
بسم الله الرحمن الرحیم

بازیابی اطلاعات

پروژه: بررسی الگوریتم و پیاده‌سازی سیستم‌های پشنهاددهنده (توصیه‌گر)

آرمان آزادی

زمستان ۱۴۰۴

فهرست مطالب

۱. مقدمه

این پروژه تلاشی است برای درک عمیق یک مفهوم اصلی سیستم‌های بازیابی اطلاعات: سیستم‌های توصیه‌گر که تجربه کاربری را شخصی‌سازی می‌کنند.

ما با یک چالش اصلی در حوزه بازیابی اطلاعات روبرو هستیم: طراحی یک سیستم توصیه‌گر کتاب برای پیشنهاد دادن کتاب‌های مناسب به کاربران بر اساس سلیقه آن‌ها. هدف اصلی این پروژه، درک عمیق و همچنین پیاده‌سازی عملی روش‌های فیلترینگ مشارکتی (Collaborative Filtering) بر روی یک مجموعه داده واقعی است. تمام مراحل با زبان پایتون و بدون استفاده از کتابخانه‌های آماده‌ی توصیه‌گر و صرفاً با استفاده از جبر خطی و محاسبات ماتریسی پیاده‌سازی شده است. این گزارش شرح کاملی از روند انجام پروژه، چالش‌هایی که با آن‌ها روبرو شده و نتایجی است که به دست آورده شده است.

۲. بخش اول: توصیف و تحلیل داده‌ها

اولین قدم شناخت دقیق داده‌هاست. داده‌های ارائه شده در این تمرین مربوط به یک سیستم کتابخوانی است.

ساختار فایل‌های ورودی

مجموعه داده شامل چندین فایل CSV بود که آن‌ها را با استفاده از کتابخانه pandas بررسی کرده:

interactions.csv: این مهم‌ترین فایل پروژه بوده. شامل ستون‌های user_id، book_id و rating است. نکته‌ای که در همان ابتدا توجه را جلب می‌کند، وجود تعداد زیادی امتیاز صفر است. در سیستم‌های توصیه‌گر، امتیاز صفر معمولاً به معنی "عدم علاقه" نیست، بلکه می‌تواند به معنی تعامل ضمنی (Implicit Feedback) مثل کلیک کردن یا دیدن صفحه کتاب باشد. این موضوع در بخش پیاده‌سازی بسیار تأثیرگذار بود.

books.csv: این فایل شامل اطلاعات متنی کتاب‌ها مثل عنوان، نویسنده و ناشر است. اگرچه در روش‌های فیلترینگ مشارکتی (Collaborative Filtering) که در بخش ۳ استفاده کردیم نیازی به این محتوا نداریم، اما این فایل برای درک اینکه آیتم‌ها چه هستند مفید است.

users.csv و test_users.csv: لیست کاربران کل سیستم و کاربرانی که باید برای آن‌ها توصیه تولید کنیم.

gold_interactions.csv: این فایل شامل پاسخ‌های صحیح (Ground Truth) برای کاربران تست بود که برای ارزیابی نهایی از آن استفاده شده است.

در (شکل ۱) میتوان کلاس پیکربندی برای قسمت پیاده‌سازی سیستم توصیه‌گر (section three) و در (شکل ۲) برای قسمت ارزیابی سیستم توصیه‌گر (section four) را دید:

```python
class Config:
    TRAIN_INTERACTIONS = 'interactions.csv'
    TEST_USERS = 'test_users.csv'

    # Columns
    USER_ID_COL = 'user_id'
    BOOK_ID_COL = 'book_id'
    RATING_COL = 'rating'

    TOP_N = 10
```

شکل ۱

```python
class Config:
    GOLD_FILE = 'gold_interactions.csv'

    USER_BASED_FILE = 'user_based_results.csv'
    ITEM_BASED_FILE = 'item_based_results.csv'

    # Evaluation constraints
    K_PRECISION = [5, 10]
    NDCG_K = 10
```

شکل ۲

چالش پراکندگی داده‌ها (Sparsity)

یکی از اولین بررسی‌های آماری که انجام شده، محاسبه ابعاد ماتریس تعاملات بود.

