Parte 6 — Sistemas de Recomendación

7 clases · Parte 6 del programa

Parte 6 — Sistemas de Recomendación

> *7 clases · bundle consolidado del currículo v3.*

Índice de clases

Clase 216 — 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.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 217 — Clase 217 — Factorización de matrices: SVD, ALS

> *Parte: 6 — Sistemas de Recomendación · Fuente: Koren, Bell, Volinsky Matrix Factorization Techniques for Recommender Systems (IEEE Computer, 2009) + Hu, Koren, Volinsky Collaborative Filtering for Implicit Feedback Datasets (ICDM 2008). Duración estimada: 80 min.*

Objetivo

Reemplazar la matriz usuario-item dispersa por dos matrices densas de baja dimensión: R ≈ P × Q^T donde P (n_users × k) y Q (n_items × k). Aprender los embeddings k-dimensionales que capturan los factores latentes (género de película, gusto del usuario). Usar SVD (cuando hay datos completos) y ALS (Alternating Least Squares — robusto a sparse). Implicit-feedback ALS (Hu et al. 2008) es el algoritmo que ganó la mayoría de los Netflix Prize spin-offs en producción.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Explicar el modelo r̂(u, i) = p_u · q_i + b_u + b_i + μ (con biases).
- Aplicar SVD truncado sobre matriz densa (poco realista, pero base teórica).
- Implementar ALS explicit (rating predicho) y ALS implicit (con confidence weighting c_ui = 1 + α × r_ui).
- Elegir hiperparámetros: factors (típicamente 20-200), regularization (λ para evitar overfitting), iterations (10-30).
- Usar implicit.AlternatingLeastSquares (Cython, multi-thread, rápido).

Temas

#	Tema	Por qué importa				
1	Modelo latente: fac	"Acción", "drama",				
2	SVD vs SVD trunca	SVD asume matriz				
3	Biases: μ, b_u, b_i	"Usuarios duros" y				
4	Implicit feedback +	0 no es "dislike"; p				
5	Regularización L2	`λ × (	p	² +	q	²) para evitar overfitting con k` alto
6	Cold-start parcial: c	Para user nuevo: r̂				

Definiciones y características

- Factor latente: dimensión del embedding que captura una variable no observada (género, mood, demographic).
- SVD truncado: descomposición R = U Σ V^T quedándose con top-k singular values. Asume R completa (no es nuestro caso).
- ALS (Alternating Least Squares): fija Q, resuelve P por LSQ → fija P, resuelve Q → iterá. Cada subproblema es lineal y paralelizable.
- Modelo con biases: r̂(u, i) = μ + b_u + b_i + p_u · q_i. μ es la media global; b_u/b_i capturan el sesgo

del usuario/item.

- Implicit feedback ALS: en vez de minimizar error sobre ratings observados, minimiza sobre TODA la matriz, ponderando por confidence c_ui = 1 + α × r_ui. Items con interacción tienen alta confianza de "1" (preferencia); sin interacción tienen confianza baja de "0".
- Regularización: L = Σ (r_ui - p_u · q_i)² + λ × (|p_u|² + |q_i|²). Sin esto, embeddings explotan a memorizar el train.
- Factors k: típicamente 20-200. Más → más capacidad y más overfitting. Tunear con validación.
- Funk SVD (Simon Funk, Netflix Prize 2006): el "SVD" popular del Netflix Prize NO es SVD sensu stricto — es factorización por SGD con regularización. El nombre quedó.

Dataset / recursos

- Explicit: MovieLens 1M con ratings 1-5.
- Implicit: Last.fm "lastfm-360K" o convertir MovieLens (rating ≥ 4 = "le gustó" = 1).
- Librerías: implicit>=0.7 (ALS rápido en Cython), scipy.sparse.linalg.svds, opcional surprise.

Ejercicios

1. SVD truncado: tomar matriz R densa imputando 0. U, sigma, Vt = svds(R, k=20). Reconstruir R̂ = U × diag(sigma) × Vt. Verificar que R̂ predice valores no-cero similares y "rellena" los ceros con guesses.
2. ALS explicit con Surprise: from surprise import SVD; algo = SVD(n_factors=50, n_epochs=20). algo.fit(trainset). Predict algo.predict(user, item). RMSE sobre test split.
3. ALS implicit con implicit: model = implicit.als.AlternatingLeastSquares(factors=64, regularization=0.05, iterations=15). model.fit(R_train * 40) (multiplicar por α=40). model.recommend(user_id, R_train[user_id], N=10).
4. Inspeccionar embeddings: PCA de model.item_factors a 2D y plot. Items "parecidos" deben caer cerca.
5. Biases analysis: imprimir top 10 movies por b_i (las que todos aman / odian) y top 10 usuarios por b_u.

Homework verificable

Notebook con:

1. Comparar 3 modelos sobre MovieLens 1M con leave-one-out por user:

- kNN item-based (Clase 216).
- Funk SVD (Surprise).
- ALS implicit (binarizando rating ≥ 4).

1. Reportar NDCG@10, recall@10, tiempo de entrenamiento.
2. Plot 2D (UMAP/t-SNE) de item_factors. Colorear por género — los géneros deberían formar clusters visibles.
3. Estudio de sensibilidad: k ∈ {10, 50, 100, 200} × λ ∈ {0.001, 0.01, 0.1} con cross-validation.
4. Recomendar para 3 users diferentes (un cinéfilo, un casual, un nuevo) — mostrar diferencia cualitativa.

