Clase 222 — Librerías: LightFM, Implicit, Surprise

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

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.

Siguiente clase

Clase 223 — Sesgos algorítmicos: tipos y origen

Fin de la Parte 6 — Sistemas de Recomendación. Tenés CF (216-217) + content (218) + híbridos (219)

+ métricas (220) + cold-start (221) + ecosistema de librerías (222). Parte 7 cierra el ciclo con ética, fairness, privacidad — qué hacer cuando tus recomendaciones afectan vidas reales.

Apéndice: notebook (primer bloque)

Mismo dataset sintético, las 3 librerías. NDCG@10 + tiempo de entrenamiento. Requiere: pip install scikit-surprise implicit lightfm scipy.

```
import os; os.environ['OPENBLAS_NUM_THREADS'] = '1'   # evitar warning de implicit
import numpy as np, pandas as pd, time
from scipy.sparse import csr_matrix

rng = np.random.default_rng(42)
n_users, n_items = 500, 300

# Generar ratings sintéticos
P_true = rng.normal(0, 1, (n_users, 8))
Q_true = rng.normal(0, 1, (n_items, 8))
scores_true = P_true @ Q_true.T
noise = rng.normal(0, 0.3, scores_true.shape)
interact_prob = 1 / (1 + np.exp(-(scores_true + noise)))
R = (rng.random(scores_true.shape) < 0.15 * interact_prob).astype(float)
ratings = np.where(R > 0, rng.integers(3, 6, R.shape), 0).astype(int)

df = pd.DataFrame([
    {'user_id': u, 'item_id': i, 'rating': int(ratings[u, i])}
    for u in range(n_users) for i in range(n_items) if ratings[u, i] > 0
])
print(f'dataset: {len(df):,} ratings')
```

Archivos complementarios

- notebook.ipynb