o métricas top-N (NDCG@k) en vez de RMSE.

¿Cuándo CF se rompe?

(1) Cold-start total: user/item nuevo con 0 historia (Clase 221). (2) Sparsity extremo (>99.99%): similitudes son ruidosas. (3) Long-tail dominante: 1% de items reciben 80% del tráfico, recomendador converge a "los populares".

¿Surprise/LightFM/Implicit hacen esto por mí?

Sí (Clase 222). Surprise tiene KNNBasic/KNNWithMeans. Implicit es especialista en implicit feedback. LightFM combina CF + content (Clase 219).

¿Y si tengo 100M users × 10M items?

kNN no escala. Saltá directo a factorización de matrices con ALS distribuido (Spark ALS o Implicit als.AlternatingLeastSquares), o a embeddings deep (TwoTowers, BERT4Rec).

Referencias

- Aggarwal, C. Recommender Systems: The Textbook (Springer, 2016), cap. 2 — Neighborhood-Based Collaborative Filtering.
- Linden, G., Smith, B., York, J. Amazon.com Recommendations: Item-to-Item Collaborative Filtering (IEEE Internet Computing, 2003) — el paper fundacional.
- MovieLens datasets.
- sklearn.metrics.pairwise.cosine_similarity — con dense_output=False para sparse.
- Programming Collective Intelligence (Segaran, 2007) — introducción accesible.

Siguiente clase

Clase 217 — Factorización de matrices: SVD, ALS

Apéndice: notebook (primer bloque)

Sintético small (replica MovieLens 100K en tamaño). Requiere: pip install scipy scikit-learn pandas.

```
import numpy as np, pandas as pd
from scipy.sparse import csr_matrix
from sklearn.metrics.pairwise import cosine_similarity

rng = np.random.default_rng(42)
n_users, n_items = 1000, 500
n_ratings = 30_000   # ~6% sparsity

users = rng.integers(0, n_users, n_ratings)
items = rng.integers(0, n_items, n_ratings)
# Ratings sesgados por user (algunos califican alto, otros bajo)
user_bias = rng.normal(0, 0.5, n_users)
ratings = np.clip(3 + user_bias[users] + rng.normal(0, 0.8, n_ratings), 1, 5).round()

df = pd.DataFrame({'u': users, 'i': items, 'r': ratings}).drop_duplicates(['u', 'i'])
print(f'ratings únicos: {len(df):,}')

R = csr_matrix((df.r, (df.u, df.i)), shape=(n_users, n_items))
print(f'matrix: {R.shape}, nnz={R.nnz:,}, sparsity={1 - R.nnz / np.prod(R.shape):.4f}')
```

Archivos complementarios

- notebook.ipynb