Criterio de aceptación: ALS implicit gana en NDCG@10 sobre kNN; el grid de (k, λ) revela el sweet spot; los embeddings de items muestran estructura por género al PCAear.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
svds da componentes con signo arbitrario	SVD no fija el signo de los singular vecto
ALS converge a embeddings constantes	λ muy alto. Fix: bajar regularización (0.0

MemoryError con ALS sobre 10M users × 1M i	Implementación naive. Fix: implicit librar
Implicit-ALS overfit: recomienda solo lo q	Sin confidence weighting o con α muy alto.
Embeddings no muestran estructura semántic	k muy chico (k=5 no capta nada) o train se
Predicciones constantes para todos los use	Modelo bajó a solo predecir biases. Fix: c
Cold-start total: nuevo user → score=0	p_u no existe para user nuevo. Fix: fallba

Preguntas frecuentes

¿SVD del Netflix Prize es SVD matemático?

No. Lo que ganó Netflix era factorización por SGD con regularización ("Funk SVD"), no SVD-decomposition real. El nombre pegó por marketing.

¿implicit o lightfm o tensorflow recommenders?

- implicit: ALS implicit + BPR. Rápido (Cython), implicit-only. Default para CF moderno.
- lightfm: hybrid (CF + content features). Bueno si tenés metadata (Clase 219).
- tensorflow recommenders / recsim: deep learning, two-towers, sequential. Para escala grande + features ricas.

¿Qué k (factors) elijo?

Típicamente 50-200. Más k requiere más data + más regularización. Si tu dataset es chico (<100K interactions): k=20-50. Si tenés millones de interactions: k=100-200. Cross-validation es la única respuesta correcta.

¿ALS vs SGD para optimizar?

- ALS: cada subproblema es LSQ lineal → paralelizable, converge en pocas iteraciones. Bueno cuando matrix cabe en memoria y querés multi-core.
- SGD: por minibatch, on-line, escala a streaming. Más sensible a learning rate.

implicit usa ALS; PyTorch implementations típicamente usan SGD.

¿Embeddings se pueden usar para más cosas?

Sí. Aplicaciones: (1) similar items (cosine(q_i, q_j)), (2) clustering de items, (3) features para modelos downstream (CTR prediction), (4) two-tower retrieval (precomputar q_i, buscar nearest neighbor de p_u en producción con FAISS).

¿Y deep learning recommenders (NCF, deep CF)?

Two-towers + transformers (BERT4Rec, SASRec) son SOTA para sequential recommendation. Pero: ALS implicit sigue compitiendo en producción por simplicidad, latencia y costo. Empezá con ALS; pasá a DL si la métrica lo justifica.

Referencias

- Koren, Y., Bell, R., Volinsky, C. Matrix Factorization Techniques for Recommender Systems (IEEE Computer, 2009) — el clásico Netflix Prize.
- Hu, Y., Koren, Y., Volinsky, C. Collaborative Filtering for Implicit Feedback Datasets (ICDM 2008) — implicit ALS.
- implicit library — ALS + BPR en Cython.
- Funk SVD blog (Simon Funk, 2006) — el blog que cambió la historia.
- Recommender Systems: An Introduction (Jannach et al., 2010).

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 218 — Clase 218 — Content-based filtering

> *Parte: 6 — Sistemas de Recomendación · Fuente: Aggarwal Recommender Systems: The Textbook cap. 4 + docs scikit-learn TfidfVectorizer. Duración estimada: 70 min.*

Objetivo

Recomendar items basándose en sus atributos (texto, género, categoría) en vez de interacciones — útil cuando hay cold-start de items (Clase 221) o cuando los items tienen rica metadata. Combinar TF-IDF / embeddings sobre descripciones + scoring por similitud al perfil del usuario.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Construir un item profile desde texto + features categóricas con TF-IDF / one-hot.
- Construir un user profile como agregado ponderado de items que le gustaron.
- Calcular score(u, i) = cos(user_profile, item_profile) y rankear.
- Reemplazar TF-IDF por embeddings modernos (sentence-transformers) para entender semántica.
- Reconocer límites: serendipia baja (recomienda variantes de lo conocido); sobreespecialización.

Temas

#	Tema	Por qué importa
1	TF-IDF + cosine similarity	El baseline desde 1990.
2	One-hot vs multi-hot features categóricas	Géneros, tags, autores.
3	User profile como media ponderada	El "embedding del gusto".
4	Embeddings semánticos (sentence-transform	all-MiniLM-L6-v2 reemplaza TF-IDF con much
5	FAISS para top-N rápido	Cuando items son millones.
6	Hybrid con CF (Clase 219)	Content-only sufre serendipia.

Definiciones y características

- Content-based: usa atributos del item (texto, género, autor, año, precio) para recomendar. NO usa interacciones de otros usuarios (a diferencia de CF).
- TF-IDF (Term Frequency × Inverse Document Frequency): convierte texto en vector. Frecuencia de la palabra en el item × log(N / docs que contienen la palabra).
- Item profile: vector que representa al item. Para texto: TF-IDF. Para géneros: multi-hot. Para mix: concatenación.
- User profile: vector que representa el "gusto" del user. Típicamente: media (ponderada por rating) de los item profiles que consumió.
- Cosine similarity: ángulo entre vectores, robusto a magnitud. Default para content-based.
- Embedding semántico: vector denso (e.g. 384-dim) de un modelo pre-entrenado (BERT, MiniLM) que captura significado, no solo palabras exactas. "Auto" y "carro" → similar.

- Serendipia: probabilidad de descubrir algo inesperado. Content-based la mata (recomienda variantes de lo conocido). CF sufre menos.

