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.

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.

Siguiente clase

Clase 219 — Recomendadores híbridos

Apéndice: notebook (primer bloque)

Movies sintéticas con título + overview + genres. Recomendamos por similitud de contenido. Requiere: pip install scikit-learn sentence-transformers faiss-cpu.

```
import numpy as np, pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

movies = pd.DataFrame([
    {'id': 1,  'title': 'Toy Story',           'overview': 'Cowboy doll and astronaut toy adventures friendship.', 'genres': 'animation children comedy'},
    {'id': 2,  'title': 'Jumanji',             'overview': 'Magical board game jungle wild animals.',              'genres': 'adventure children fantasy'},
    {'id': 3,  'title': 'Heat',                'overview': 'Bank heist crew detective pursuit Los Angeles.',       'genres': 'action crime thriller'},
    {'id': 4,  'title': 'Goldeneye',           'overview': 'British spy secret agent satellite weapon villain.',   'genres': 'action adventure thriller'},
    {'id': 5,  'title': 'The Lion King',       'overview': 'Lion cub father betrayed uncle Africa savanna.',       'genres': 'animation adventure drama'},
    {'id': 6,  'title': 'Pulp Fiction',        'overview': 'Hitmen boxer gangster intersecting stories LA.',       'genres': 'crime drama thriller'},
    {'id': 7,  'title': 'Finding Nemo',        'overview': 'Clownfish ocean adventure son rescue father.',         'genres': 'animation children adventure'},
    {'id': 8,  'title': 'The Matrix',          'overview': 'Hacker discovers reality simulation rebel against AI.',
                                                                                                                    'genres': 'action sci-fi thriller'},
    {'id': 9,  'title': 'Forrest Gump',        'overview': 'Slow-witted man witnesses 20th century history love.', 'genres': 'comedy drama romance'},
    {'id': 10, 'title': 'Inception',           'overview': 'Dream thief mind heist subconscious layers.',          'genres': 'action sci-fi thriller'},
    {'id': 11, 'title': 'Shrek',               'overview': 'Ogre swamp princess rescue donkey fairy tale.',        'genres': 'animation comedy adventure'},
    {'id': 12, 'title': 'The Dark Knight',     'overview': 'Batman Joker chaos Gotham villain moral.',             'genres': 'action crime thriller'},
])

movies['text'] = movies.title + ' ' + movies.overview + ' ' + movies.genres
print(movies[['id', 'title']].to_string(index=False))
```

Archivos complementarios

- notebook.ipynb