به نام خدای یکتا

گزارش تمرین دوم درس سیستم های تصمیم یار

نام و نام خانوادگی :

مبین رمضانی 40311415016

نام استاد :

آقای دکتر امین هاشمی

آذر 1404

سوال تمرین :

شرح مسئله: یک دیتاست به فرمت CSV به همراه تمرین در اختیار شما قرار گرفته است که حاوی نظرات ۱۰۰۰ کاربر برای ۲۰۰ فیلم است. این نظرات در بازه ۱–۵ هستند و درایه های با مقدار صفر، درایه هایی هستند که کاربر نظری در مورد آنها نداده است. هدف پیش بینی مقادیر صفر در این دیتاست است. روش ALS را به طور کامل و دقیقا مطابق با مراحل ارائه شده در اسلایدهای درس پیاده سازی کنید.

در پایان مقادیر به دست آمده توسط روش ALS را گرد کنید. به این صورت که اگر مقدار اعشار کمتر از ۰.۵ بود به سمت پایین و در غیراینصورت به سمت بالا گرد کنید.

در پایان نیز خطای ماتریس پیش بینی شده با ماتریس اصلی را بر اساس Sum Square Error (SSE) گزارش کنید.

حل تمرین :

کدی که نوشته شده بصورت زیر هست :

```
import numpy as np
import pandas as pd

def SSE(R_true, R_pred_round):
    mask = (R_true != 0)
    diff = R_true[mask] - R_pred_round[mask]
    return np.sum(diff ** 2)


def ALS(R, k, lambda_reg, n_iters):
    np.random.seed(42)

    m, n = R.shape

    U = np.random.rand(m, k) * 5.0
    V = np.random.rand(n, k) * 5.0

    observed_mask = (R != 0)

    for it in range(n_iters):

        for i in range(m):
            idx_items = np.where(observed_mask[i, :])[0]
            if idx_items.size == 0:
                continue
```

```
            V_i = V[idx_items, :]
            r_i = R[i, idx_items]

            A = V_i.T @ V_i + lambda_reg * np.eye(k)
            b = V_i.T @ r_i

            U[i, :] = np.linalg.solve(A, b)

        for j in range(n):
            idx_users = np.where(observed_mask[:, j])[0]
            if idx_users.size == 0:
                continue

            U_j = U[idx_users, :]
            r_j = R[idx_users, j]

            A = U_j.T @ U_j + lambda_reg * np.eye(k)
            b = U_j.T @ r_j

            V[j, :] = np.linalg.solve(A, b)

        R_hat = U @ V.T
        R_hat_clipped = np.clip(R_hat, 1, 5)
        R_hat_round = np.floor(R_hat_clipped + 0.5)
        sse = SSE(R, R_hat_round)
        print(f"Iteration {it+1}/{n_iters} => SSE : {sse:.2f}")

    return U, V


df = pd.read_csv("dataset/matrix_1000x200_sparse40.csv")
R = df.values.astype(float)
print(f"Shape of R: {R.shape[0]}x{R.shape[1]}")

K = 130      # تعداد فاکتورهای نهان
LAMBDA = 0.1  # ضریب منظم سازی (λ)
N_ITERS = 15  # تعداد تکرار ALS

U, V = ALS(R, k=K, lambda_reg=LAMBDA, n_iters=N_ITERS)

R_hat = U @ V.T

R_hat_clipped = np.clip(R_hat, 1, 5)

R_hat_round = np.floor(R_hat_clipped + 0.5)

sse = SSE(R, R_hat_round)
print("Final SSE:", sse)

df_out = pd.DataFrame(R_hat_round)
df_out.to_excel("output/R_completed_round.xlsx", index=False, header=False)
```