Dataset / recursos

- Dataset: MovieLens + sinopsis (de TMDB API) o kaggle/the-movies-dataset.
- Librerías: scikit-learn (TfidfVectorizer), sentence-transformers, opcional faiss-cpu.

Ejercicios

1. TF-IDF base: cargar movies con title + overview + genres. TfidfVectorizer(max_features=10000, ngram_range=(1,2)). Matrix shape (n_items, 10000).
2. Item-item similarity: cosine_similarity(tfidf_matrix). Para Toy Story, mostrar top-10 más similares — deberían ser otras animaciones.
3. User profile: para user_id=42, tomar items rateados ≥4. user_profile = mean(item_profiles) (ponderado por rating). Recomendar top-10.
4. Embeddings modernos: model = SentenceTransformer('all-MiniLM-L6-v2'). Generar embeddings de cada movie. Comparar top-10 de TF-IDF vs embeddings — los embeddings entienden semántica.
5. FAISS rápido: index = faiss.IndexFlatIP(384); index.add(embeddings). index.search(user_profile, k=10) → top-10 en <1 ms aunque haya 1M items.

Homework verificable

Notebook con:

1. Recomendador content-based con sentence-transformers sobre MovieLens + sinopsis (10K movies).
2. FAISS index para retrieval <10 ms p99.
3. Comparativa con CF (Clase 216-217): NDCG@10, coverage (% items recomendados al menos una vez), diversidad (1 - avg intra-list similarity).
4. Demonstración cold-start: agregar una "movie nueva" sin ratings; verificar que content-based la recomienda; CF no.
5. README discutiendo: cuándo content-based gana (cold-start, niche items), cuándo pierde (filter bubble, serendipia baja).

Criterio de aceptación: content-based recomienda la movie nueva sin ratings (CF la ignora); coverage de content-based es ≥CF; el estudiante justifica un hybrid (Clase 219).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
TF-IDF gigante: shape (10K items, 100K fea	Sin max_features o stopwords. Fix: max_fea
Recomendaciones todas iguales: variantes d	Sobreespecialización (filter bubble). Fix:
User profile vacío para new user	No tiene interactions todavía. Fix: onboar
Embeddings de 100K items en RAM = 150 MB	Es OK pero el cosine_similarity 100K × 100
Multi-hot de géneros domina el TF-IDF	Concatenaste sin normalizar. Fix: normaliz
sentence-transformers tarda mucho al carga	Es lento la primera vez (download del chec

Preguntas frecuentes

¿Content-based vs CF?

Resuelven cosas distintas, y la mejor solución suele ser hybrid (Clase 219).

- Content-based: maneja item cold-start, explica por qué ("te recomendamos X porque te gustó Y, ambos de género acción"), pero filter bubble.

- CF: capta señales sociales (a gente como vos le gustó), serendipia, pero no funciona con items nuevos.

¿TF-IDF o embeddings?

Embeddings (sentence-transformers) ganan en calidad semántica. TF-IDF gana en velocidad de inferencia y simplicidad. Para 2026 greenfield: empezá con embeddings, agregá FAISS para retrieval.

¿Cómo combino texto + features categóricas?

Tres opciones: (1) concatenar TF-IDF + one-hot (escalado), (2) entrenar un modelo de feature embedding (fastText, USE) sobre todo, (3) usar metadata como features adicionales en LightFM (Clase 222).

¿sentence-transformers modelo elijo?

- all-MiniLM-L6-v2: 384-dim, rápido, buena calidad. Default razonable.
- all-mpnet-base-v2: 768-dim, mejor calidad, ~3× más lento.
- multilingual: paraphrase-multilingual-MiniLM-L12-v2.

¿FAISS es necesario?

Si tenés <10K items: cosine_similarity con sklearn alcanza. Si tenés >100K items y querés top-N en <50 ms: FAISS obligatorio. Para 1M+ items y escala: FAISS con índices avanzados (HNSW, IVF) o Milvus/Pinecone (managed vector DBs).

¿Y los embeddings de OpenAI?

text-embedding-3-small/large son fuertes; pagás por inference + lock-in. Para empezar/offline: sentence-transformers gratis. Para producción con presupuesto: OpenAI / Cohere / Voyage.

Referencias

- Aggarwal, C. Recommender Systems: The Textbook (Springer, 2016), cap. 4 — Content-Based Recommender Systems.
- sklearn TfidfVectorizer.
- sentence-transformers — modelo pre-entrenados.
- FAISS — similarity search.
- Carbonell & Goldstein, The Use of MMR, Diversity-Based Reranking for Reordering Documents (1998) — para combatir filter bubble.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 219 — Clase 219 — Recomendadores híbridos

> *Parte: 6 — Sistemas de Recomendación · Fuente: Burke, Hybrid Recommender Systems: Survey and Experiments (UMUAI 2002) + Kula, Metadata Embeddings for User and Item Cold-start Recommendations (LightFM, 2015). Duración estimada: 75 min.*

Objetivo

Combinar CF (filtrado colaborativo, Clase 216-217) + content-based (Clase 218) para conseguir lo mejor de ambos: serendipia + cold-start + explicabilidad. Aplicar los 7 patrones de hybrid de Burke (2002) y usar

LightFM (que aprende un modelo único con CF + features).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Diferenciar los 7 patrones de hybrid: weighted, switching, mixed, feature combination, cascade, feature augmentation, meta-level.
- Implementar un hybrid weighted: score = α × score_cf + (1-α) × score_content.
- Implementar un hybrid switching: usar content para users con <N interactions, CF para el resto.
- Usar LightFM como hybrid built-in: el modelo aprende embeddings que combinan CF + content features.
- Tunear α con validation y entender por qué α óptimo varía por user/item segment.