تعداد کاربران یکتا: ۲۲۸۵

تعداد کتاب‌های یکتا: ۶۸۸۰

تعداد کل خانه‌های ماتریس: 2285 * 6880 = 15, 720, 800

اما تعداد رکوردهای موجود در فایل interactions.csv بسیار کمتر از این مقدار بود. این یعنی ما با یک ماتریس بسیار تنک (Sparse) روبرو هستیم. این پراکندگی باعث می‌شود که بسیاری از الگوریتم‌های ساده نتوانند همسایگان مشابهی پیدا کنند، چون همپوشانی بین کاربران بسیار کم است.

۳. بخش دوم: پیاده‌سازی سیستم توصیه‌گر

هدف این بود که برای کاربرانی که در test_users.csv مشخص شده‌اند، ۱۰ کتاب پیشنهاد دهیم. طبق خواسته پروژه، از روش‌های مبتنی بر Collaborative Filtering استفاده شده است.

رویکرد کلی و ماتریس تعاملات

به جای استفاده از کتابخانه‌های آماده، از جبر خطی با numpy استفاده شده است.

ابتدا یک ماتریس تعاملات R با ابعاد $N\{users\} \times M\{books\}$ ساخته میشود(سطر همان کاربر و ستون کتابها).

نکته بسیار مهمی که در اینجا رعایت شده است، مدیریت داده‌های صفر بود. اگر فقط امتیازات غیرصفر را در نظر می‌گرفتیم، بسیاری از اطلاعات از دست می‌رفت. پس ماتریس را با مقادیر اصلی پر کردم.

```
30      # 4. Build Interaction Matrix R and Seen Mask
31      # R: Stores the ratings (0-10)
32      # seen_mask: Stores True if ANY interaction happened (even if rating is 0)
33      R = np.zeros((n_users, n_books), dtype=np.float32)
34      seen_mask = np.zeros((n_users, n_books), dtype=bool)
35
36      # Efficient filling
37      u_idxs = [user_to_idx[u] for u in interactions[Config.USER_ID_COL]]
38      b_idxs = [book_to_idx[b] for b in interactions[Config.BOOK_ID_COL]]
39      r_vals = interactions[Config.RATING_COL].values
40
41      R[u_idxs, b_idxs] = r_vals
42      seen_mask[u_idxs, b_idxs] = True  # Mark as seen regardless of rating value
43
```

شکل ۳

محاسبه شباهت

شباهت کسینوسی (Cosine Similarity) مدنظر است که فرمول آن به صورت برداری: شباهت بین سطر u و سطر v در ماتریس امتیازات محاسبه شد. این کار را با ضرب ماتریسی نرمال‌شده انجام میدهیم تا سرعت اجرا بالا برود.

$$\text{Sim}(U, V) = \frac{U.V}{||U|| \times ||V||}$$

```python
def compute_cosine_similarity(M, kind='user'):
    """
    Computes Cosine Similarity: DotProduct(A, B) / (|A|*|B|)
    """
    print(f"Calculating {kind}-based Cosine Similarity:")

    if kind == 'user':
        # Sim(u, v) = R[u] . R[v] / (|R[u]| * |R[v]|)
        row_norms = np.sqrt(np.sum(M**2, axis=1, keepdims=True))
        row_norms[row_norms == 0] = 1.0 # Avoid div/0
        M_norm = M / row_norms
        sim_matrix = np.dot(M_norm, M_norm.T)

    elif kind == 'item':
        # Sim(i, j) = R[:,i] . R[:,j] / (|R[:,i]| * |R[:,j]|)
        col_norms = np.sqrt(np.sum(M**2, axis=0, keepdims=True))
        col_norms[col_norms == 0] = 1.0
        M_norm = M / col_norms
        sim_matrix = np.dot(M_norm.T, M_norm)

    np.fill_diagonal(sim_matrix, 0)
    return sim_matrix
```