توضیح و اجرای کد :

```
import numpy as np
import pandas as pd
```

در ابتدای پیاده‌سازی، دو کتابخانه‌ی **NumPy** و **Pandas** با نام های مستعار **np** و **pd** فراخوانی شده‌اند. کتابخانه‌ی **NumPy** به‌عنوان ابزار اصلی محاسبات عددی و ماتریسی در پایتون مورد استفاده قرار می‌گیرد و عملیات‌هایی نظیر تولید مقادیر تصادفی، ضرب ماتریسی، اعمال توابع روی کل آرایه‌ها، و حل دستگاه‌های معادلات خطی را فراهم می‌کند. از آنجا که الگوریتم **ALS** مبتنی بر محاسبات خطی روی ماتریس امتیازها و ماتریس‌های فاکتورهای نهان است، استفاده از **NumPy** برای پیاده‌سازی کارا و بهینه‌ی این محاسبات ضروری است.

کتابخانه‌ی **Pandas** نیز برای کار با داده‌های جدولی به‌کار گرفته شده است. در این پیاده‌سازی، **Pandas** وظیفه‌ی خواندن ماتریس امتیازها از فایل **CSV** و در انتها ذخیره‌سازی ماتریس تکمیل‌شده در قالب فایل اکسل را بر عهده دارد. بدین ترتیب، **NumPy** نقش «هسته‌ی محاسبات ماتریسی» و **Pandas** نقش «مدیریت و ورود/خروج داده‌ها» را در پیاده‌سازی الگوریتم **ALS** ایفا می‌کنند.

```
def SSE(R_true, R_pred_round):
    mask = (R_true != 0)
    diff = R_true[mask] - R_pred_round[mask]
    return np.sum(diff ** 2)
```

تابع **SSE** برای محاسبه‌ی مجموع مربعات خطا **(Sum of Squared Errors - SSE)** بین ماتریس امتیازهای واقعی و ماتریس امتیازهای پیش‌بینی‌شده به‌کار می‌رود. در این تابع، ورودی **R_true** ماتریس امتیازهای اصلی (شامل مقادیر صفر به‌عنوان «عدم امتیازدهی») و **R_pred_round** ماتریس امتیازهای پیش‌بینی‌شده و گردشده است.

در قدم اول، با استفاده از عبارت **mask = (R_true != 0)** یک ماسک بولی ساخته می‌شود که در آن، تنها خانه‌هایی که امتیاز واقعی دارند (درایه‌های غیرصفر ماتریس اصلی) به‌صورت **True** علامت‌گذاری می‌شوند. بدین ترتیب، خطا فقط روی داده‌های مشاهده‌شده محاسبه می‌شود و خانه‌هایی که در ماتریس اولیه برابر با صفر بوده‌اند (داده‌ی گمشده) در محاسبه‌ی خطا لحاظ نمی‌گردند.

در ادامه، اختلاف بین مقادیر واقعی و مقادیر پیش‌بینی‌شده تنها در همین موقعیت‌های دارای امتیاز واقعی محاسبه می‌شود**(diff = R_true[mask] - R_pred_round[mask])** . سپس مربع این اختلاف‌ها گرفته شده و با استفاده از **np.sum(diff ** 2)**جمع می‌شوند. خروجی تابع، مقدار نهایی **SSE** است که به‌عنوان معیار خطای کلی مدل روی داده‌های واقعی (غیرصفر) مورد استفاده قرار می‌گیرد.

```python
def ALS(R, k, lambda_reg, n_iters):
    np.random.seed(42)

    m, n = R.shape

    U = np.random.rand(m, k) * 5.0
    V = np.random.rand(n, k) * 5.0

    observed_mask = (R != 0)

    for it in range(n_iters):

        for i in range(m):
            idx_items = np.where(observed_mask[i, :])[0]
            if idx_items.size == 0:
                continue


            V_i = V[idx_items, :]
            r_i = R[i, idx_items]

            A = V_i.T @ V_i + lambda_reg * np.eye(k)
            b = V_i.T @ r_i

            U[i, :] = np.linalg.solve(A, b)

        for j in range(n):
            idx_users = np.where(observed_mask[:, j])[0]
            if idx_users.size == 0:
                continue

            U_j = U[idx_users, :]
            r_j = R[idx_users, j]

            A = U_j.T @ U_j + lambda_reg * np.eye(k)
            b = U_j.T @ r_j

            V[j, :] = np.linalg.solve(A, b)

        R_hat = U @ V.T
        R_hat_clipped = np.clip(R_hat, 1, 5)
        R_hat_round = np.floor(R_hat_clipped + 0.5)
        sse = SSE(R, R_hat_round)
        print(f"Iteration {it+1}/{n_iters} => SSE : {sse:.2f}")

    return U, V
```