Temas

#	Tema	Por qué importa
1	7 patrones de Burke (2002)	Vocabulario para discutir arquitecturas.
2	Weighted hybrid: α × CF + (1-α) × CB	El más simple y muy efectivo.
3	Switching: cold-start triage	"Si user tiene <5 ratings, usá content".
4	LightFM: hybrid aprendido	Embeddings que combinan ambas señales.
5	Tuning α por segmento	Cold-start: peso a content; long-tail: pes
6	Two-tower modelo (concept)	El sucesor moderno de LightFM.

Definiciones y características

- Weighted: combinar scores por suma ponderada. score = α × CF + (1-α) × CB. α se tunea con validación.
- Switching: elegir uno u otro según contexto. Ej: cold-start (user nuevo) → content; usuario maduro → CF.
- Mixed: presentar resultados de ambos en la misma página (ej: carousel "porque te gustó X" + carousel "popular ahora").
- Feature combination: agregar señales CF (ratings) como features a un modelo content-based, o viceversa.
- Cascade: primero filtra con un modelo (e.g. CF top-100), después re-rankea con otro (e.g. content-based fine-grained).
- Feature augmentation: usar el output de un modelo (e.g. cluster CF del user) como feature de otro.
- Meta-level: el modelo aprende cuándo usar cada uno (gating network, mixture of experts).
- LightFM: librería que aprende un único embedding por user/item combinando CF + content features. Funciona en pure CF, pure content, o hybrid.
- Two-tower architecture: red neuronal con dos encoders (user tower + item tower), ambos producen embeddings que se combinan via dot product. SOTA moderno para retrieval.

Dataset / recursos

- Dataset: MovieLens 100K con u.item que tiene géneros (19 binarios).
- Librerías: lightfm>=1.17, scipy.sparse, pandas.

Ejercicios

1. Weighted hybrid manual: tomar scores de Clase 217 (ALS) y Clase 218 (content-based). Combinar score = α × cf + (1-α) × cb para α {0, 0.25, 0.5, 0.75, 1}. Reportar NDCG@10 para cada α.
2. Switching por user: si interactions(u) < 5: usar content; si no: usar CF. Comparar contra weighted para

users en distintos segmentos (new/mature).

3. LightFM hybrid: entrenar LightFM(loss='warp') con item_features (géneros) y user_features (demographics). Comparar NDCG vs pure CF (sin features).
4. Cold-start eval: held-out incluye items nuevos (no en train) y users nuevos (sin ratings). Comparar pure CF (~0%), pure content (~OK), LightFM hybrid (~mejor).
5. Cascade: top-100 con CF, re-rankear top-10 con content (boost a items con descripción similar al historial del user).

Homework verificable

Notebook con:

1. 4 modelos sobre MovieLens 100K:
- Pure CF (implicit ALS).
- Pure content-based (sentence-transformers).
- Weighted hybrid (α tunado).
- LightFM hybrid (con item features).
1. Tabla comparativa: NDCG@10, recall@10, coverage, diversity, tiempo train, tiempo predict.
2. Análisis por segmento: cold-start users (≤3 ratings), warm users (≥20). ¿Cuál gana en cada?
3. Curva α vs NDCG@10 para weighted hybrid.
4. Documentación: arquitectura recomendada para 3 casos: e-commerce, streaming, news.

Criterio de aceptación: weighted hybrid o LightFM gana sobre pure CF en al menos uno de los segmentos; el α óptimo está justificado con números; las recomendaciones por arquitectura son sensatas.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Weighted hybrid no mejora sobre CF	α no tuneado, o scores en escalas distinta
LightFM peor que pure CF	Item features con poca info (e.g. 19 géner
Switching tiene "jump" entre user con 4 vs	Discontinuidad. Fix: weighted con α(n) = s
Cold-start eval reporta ~0 para CF	Esperado. Fix: ese es el punto del ejercic
LightFM WARP loss muy lento	WARP es O(n_items × negatives) por ejemplo
Mismas recomendaciones para todos los user	Probablemente α=0 o α=1 colapsa todo. Fix:

Preguntas frecuentes

¿Qué hybrid pattern elegir?

Empezá con weighted o switching — son los más simples e interpretables. LightFM si tenés features ricos. Two-tower deep solo si tenés equipo ML + escala que lo justifique.

¿LightFM sigue siendo relevante en 2026?

Sí para datasets pequeños/medianos con features. Para grandes escalas: two-tower (TF Recommenders, recsys-pytorch). Pero LightFM es el sweet spot "complejidad-calidad" para casos chicos-medianos.

¿Cómo tuneo α?

(1) Grid search con validation NDCG@10. (2) Bayesian opt si tenés más hiperparámetros. (3) Por segmento (cold-start users → α bajo, mature → α alto). (4) Online via contextual bandit (avanzado).

¿Mixed (carousels) cuenta como hybrid?

Sí — es el patrón más usado en producción real. Spotify/Netflix muestran múltiples carousels con distintas estrategias ("porque viste X", "popular en tu país", "nuevos lanzamientos"). Cada carousel es un recomendador distinto; el "hybrid" es la página final.

¿Cuándo NO usar hybrid?

Si: (1) tu dataset es 100% interactions sin features útiles → pure CF. (2) Tus items no tienen historial pero tenés metadata rica → pure content. (3) Sos un startup con 1 ML eng → empezá simple, sumá hybrid cuando duela.

¿Two-tower vs LightFM?

Two-tower (TF Recommenders, PyTorch) escala mejor + permite features más complejos (text embeddings, image embeddings). LightFM es matriz factorization + lineal sobre features. Para 2026 SOTA: two-tower. Para empezar: LightFM.