شکل ۴

پیاده‌سازی روش کاربر-محور (User-Based)

ایده: " کاربران شبیه به من چه کتاب‌هایی خوانده‌اند که من نخوانده‌ام؟"

برای این کار مراحل زیر را طی کردم:

1. **محاسبه شباهت :**شباهت کسینوسی (Cosine Similarity) را بین سطرهای ماتریس (کاربران) محاسبه میکنیم. امتیاز پیش‌بینی شده برای کاربر u و کتاب i برابر است با میانگین وزن‌دار امتیازاتی که "همسایگان" کاربر u به کتاب i داده‌اند.

2. **پیش‌بینی :**امتیاز کتاب‌ها را با ضرب ماتریس شباهت در ماتریس امتیازات محاسبه میکنیم. این کار معادل میانگین‌گیری وزن‌دار از نظرات همسایگان است.

$$\mathbf{P = Sim_{user} \times R}$$

پیاده‌سازی روش آیتم-محور (Item-Based)

ایده: " کتاب‌های شبیه به کتاب‌هایی که قبلاً خوانده‌ام کدامند؟"

این بار شباهت را بین ستون‌های ماتریس محاسبه میکنیم. یعنی دو کتاب چقدر توسط کاربران یکسان، امتیاز مشابه گرفته‌اند.

چالش اصلی در اینجا این بود که به دلیل تنکی ماتریس، احتمال اینکه دو کتاب دقیقاً توسط کاربران مشترکی خوانده شده باشند کم است، که باعث می‌شود شباهت آیتم-آیتم اغلب صفر شود(شکل ۵).

$$\mathbf{P = R \times Sim_{item}}$$

```python
def predict(R, sim_matrix, kind='user'):
    print(f"Predicting ratings using {kind}-based method:")
    if kind == 'user':
        pred = np.dot(sim_matrix, R)
    elif kind == 'item':
        pred = np.dot(R, sim_matrix)
    return pred
```

شکل ۵

مدیریت داده‌های دیده شده (Seen Items)

یکی از الزامات پروژه این بود که پیشنهادها باید از بین Unseen Items باشند.

برای این کار یک ماتریس ماسک (seen_mask) ایجاد میکنیم. هر خانه‌ای که کاربر در فایل اصلی تعاملی با آن داشت (حتی با امتیاز ۰)، در این ماسک True شد. هنگام تولید لیست نهایی ۱۰ تایی، امتیاز آیتم‌های دیده شده را به منفی بی‌نهایت (-inf) تغییر میدهیم تا در مرتب‌سازی نهایی ظاهر نشوند(شکل ۶).

```python
def save_recommendations(seen_mask, pred_matrix, user_to_idx, idx_to_book, target_uids, filename):
    results = []

    print(f"Extracting top {Config.TOP_N} recommendations:")

    for uid in target_uids:
        if uid not in user_to_idx:
            results.append({'user_id': uid, 'recommendations': ""})
            continue

        u_idx = user_to_idx[uid]
        user_preds = pred_matrix[u_idx]

        # CORRECT FILTERING:
        # Use the seen_mask to filter out ALL interacted items (including 0 ratings)
        user_seen_bool = seen_mask[u_idx]
        user_preds[user_seen_bool] = -np.inf

        # Get Top N Indices
        top_indices = np.argsort(user_preds)[-Config.TOP_N:][::-1]

        # Map back to IDs
        recs = [idx_to_book[i] for i in top_indices]
```

شکل ۶

ذخیره‌سازی نتایج

در نهایت نتایج مربوط به پیشنهادات را در دو فایل جداگانه item_based_results.csv و user_based_results.csv ذخیره میکنم که بتوان در مرحله ارزیابی آن را با مرجع درستی مقایسه کرد.

