lmuqfcekg

October 22, 2025

0.1 Sistema de recomendación de películas (content-based)

Objetivo. Construir un recomendador ligero y reproducible que funcione sin backend dedicado, exportando artefactos reutilizables en hosting compartido.

Dataset. TMDB-5000 (películas y créditos). Se usan `tmdb_5000_movies.csv` y `tmdb_5000_credits.csv`.

Features (tags) por película. - `overview` tokenizado - `genres.name` - `keywords.name` - `cast.name` (top 3) - `crew.name` filtrado a `director`

Preprocesamiento. - Unir por `title` - Relleno de nulos - Parseo de listas JSON con `ast.literal_eval` - Normalización a minúsculas y *strip* de espacios, p. ej. `"Science Fiction"` → `"sciencefiction"` - Concatenación en `tags` (string única por película)

Modelo. - Vectorización: `CountVectorizer(max_features=5000, stop_words='english')` - Similitud: matriz `cosine_similarity(vectors)` - Dada una película, se ordenan las demás por similitud y se devuelven las *top-k* (omitiendo la propia).

Exportación (para front estático). - `movies.json` → `[{ id, title, slug }]` - `recs_top20.json` → `{ "<id>": [{ id, title, score }... ] }` - Peso aprox.: ~300 KB + ~5 MB

Front estático. - Un `index.html` puro (HTML+CSS+JS) que: - Carga los JSON y muestra recomendaciones - (Opcional) pide `poster_path` a TMDB con **API Key** y cachea en `localStorage` - Estructura de hosting:
`public/recommender/index.html`
`public/recommender/data/movies.json`
`public/recommender/data/recs_top20.json`

Complejidad y desempeño. - Entrenamiento: $O(N \cdot V)$ para vectorización (N=películas, V=vocabulario).
- Consulta: O(N) por fila para obtener la fila de similitud ya precomputada (se lee de `similarity` en memoria durante la construcción; para el front se usa la tabla pre-ordenada `recs_top20.json` → O(1) por consulta).

Limitaciones. - Content-based: tiende a recomendar "más de lo mismo". - No usa señales de usuario (ratings/clics). - Idioma/stopwords: `english`; si el *overview* está en otro idioma, conviene cambiar o limpiar mejor.

Posibles mejoras. - TF-IDF y/o lematización - Aumentar `max_features` con más RAM - Mezclar con un colaborativo ligero (si se disponen de ratings) - Recalcular `recs_topK` por subgénero/colecciones

[1]:
```
!pip -q install pandas scikit-learn
```

[5]:
```
import os, ast, json, difflib, pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from urllib.request import urlretrieve

os.makedirs("/content/data", exist_ok=True)
MOVIES_URL  = "https://raw.githubusercontent.com/rajeevratan84/dataset-repo/
 ↪master/tmdb_5000_movies.csv"
CREDITS_URL = "https://raw.githubusercontent.com/rajeevratan84/dataset-repo/
 ↪master/tmdb_5000_credits.csv"
movies_path  = "/content/tmdb_5000_movies.csv"
credits_path = "/content/tmdb_5000_credits.csv"

for url, path in [(MOVIES_URL, movies_path),(CREDITS_URL, credits_path)]:
    if not os.path.exists(path):
        urlretrieve(url, path)

movies  = pd.read_csv(movies_path)
credits = pd.read_csv(credits_path)

movies = movies[['id','title','overview','genres','keywords']].copy()
credits = credits[['title','cast','crew']].copy()
df = movies.merge(credits, on='title', how='inner')
for col in ['overview','genres','keywords','cast','crew']:
    df[col] = df[col].fillna('[]')
df['overview'] = df['overview'].replace('[]','').fillna('')
len(df), df.head(2)
```

[5]:
```
(4809,
       id                                       title  \
 0  19995                                     Avatar
 1    285  Pirates of the Caribbean: At World's End

                                             overview  \
 0  In the 22nd century, a paraplegic Marine is di…
 1  Captain Barbossa, long believed to be dead, ha…

                                               genres  \
 0  [{"id": 28, "name": "Action"}, {"id": 12, "nam…
 1  [{"id": 12, "name": "Adventure"}, {"id": 14, "…

                                             keywords  \
 0  [{"id": 1463, "name": "culture clash"}, {"id":…
 1  [{"id": 270, "name": "ocean"}, {"id": 726, "na…
```

```
                                                  cast  \
0  [{"cast_id": 242, "character": "Jake Sully", "…
1  [{"cast_id": 4, "character": "Captain Jack Spa…

                                                  crew
0  [{"credit_id": "52fe48009251416c750aca23", "de…
1  [{"credit_id": "52fe4232c3a36847f800b579", "de…  )
```