تابع ALS پیاده‌سازی الگوریتم Alternating Least Squares برای فاکتورگیری ماتریس امتیازها است. ورودی این تابع شامل ماتریس امتیازها R، تعداد فاکتورهای نهان k، ضریب منظم‌سازی lambda_reg و تعداد تکرارهای الگوریتم

n_iters است. در ابتدا با استفاده از np.random.seed(42) بذر تولید اعداد تصادفی ثابت در نظر گرفته می‌شود تا نتایج قابل تکرار باشند. سپس ابعاد ماتریس امتیازها (تعداد کاربران و تعداد آیتم‌ها) استخراج شده و دو ماتریس U و V با ابعاد m×k و n×k به صورت تصادفی مقداردهی اولیه می‌شوند. این دو ماتریس نمایانگر بردارهای ویژگی پنهان کاربران و آیتم‌ها هستند. علاوه بر این، با استفاده از observed_mask = (R != 0) ماسکی تعریف می‌شود که محل درایه‌های دارای امتیاز واقعی (غیرصفر) را مشخص می‌کند تا در به‌روزرسانی‌ها فقط این داده‌های مشاهده‌شده در نظر گرفته شوند.

هسته‌ی اصلی الگوریتم در یک حلقه‌ی تکرار به تعداد n_iters اجرا می‌شود. در هر تکرار، ابتدا ماتریس U به‌روزرسانی می‌شود؛ به این صورت که برای هر کاربر i، ابتدا مجموعه‌ی آیتم‌هایی که آن کاربر به آن‌ها امتیاز داده است با استفاده از ماسک استخراج می‌گردد. در صورتی که کاربری هیچ امتیازی ثبت نکرده باشد، آن سطر نادیده گرفته می‌شود. سپس زیرماتریس V_i (شامل فاکتورهای آیتم‌های امتیاز داده‌شده) و بردار امتیازهای متناظر r_i از ماتریس R تشکیل می‌شوند. در ادامه، ماتریس

$$A = V_i^T V_i + \lambda I_k$$

و بردار

$$b = V_i^T r_i$$

محاسبه می‌شوند که متناظر با حل مسئله‌ی کمترین مربعات با منظم‌سازی Ridge هستند. در نهایت، با حل دستگاه خطی $AU_i^T = b$ به‌وسیله‌ی تابع np.linalg.solve، بردار فاکتورهای نهان کاربر i در سطر متناظر ماتریس U به‌روزرسانی می‌شود.

در گام بعدی و در همان تکرار، ماتریس V به‌صورت مشابه به‌روزرسانی می‌گردد. برای هر آیتم j، مجموعه‌ی کاربرانی که به آن آیتم امتیاز داده‌اند تعیین می‌شود و در صورت فقدان امتیاز، آن آیتم نیز نادیده گرفته می‌شود. سپس زیرماتریس U_j (شامل بردارهای ویژگی کاربرانِ امتیازدهنده) و بردار امتیازهای متناظر r_j استخراج می‌شوند. در ادامه، ماتریس

$$A = U_j^T U_j + \lambda I_k$$

و بردار

$$b = U_j^T r_j$$

محاسبه شده و با حل دستگاه $AV_j^T = b$، سطر متناظر آیتم j در ماتریس V به‌روزرسانی می‌شود. بدین ترتیب، در هر تکرار، ابتدا بردارهای ویژگی کاربران و سپس بردارهای ویژگی آیتم‌ها به صورت متناوب (Alternating) و بر اساس داده‌های مشاهده‌شده به‌روزرسانی می‌شوند.