Referencias

- Burke, R. Hybrid Recommender Systems: Survey and Experiments (UMUAI 2002) — los 7 patrones canónicos.
- Kula, M. Metadata Embeddings for User and Item Cold-start Recommendations (DLRS 2015) — LightFM paper.
- LightFM docs.
- TensorFlow Recommenders — two-tower moderno.
- Aggarwal cap. 6 — Ensemble-Based and Hybrid Recommender Systems.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 220 — Clase 220 — Métricas: MAP@k, NDCG, recall@k

> *Parte: 6 — Sistemas de Recomendación · Fuente: Aggarwal cap. 7 + Järvelin & Kekäläinen, Cumulated Gain-Based Evaluation of IR Techniques (TOIS 2002 — NDCG paper). Duración estimada: 70 min.*

Objetivo

Evaluar recomendadores con las métricas correctas: NO accuracy ni RMSE de rating (irrelevante para top-N). Sí recall@k (¿cuántos relevantes recuperaste en top-k?), precision@k (¿qué % del top-k es relevante?), MAP@k (precision promedio sensible al ranking), NDCG@k (gain descontado por posición). Decidir qué métrica reportar según objetivo de negocio.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Calcular precision@k, recall@k, F1@k, hit rate desde cero.
- Calcular MAP@k (Mean Average Precision) y entender por qué sensible al ranking dentro del top-k.
- Calcular NDCG@k (Normalized Discounted Cumulative Gain) y entender el descuento logarítmico.
- Diseñar el split correcto: leave-one-out por user vs temporal split vs random.
- Decidir métrica según objetivo: recall (catálogo grande, descubrimiento) vs NDCG (orden importa) vs

MAP (búsqueda).

Temas

#	Tema	Por qué importa
1	Por qué NO usar RMSE de rating	Top-N no usa el rating predicho — usa el r
2	Precision@k vs recall@k	Trade-off; recall importa más con catálogo
3	MAP@k: precision promediada por posición	Sensible a poner relevantes ARRIBA del top
4	NDCG@k con descuento log2(i+1)	El relevante en posición 1 vale más que en
5	Leave-one-out vs temporal split	Temporal evita data leakage en RS de tiemp
6	Coverage + diversity (beyond accuracy)	Métricas de "salud" del catálogo.

Definiciones y características

- Recall@k: |relevantes ∩ top-k| / |relevantes totales|. Pregunta: "¿qué % de lo que el user quiere mostré?". Ideal con catálogo grande.
- Precision@k: |relevantes ∩ top-k| / k. Pregunta: "¿qué % del top-k es relevante?". Ideal cuando el usuario consume pocos (top-5).
- F1@k: armónico de precision y recall.
- Hit rate@k: 1 si al menos 1 relevante en top-k, 0 si no. Más simple, popular en deep recsys research.
- AP@k (Average Precision): Σ_i (precision@i × rel(i)) / |relevantes| para i [1, k]. Penaliza relevantes en posiciones bajas.
- MAP@k (Mean AP): promedio de AP@k sobre todos los users.
- DCG@k (Discounted Cumulative Gain): Σ_i rel(i) / log2(i+1). Posiciones bajas valen menos.
- NDCG@k: DCG@k / iDCG@k donde iDCG es el máximo posible. Normaliza a [0, 1].
- Coverage (catalog coverage): |items recomendados al menos 1 vez en N users| / |items totales|.
- Diversity (intra-list): 1 - avg(sim(i, j)) entre pares en la lista. Alto = recomendaciones diversas.

Dataset / recursos

- Dataset: MovieLens 100K.
- Librerías: numpy puro (las métricas son fáciles), opcional recmetrics.

Ejercicios

1. Implementar precision@k, recall@k: dadas listas [1, 0, 0, 1, 0] (relevancia) y k=5, calcular. Verificar contra recmetrics u otra librería.
2. MAP@k: implementar AP@k. Mostrar diferencia con precision@k: AP penaliza tener los relevantes al final.
3. NDCG@k: implementar DCG y iDCG. Mostrar caso donde recall@k es igual pero NDCG distinto porque el orden cambia.
4. Leave-one-out por user: para cada user con ≥5 ratings, dejar el último (cronológico o random) para test, resto para train. Evaluar.
5. Coverage y diversity: para 1000 users, ¿qué % del catálogo fue recomendado? Diversidad media intra-list (cosine entre items recomendados).

Homework verificable

Notebook con:

1. Implementación de 6 métricas: recall@10, precision@10, F1@10, MAP@10, NDCG@10, hit rate@10. Validar contra librería.
2. Aplicar a 4 modelos (kNN, ALS, content, hybrid de Clases 216-219). Tabla comparativa.

3. Análisis: ¿qué métrica decidiría cuál modelo gana? Discutir.
4. Beyond accuracy: coverage, diversity, novelty (1 - popularity rank promedio). ¿Algún modelo gana en accuracy pero pierde en diversity?
5. Temporal split: si tu dataset tiene timestamps, hacer split por fecha (no por ratio). Comparar con random split. Discutir por qué temporal es más realista.

Criterio de aceptación: las 6 métricas coinciden con recmetrics ±1e-6; la elección de "mejor modelo" depende de la métrica; el estudiante justifica una arquitectura con métricas de negocio.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
MAP@k diferente a la librería	Diferencias de convención: ¿dividís por mi
NDCG@k > 1	Bug: estás computando DCG sin normalizar p
Recall=0.0 cuando esperaba algo	Probablemente recomendaste items que ya es
RMSE alto pero recall@10 alto	Esperado en implicit. RMSE no importa para
Random split en dataset temporal → métrica	Data leakage: train tiene futuro vs test.
Coverage 100% → "óptimo"	Coverage 100% probablemente significa reco
Diversity alta pero NDCG bajo	El modelo está recomendando cosas random.