۴. بخش سوم: ارزیابی و تحلیل نتایج

پس از اجرای مدل‌ها و تولید فایل‌های user_based_results.csv و item_based_results.csv، نوبت به سنجش عملکرد رسید.

معیارهای ارزیابی

اسکریپت evaluation.ipynb معیارهای زیر را محاسبه میکند:

- **Precision@K (P@5, P@10)**: دقت سیستم در ۵ و ۱۰ پیشنهاد اول. یعنی چند درصد پیشنهادها واقعاً در لیست gold کاربر بوده‌اند.
- **NDCG**: معیاری که رتبه پیشنهاد را هم در نظر می‌گیرد. اگر کتاب محبوب کاربر را در رتبه ۱ بیاوریم، امتیاز بیشتری دارد تا رتبه ۱۰.
- **Spearman Correlation**: همبستگی رتبه‌بندی ما با رتبه‌بندی واقعی.

نتایج عددی به دست آمده

جدول ۱

معیار (Metric)	روش کاربر-محور (User-Based)	روش آیتم-محور (Item-Based)
P@5	**0.0384**	0.0056
P@10	**0.0298**	0.0060
NDCG	**0.0384**	0.0055
Spearman	-0.0026	-0.0040

```
--- Evaluating User-Based CF ---
Results for User-Based CF:
Mean P@5:      0.0384
Mean P@10:     0.0298
Mean NDCG:     0.0384
Mean Spearman: -0.0026

--- Evaluating Item-Based CF ---
Results for Item-Based CF:
Mean P@5:      0.0056
Mean P@10:     0.0060
Mean NDCG:     0.0055
Mean Spearman: -0.0040
```

شکل ۷

تحلیل چرایی تفاوت عملکرد روش‌ها

همانطور که می‌بینید، روش User-Based حدود ۶ تا ۷ برابر بهتر از Item-Based عمل کرده است. چرا؟

۱. **مشکل هم‌پوشانی آیتم‌ها** :تعداد کتاب‌ها (۶۸۸۰) نسبت به تعداد کاربران زیاد است. احتمال اینکه دو کتاب خاص دقیقاً توسط یک مجموعه کاربر خوانده شده باشند بسیار پایین است. این باعث می‌شود ماتریس شباهت آیتم-آیتم بسیار خالی باشد و الگوریتم نتواند همسایگان درستی برای کتاب‌ها پیدا کند.

۲. **رفتار کاربران** :کاربران معمولاً تمایل دارند کتاب‌های محبوب (Popular) را بخوانند. پیدا کردن کاربرانی که سلیقه مشابه در کتاب‌های پرفروش دارند راحت‌تر است. روش User-Based از این "خوشه‌های رفتاری" بهتر استفاده می‌کند.

۳. **تأثیر داده‌های صفر** :استفاده از داده‌های ضمنی (امتیاز ۰) به User-Based کمک کرد تا پروفایل کاربران را پرتر کند، اما در Item-Based به دلیل تنوع زیاد کتاب‌ها، تاثیر کمتری داشت.

۵. نتیجه‌گیری نهایی

در این پروژه، ما چرخه کامل یک سیستم توصیه‌گر را طی کردیم.

- در بخش توصیه‌گر، دریافتیم که برای داده‌های پراکنده مثل کتاب، رویکرد User-Based Collaborative Filtering کارایی بسیار بالاتری دارد.
- همچنین یاد گرفتیم که پیش‌پردازش درست داده‌ها (مثل مدیریت امتیازهای صفر و فیلتر کردن کتاب‌های دیده شده) تاثیر حیاتی بر کیفیت خروجی نهایی دارد. اگر این موارد را رعایت نمی‌کردیم، Precision سیستم احتمالاً زیر ۱٪ باقی می‌ماند.

این پروژه نشان داد که در دنیای واقعی، انتخاب الگوریتم مناسب (کاربر-محور در برابر آیتم-محور) کاملاً وابسته به چگالی و ساختار داده‌هاست و یک راه حل همیشگی وجود ندارد.