پس از اتمام به‌روزرسانی‌های U و V در هر تکرار، ابتدا ماتریس تقریبی امتیازها به صورت R_hat = U @ V.T محاسبه می‌شود. سپس برای جلوگیری از خروج مقادیر پیش‌بینی‌شده از بازه‌ی مجاز امتیازدهی، این مقادیر در بازه‌ی [1, 5] بریده (clip) می‌شوند و با استفاده از عبارت np.floor(R_hat_clipped + 0.5) به نزدیک‌ترین عدد صحیح طبق قانون گرد کردن مورد نظر مسئله تبدیل می‌گردند. در ادامه، با فراخوانی تابع SSE، مقدار مجموع مربعات خطا بین ماتریس اصلی و ماتریس پیش‌بینی‌شده (فقط روی درایه‌های غیرصفر ماتریس اولیه) محاسبه شده و برای هر تکرار چاپ می‌شود تا روند همگرایی الگوریتم قابل مشاهده باشد. در پایان، تابع ALS دو ماتریس نهایی U و V را بازمی‌گرداند که از آن‌ها برای بازسازی ماتریس کامل‌شده‌ی امتیازها استفاده می‌شود.

```
df = pd.read_csv("dataset/matrix_1000x200_sparse40.csv")
R = df.values.astype(float)
print(f"Shape of R: {R.shape[0]}x{R.shape[1]}")
```

در خط اول، ماتریس امتیازها از روی فایل CSV خوانده می‌شود. برای این منظور از تابع read_csv کتابخانه‌ی Pandas استفاده شده است. در خط دوم خروجی این دستور یک شیء از نوع DataFrame است که داده‌های جدول‌مانند فایل CSV (ردیف‌ها به‌عنوان کاربران و ستون‌ها به‌عنوان آیتم‌ها) را در خود نگه می‌دارد. در گام بعدی، به‌منظور انجام محاسبات ماتریسی با استفاده از NumPy، داده‌های این DataFrame به یک آرایه‌ی عددی تبدیل می‌شوند. بدین ترتیب ماتریس امتیازها به صورت یک آرایه‌ی دوبعدی از نوع float در متغیر R ذخیره می‌شود. در خط سوم ابعاد ماتریس (تعداد کاربران و تعداد آیتم‌ها) چاپ می‌شود تا مشخص گردد که اندازه‌ی داده‌های ورودی صحیح خوانده شده است.

```
K = 130       # تعداد فاکتورهای نهان
LAMBDA = 0.1  # ضریب منظم‌سازی (λ)
N_ITERS = 15  # تعداد تکرار ALS
```

در این بخش سه هایپرپارامتر اصلی الگوریتم تعیین شده‌اند. متغیر K تعداد فاکتورهای نهان (ابعاد فضای پنهان ویژگی‌ها) را مشخص می‌کند؛ هرچه مقدار K بزرگ‌تر انتخاب شود، مدل توانایی بیشتری برای مدل‌کردن الگوهای پیچیده‌ی سلیقه‌ی کاربران خواهد داشت، اما در مقابل احتمال بیش‌برازش (overfitting) و همچنین زمان محاسباتی افزایش می‌یابد. پارامتر LAMBDA (λ) ضریب منظم‌سازی است که در فرمول ALS به ماتریس λI_k افزوده می‌شود و با کنترل اندازه‌ی بردارهای U و V از بیش‌برازش جلوگیری می‌کند؛ مقدار بسیار کوچک λ مدل را مستعد بیش‌برازش و مقدار بسیار بزرگ آن باعث بیش‌ازحد هموار شدن (underfitting) مدل می‌شود. پارامتر N_ITERS نیز تعداد تکرارهای اجرای گام‌های متناوب به‌روزرسانی U و V را تعیین می‌کند؛ افزایش این مقدار معمولاً تا جایی مفید است که مقدار خطا (SSE) در تکرارهای متوالی تقریباً ثابت شود و پس از آن افزایش بیشتر تکرارها تأثیر قابل‌توجهی بر دقت ندارد.

```
U, V = ALS(R, k=K, lambda_reg=LAMBDA, n_iters=N_ITERS)
R_hat = U @ V.T
R_hat_clipped = np.clip(R_hat, 1, 5)
R_hat_round = np.floor(R_hat_clipped + 0.5)
```