Preguntas frecuentes

¿Qué métrica reportar?

Casi siempre: NDCG@k + recall@k + coverage. NDCG mide calidad del ranking, recall mide cobertura del relevante, coverage mide salud del catálogo. Si tu UI muestra top-5: k=5. Top-20: k=20.

¿MAP o NDCG?

NDCG es más expresivo (acepta relevancia graduada: rating 5 > rating 3). MAP solo binario. En implicit feedback (binary): equivalentes en práctica. Default: NDCG.

¿RMSE alguna vez sirve?

Sí, en sistemas de rating prediction donde el rating predicho se muestra al usuario (ej. Netflix mostraba "92% match"). Si solo mostrás top-N: NO usar RMSE.

¿Leave-one-out o random split?

LOO por user es estándar en RS papers — emula la situación "predecir el próximo item". Random split puede sesgar (data leakage si hay temporalidad). Para producción real: temporal split (train con datos hasta T-7d, test con T-7d a T-0d).

¿Cómo manejo cold-start en evaluación?

(1) Reportar métricas por segmento (cold vs warm users; new vs popular items). (2) En "all", incluir cold-start (es honesto, aunque baje promedio). (3) NO excluir cold-start users — el sistema real los tiene.

¿Online metrics vs offline?

Las offline (NDCG, recall) son proxies. Online (CTR, conversion, watch time, retention) son la verdad. La correlación offline-online suele ser positiva pero no perfecta. En producción: A/B test (Clase 204).

Referencias

- Aggarwal Recommender Systems: The Textbook (Springer, 2016), cap. 7 — Evaluating Recommender

Systems.
- Järvelin, K. & Kekäläinen, J. Cumulated Gain-Based Evaluation of IR Techniques (ACM TOIS 2002) — paper original de NDCG.
- recmetrics — libreria con todas implementadas.
- Beyond accuracy: evaluating recommender systems by coverage and serendipity (Ge et al., RecSys 2010).

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 221 — Clase 221 — Cold-start problem

> *Parte: 6 — Sistemas de Recomendación · Fuente: Schein, Popescul, Ungar, Pennock, Methods and Metrics for Cold-Start Recommendations (SIGIR 2002) + Aggarwal cap. 13. Duración estimada: 70 min.*

Objetivo

Cuándo un user o item es "nuevo" (0 interacciones), CF (Clase 216-217) no funciona. Estrategias concretas para los 3 tipos de cold-start: user cold-start (onboarding), item cold-start (catalog launch), system cold-start (lanzamiento del producto). Las estrategias correctas son la diferencia entre un recomendador útil desde el día 1 vs uno inútil hasta el mes 6.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Diferenciar user cold-start (usuario nuevo), item cold-start (item nuevo), system cold-start (todo nuevo).
- Aplicar popularity fallback con shrinkage Bayesiano ((c + m × C) / (n + m)) para users sin historia.
- Diseñar onboarding explícito (pedir N preferencias antes de personalizar).
- Usar content features (Clase 218) para items nuevos.
- Aplicar exploration-exploitation con bandits (epsilon-greedy, Thompson sampling) para users mid-cold-start.

Temas

#	Tema	Por qué importa
1	3 tipos de cold-start	Cada uno necesita estrategia distinta.
2	Popularity fallback	Default sano cuando no sabés nada.
3	Bayesian shrinkage para popularity	Evita que items con 2 ratings 5/5 dominen.
4	Onboarding: preguntar al user	"Elegí 3 géneros" cambia el juego.
5	Content-based para item cold-start	Embeddings de texto/imágenes.
6	Bandits para explorar	Cuando offline metrics no alcanzan.

Definiciones y características

- User cold-start: usuario sin historial (recién registrado). El modelo no tiene p_u. Estrategias: popularity, onboarding, demographics.
- Item cold-start: item nuevo sin ratings. El modelo no tiene q_i. Estrategias: content features

(descripción, categoría), boost temporal "nuevos".
- System cold-start: lanzamiento del producto, ni users ni items con historia. Estrategias: editorial picks, externos (importar gustos desde Facebook/Google), popularity por geo/demographic.
- Popularity shrinkage Bayesiano: score = (Σ ratings + m × C) / (n + m) donde C es la media global, m es el "weight" del prior. Items con n=2 ratings 5/5 no superan items con n=1000 ratings 4.2/5.
- Onboarding explícito: pedir al user N preferencias (géneros, intereses) antes de personalizar. Estilo Spotify/Netflix.
- Bandits: framework de RL para exploration vs exploitation. Epsilon-greedy: con prob ε explora random, con prob 1-ε recomienda el "best". Thompson sampling: muestra de la posterior de cada arm.
- Contextual bandits: el contexto (user features) influye en la decisión. Más sofisticado que vanilla bandits.

Dataset / recursos

- Dataset: MovieLens 100K + new users / new items simulados.
- Librerías: numpy, opcional vowpalwabbit (bandits), scikit-learn.