[6]:
```
def parse_name_list(text):
    try:
        items = ast.literal_eval(text)
        return [d.get('name','') for d in items if isinstance(d, dict) and d.
 ↪get('name')]
    except Exception:
        return []

def parse_cast_top_n(text, n=3):
    try:
        items = ast.literal_eval(text)
        out=[]
        for d in items:
            if isinstance(d, dict) and d.get('name'):
                out.append(d['name'])
            if len(out)>=n: break
        return out
    except Exception:
        return []

def parse_director(text):
    try:
        items = ast.literal_eval(text)
        for d in items:
            if isinstance(d, dict) and d.get('job','').lower()=='director' and␣
 ↪d.get('name'):
                return [d['name']]
        return []
    except Exception:
        return []

def norm(tokens):
    out=[]
    for t in tokens:
        t=t.strip().lower().replace(' ','')
        if t: out.append(t)
    return out

def overview_tokens(s):
```

```
    s=(s or '').lower()
    return [w.strip('.,;:!?()[]{}"\'') for w in s.split() if w.strip('.,;:!?
 ↪()[]{}"\'')]

df['genres_list']   = df['genres'].map(parse_name_list).map(norm)
df['keywords_list'] = df['keywords'].map(parse_name_list).map(norm)
df['cast_list']     = df['cast'].map(lambda x: norm(parse_cast_top_n(x,3)))
df['director_list'] = df['crew'].map(parse_director).map(norm)
df['overview_list'] = df['overview'].map(overview_tokens).map(norm)

df['tags'] = (df['overview_list'] + df['genres_list'] +
              df['keywords_list'] + df['cast_list'] + df['director_list'])
model_df = df[['id','title','tags']].copy()
model_df['tags'] = model_df['tags'].map(lambda xs: ' '.join(xs))
model_df = model_df.dropna().reset_index(drop=True)
model_df.head(3)
```

[6]:

	id	title	tags
0	19995	Avatar	in the 22nd century a paraplegic marine is dis…
1	285	Pirates of the Caribbean: At World's End	captain barbossa long believed to be dead has …
2	206647	Spectre	a cryptic message from bond’s past sends him o…

[7]:
```
cv = CountVectorizer(max_features=5000, stop_words='english')
vectors = cv.fit_transform(model_df['tags'])
sim = cosine_similarity(vectors)

def topk_for_index(i, k=20):
    scores = list(enumerate(sim[i]))
    scores.sort(key=lambda x:x[1], reverse=True)
    # saltar el propio i
    out=[]
    for j, s in scores:
        if j==i: continue
        out.append((j, float(s)))
        if len(out)>=k: break
    return out

# construir índices
id_list    = model_df['id'].astype(int).tolist()
title_list = model_df['title'].tolist()
title_to_idx = {t.lower():i for i,t in enumerate(title_list)}
```

```
def slugify(s):
    import re, unicodedata
    s = unicodedata.normalize('NFKD', s).encode('ascii','ignore').
 ↪decode('ascii')
    s = s.lower()
    s = re.sub(r'[^a-z0-9]+','-', s).strip('-')
    return s

movies_json = [
    {"id": int(i), "title": t, "slug": slugify(t)}
    for i,t in zip(id_list, title_list)
]

recs_map = {}
for i, (mid, t) in enumerate(zip(id_list, title_list)):
    knn = topk_for_index(i, k=20)
    recs_map[str(mid)] = [
        {"id": int(id_list[j]), "title": title_list[j], "score": round(s,6)}
        for j,s in knn
    ]

# exportar
os.makedirs("/content/export", exist_ok=True)
with open("/content/export/movies.json","w",encoding="utf-8") as f:
    json.dump(movies_json, f, ensure_ascii=False)

with open("/content/export/recs_top20.json","w",encoding="utf-8") as f:
    json.dump(recs_map, f, ensure_ascii=False)

!ls -lh /content/export && echo " Listo: descarga movies.json y recs_top20.
 ↪json"
```

```
total 6.1M
-rw-r--r-- 1 root root 331K Oct 22 19:11 movies.json
-rw-r--r-- 1 root root 5.8M Oct 22 19:11 recs_top20.json
 Listo: descarga movies.json y recs_top20.json
```