در این بخش از برنامه ابتدا با فراخوانی تابع ALS و ارسال ماتریس امتیازها R به همراه مقادیر انتخاب‌شده برای تعداد فاکتورهای نهان (K)، ضریب منظم‌سازی (lambda_reg) و تعداد تکرارها (n_iters)، دو ماتریس فاکتورهای نهان

کاربران و آیتم‌ها به دست می‌آید؛ به این صورت که U نمایانگر بردارهای ویژگی پنهان کاربران و V نمایانگر بردارهای ویژگی پنهان آیتم‌ها است. پس از به‌دست‌آمدن این دو ماتریس، ماتریس تقریب‌زده‌ی امتیازها با ضرب ماتریسی U در ترانهاده‌ی V ساخته می‌شود (R_hat = U @ V.T) که در واقع پیش‌بینی مدل از امتیاز هر کاربر به هر آیتم را ارائه می‌کند. از آنجا که مقادیر پیش‌بینی‌شده در R_hat ممکن است کمی از بازه‌ی معتبر امتیازدهی (۱ تا ۵) خارج شوند، در گام بعدی با استفاده از تابع np.clip این مقادیر در بازه‌ی [1, 5] محدود می‌شوند تا همه‌ی پیش‌بینی‌ها در محدوده‌ی مجاز قرار گیرند و نتیجه در ماتریس R_hat_clipped ذخیره می‌شود. در نهایت، برای تبدیل این مقادیر اعشاری به امتیازهای صحیح، از قاعده‌ی گرد کردن استفاده شده است؛ بدین صورت که با محاسبه‌ی np.floor(R_hat_clipped + 0.5) هر مقدار ابتدا ۰/۵ واحد افزایش یافته و سپس به پایین گرد می‌شود، بنابراین مقادیر با اعشار کمتر از ۰/۵ به عدد پایین‌تر و مقادیر با اعشار برابر یا بیشتر از ۰/۵ به عدد بالاتر نگاشت می‌شوند و ماتریس نهایی R_hat_round شامل امتیازهای صحیح پیش‌بینی‌شده‌ی کاربران خواهد بود.

```
sse = SSE(R, R_hat_round)
print("Final SSE:", sse)
```

در این قسمت ابتدا با فراخوانی تابع SSE، مقدار مجموع مربعات خطا بین ماتریس امتیازهای واقعی و ماتریس امتیازهای پیش‌بینی‌شده محاسبه می‌شود. در این فراخوانی، R به‌عنوان ماتریس امتیازهای اولیه (که در آن فقط درایه‌های غیرصفر به‌عنوان داده‌های واقعی در نظر گرفته می‌شوند) و R_hat_round به‌عنوان ماتریس امتیازهای نهایی پیش‌بینی و گردشده به تابع داده می‌شوند. تابع SSE همان‌طور که قبلاً توضیح داده شد، فقط روی خانه‌هایی که در ماتریس اصلی مقدار غیرصفر دارند اختلاف بین مقدار واقعی و مقدار پیش‌بینی‌شده را محاسبه کرده، مربع این اختلاف‌ها را به دست آورده و در نهایت مجموع آن‌ها را به‌عنوان یک عدد اسکالر برمی‌گرداند که نشان‌دهنده‌ی میزان خطای کلی مدل روی داده‌های مشاهده‌شده است. دستور print("Final SSE:", sse) نیز این مقدار را به‌عنوان خطای نهایی الگوریتم ALS بعد از اتمام همه‌ی تکرارها در خروجی نمایش می‌دهد تا عملکرد نهایی مدل از نظر دقت پیش‌بینی قابل ارزیابی باشد.

```
df_out = pd.DataFrame(R_hat_round)
df_out.to_excel("output/R_completed_round.xlsx", index=False, header=False)
```