Ejercicios

1. Popularity baseline: rankear items por n_ratings. Top-10 son siempre los mismos. Evaluar recall@10 para users cold-start (con 0 interactions).
2. Bayesian shrinkage: implementar (sum + m × C) / (n + m) con m=10, C=mean_rating. Comparar top-10 con popularity vanilla. Items con pocos ratings caen al promedio.
3. Onboarding 3 géneros: simular que user nuevo elige ["Action", "Sci-Fi", "Comedy"]. Recomendar top-10 movies de esos géneros (content-based + popularity tiebreaker).
4. Item cold-start: agregar 10 movies nuevas con descripción pero 0 ratings. Content-based (Clase 218) las puede ranquear; CF no. Demostrar.
5. Epsilon-greedy: en cada slot del top-10, con prob ε=0.1 recomendar item random (explore), con prob 0.9 recomendar el "best" del modelo (exploit). Medir coverage en N usuarios.

Homework verificable

Repo con:

1. Recomendador full con manejo explícito de cold-start:
- User nuevo: onboarding (elegir géneros) → content-based.
- User con ≤5 interactions: weighted hybrid con α bajo (más content).
- User maduro: CF puro.
1. Item nuevo: content-based hasta acumular 10 ratings, después ALS.
2. Bayesian shrinkage en todos los rankings "popularity-based".
3. Bandit epsilon-greedy en producción (simulado): tasa ε=0.05, decay a 0.01 después de 30 días.
4. Evaluación por segmento (cold-start, warm) con NDCG@10. Mostrar que estrategia adaptativa supera a CF puro para cold.

Criterio de aceptación: cold-start users tienen NDCG@10 > 0.05 (vs ~0 con CF puro); items nuevos aparecen en recomendaciones dentro de las primeras 48h.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Popularity baseline domina todo	Tu CF no está aprovechando — probable bug
Items nuevos nunca son recomendados ("rece	Pure content sin boost por novelty. Fix: a
Onboarding pide 20 géneros — los users se	UX trade-off. Fix: pedir 3-5 max; mejor 1

ε-greedy estanca: nunca aprende	ε muy bajo desde día 1. Fix: arrancar con
Bayesian shrinkage me da scores casi igual	m muy alto. Fix: tunear m por validación —
Geographic popularity sesga a country mayo	Sin segmentación. Fix: popularity por (cou

Preguntas frecuentes

¿Onboarding pregunta directa o implícita?

- Directa (elegir 3 géneros): rápida, controlable, pero rompe el flujo. Spotify la usa al sign-up.
- Implícita (observar primeras N interacciones, no recomendar hasta tener señal): seamless, pero los primeros 5 clicks son perdidos.

Híbrido: pregunta directa básica (1-2 items) + observar primeras N.

¿Mostrar items "Nuevo" boost por cuánto tiempo?

Depende del catálogo. News: minutos-horas. Movies: días-semanas. Productos e-commerce: días. Decay logarítmico: boost = 1 / (1 + days).

¿Bandits valen la pena vs A/B test?

A/B test mide impacto de una variante vs otra. Bandits explotan automáticamente la mejor opción mientras siguen explorando. Para múltiples variantes (10+ tipos de recomendación) o cuando el costo de no explorar es alto: bandits. Para 2-3 variantes y experimentación controlada: A/B (Clase 204).

¿Cold-start de sistema cómo arranco?

(1) Importar gustos desde otra plataforma (Facebook Connect, Google sign-in). (2) Editorial picks curados. (3) Promociones para incentivar primer rating. (4) Partner con plataforma que ya tiene data.

¿Cuándo termina el cold-start?

Heurística: cuando NDCG@10 personal supera consistente a NDCG@10 de popularity baseline. Típicamente: 5-20 interactions. Depende del catálogo.

¿Hay solo CF + content para resolverlo?

No. Cross-domain transfer (gustos en Spotify → recomendaciones en Audible), demographic CF (gente como vos suele consumir X), conversational ("¿qué te interesa hoy?") son alternativas avanzadas.

Referencias

- Schein, A. et al. Methods and Metrics for Cold-Start Recommendations (SIGIR 2002).
- Aggarwal Recommender Systems: The Textbook (Springer, 2016), cap. 13 — Advanced Topics (incluye cold-start).
- Sutton & Barto, Reinforcement Learning: An Introduction (2nd ed., 2018), cap. 2 — Multi-armed bandits.
- Vowpal Wabbit contextual bandits.
- Two Decades of Recommender Systems at Amazon (Smith & Linden, IEEE, 2017) — perspectiva histórica.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 222 — Clase 222 — Librerías: LightFM, Implicit, Surprise

> *Parte: 6 — Sistemas de Recomendación · Fuente: docs oficiales Surprise + LightFM + Implicit + TensorFlow Recommenders. Duración estimada: 70 min.*

Objetivo

Conocer las librerías de recomendación que vas a usar en la vida real sin tener que reimplementar lo de Clases 216-221. Decidir cuál usar según: tipo de feedback (explicit/implicit), tamaño del dataset, features disponibles, integración con stack.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Usar Surprise para algoritmos clásicos explicit (KNNBasic, SVD, SVD++, NMF, Co-Clustering) con API tipo sklearn.
- Usar Implicit para ALS implicit, BPR, Logistic MF — la opción más rápida en Python para datasets grandes.
- Usar LightFM cuando tenés features (Clase 219 hybrid).
- Conocer TensorFlow Recommenders y Spotlight (PyTorch) para arquitecturas deep (two-tower, sequential).
- Decidir librería según: explicit vs implicit, escala, features, deployment target.

Temas

#	Tema	Por qué importa
1	Surprise: explicit, didáctica	Empezar a aprender RS.
2	Implicit: ALS/BPR Cython	El caballo de batalla en producción.
3	LightFM: CF + features	Hybrid built-in.
4	TF Recommenders + Spotlight	Deep RS (two-tower, BERT4Rec).
5	Spark pyspark.ml.recommendation.ALS	Cuando el dataset es TB.
6	Vector DBs (FAISS, Milvus, Pinecone)	Serving de embeddings a escala.