در انتهای برنامه، پس از محاسبه‌ی ماتریس نهایی امتیازهای پیش‌بینی‌شده در قالب R_hat_round، لازم است این ماتریس به صورتی مناسب برای استفاده و تحلیل‌های بعدی ذخیره شود. برای این منظور، ابتدا با دستور df_out = pd.DataFrame(R_hat_round) آرایه‌ی دوبعدی NumPy به یک شیء DataFrame از کتابخانه‌ی pandas تبدیل می‌شود تا امکان ذخیره‌سازی آن در قالب‌های مختلف فراهم گردد. سپس با استفاده از تابع to_excel این DataFrame در قالب یک فایل اکسل با نام R_completed_round.xlsx در مسیر output ذخیره می‌شود. در این فراخوانی، پارامتر index=False باعث می‌شود که اندیس سطرها (شماره‌ی ردیف‌های DataFrame) در فایل اکسل نوشته نشود و پارامتر header=False نیز از ذخیره‌شدن نام ستون‌ها جلوگیری می‌کند؛ در نتیجه، فایل خروجی دقیقاً شامل یک ماتریس عددی خالص از امتیازهای پیش‌بینی‌شده است که هر سطر آن متناظر با یک کاربر و هر ستون آن متناظر با یک آیتم می‌باشد و می‌توان آن را به‌طور مستقیم در محیط‌هایی مانند Excel مشاهده و تحلیل کرد.

خروجی نهایی کد بصورت زیر هست :

```
Shape of R: 1000x200
Iteration 1/15 => SSE : 181.00
Iteration 2/15 => SSE : 50.00
Iteration 3/15 => SSE : 11.00
Iteration 4/15 => SSE : 1.00
Iteration 5/15 => SSE : 0.00
Iteration 6/15 => SSE : 0.00
Iteration 7/15 => SSE : 0.00
Iteration 8/15 => SSE : 0.00
Iteration 9/15 => SSE : 0.00
Iteration 10/15 => SSE : 0.00
Iteration 11/15 => SSE : 0.00
Iteration 12/15 => SSE : 0.00
Iteration 13/15 => SSE : 0.00
Iteration 14/15 => SSE : 0.00
Iteration 15/15 => SSE : 0.00
Final SSE: 0.0
```

این خروجی ابتدا نشان می‌دهد که ماتریس امتیازها با ابعاد 1000x200 درست خوانده شده است و سپس روند اجرای الگوریتم ALS را در 15 تکرار گزارش می‌کند. در هر تکرار مقدار SSE (مجموع مربعات خطا روی درایه‌های غیرصفر ماتریس اصلی) چاپ شده است که از 181 در تکرار اول به‌ترتیب به 50، 11، 1 و در تکرار پنجم به 0 می‌رسد و از آن به بعد تا تکرار پانزدهم در مقدار صفر ثابت می‌ماند. در پایان نیز مقدار «Final SSE: 0.0» نشان می‌دهد که مدل توانسته است پس از چند تکرار، امتیازهای موجود (غیرصفر) را بدون خطا بازتولید کند و روی داده‌های مشاهده‌شده، تقریبا یک بازسازیِ کامل انجام شده است.

Thank You...