Definiciones y características

- Surprise: librería didáctica, API tipo sklearn. Maneja explicit feedback con KNN, SVD, NMF, etc. Bueno para aprender, lento para grandes datasets.
- Implicit (Ben Frederickson): Cython + multi-thread. ALS implicit (Hu et al.), BPR (Bayesian Personalized Ranking), Logistic MF. Default para CF en producción Python.
- LightFM (Lyst): factorización con features. Loss WARP / BPR / Logistic. El sweet spot para hybrid con metadata.
- TensorFlow Recommenders (TFRS): API alto-nivel sobre TF/Keras para two-tower, ranking, retrieval. SOTA para deep RS, requiere setup TF.
- Spotlight (Maciej Kula): PyTorch library para sequential + factorización. Para experimentos research.
- Spark ML ALS: distribuido, escala a billones de interactions. Cuando el dataset no entra en single machine.
- Vector DB: para servir embeddings. FAISS (in-process), Milvus (open-source server), Pinecone/Qdrant/Weaviate (managed). Permiten recommend(user_emb, top_k=10) en <10 ms con M items.

Dataset / recursos

- Dataset: MovieLens 100K y 1M.
- Librerías: scikit-surprise, implicit>=0.7, lightfm>=1.17, opcional tensorflow-recommenders.

Ejercicios

1. Surprise SVD: from surprise import SVD, Dataset; data = Dataset.load_builtin('ml-100k'); algo = SVD(); cross_validate(algo, data, measures=['RMSE','MAE'], cv=5). Reportar RMSE.
2. Implicit ALS: model = implicit.als.AlternatingLeastSquares(factors=64, regularization=0.05, iterations=20). model.fit(R_sparse * 40). recommend(user_id, R[user_id], N=10).
3. Implicit BPR: mismo modelo pero implicit.bpr.BayesianPersonalizedRanking(factors=64). Comparar NDCG vs ALS.
4. LightFM: LightFM(loss='warp', no_components=32). Con item features (géneros). Comparar pure CF vs hybrid.
5. Benchmarking: mismo dataset, mismo split, las 4 librerías. Tabla con NDCG@10, recall@10, tiempo entrenamiento, tiempo predict.

Homework verificable

Notebook con:

1. Comparativa rigurosa sobre MovieLens 1M:

- Surprise SVD
- Implicit ALS
- Implicit BPR
- LightFM WARP (con features)

1. Mismo train/test split temporal. Mismas métricas (Clase 220).
2. Tabla: NDCG@10, recall@10, coverage, tiempo train (s), tiempo predict (ms/user), RAM peak.
3. Recomendaciones por caso de uso (decisión justificada):

- "Quiero algo simple para aprender" → Surprise.
- "Quiero performance en producción Python para 10M users" → Implicit.
- "Tengo features ricos y dataset chico-medio" → LightFM.
- "Tengo equipo ML serio y quiero SOTA" → TF Recommenders / two-tower.

1. Bonus: deploy del mejor modelo como servicio FastAPI (Clase 199) con FAISS para retrieval rápido.

Criterio de aceptación: 4 modelos entrenan limpio, NDCG@10 reportado correctamente, la tabla permite decidir cuál usar por caso.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Surprise lentísimo sobre 1M+	Surprise está pensado para didáctico. Fix:
implicit warning OpenBLAS threads	Conflict entre BLAS threading y OpenMP thr
LightFM WARP muy lento	WARP es O(samples) por epoch. Fix: loss='b
TFRS requiere graph mode + estructura comp	Curva de aprendizaje fuerte. Fix: empezar
FAISS index demasiado grande para RAM	IndexFlatIP guarda todo en RAM. Fix: Index
Score implicit ALS no es probabilidad	Es un score, no proba. Fix: para servir co

Preguntas frecuentes

¿Cuál instalo primero?

Implicit para CF moderno (90% de los casos). LightFM si tenés features. Surprise solo para aprender el ABC.

¿Implicit y LightFM compiten o se complementan?

Compiten. Implicit es CF puro (más rápido, más simple). LightFM es CF + features (más flexible). Si no tenés features útiles: Implicit gana. Si tenés metadata rica: LightFM.

¿TF Recommenders vale la pena?

Sí cuando: (1) Dataset >100M interactions. (2) Necesitás features complejos (text embeddings, image embeddings). (3) Tu stack ya es TF. (4) Querés sequential / session-based RS (BERT4Rec, SASRec). Para casos chicos: overkill.

¿Cómo sirvo embeddings en producción?

(1) Pre-computar item_factors offline. (2) Indexar con FAISS (in-process) o Milvus/Pinecone (managed). (3) En request: computar user_emb (rápido o desde cache) → index.search(user_emb, k=10) → top-N items. Latencia: <10 ms para millones de items con HNSW.

¿Reentrenamiento — cada cuánto?

Para CF: diario o semanal típicamente. Para deep RS con features estáticos: semanal-mensual. Para sequential: continuous training. Ver Clase 203.

¿RecBole, Cornac, Spotlight?

Frameworks de research, mucho catálogo de algoritmos. Útiles para comparar 20 modelos en paper. Para producción: empezás con uno (Implicit/LightFM) y lo customizás.

Referencias

- Surprise docs.
- Implicit docs + GitHub.
- LightFM docs.
- TensorFlow Recommenders — two-tower + ranking.
- FAISS docs — vector similarity search.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Cierre de la parte

Fin del bundle consolidado de Parte 6 — Sistemas de Recomendación · 7 clases.