

به نام خداک 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 دقیقاً 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 = 70      # تعداد فاکتورهای نهان
LAMBDA = 0.1 # ضریب منظم سازی (λ)
N_ITERS = 20 # 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)

```

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

در ابتدای پیاده‌سازی، دو کتابخانه‌ی NumPy و Pandas با نام‌های 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 $mask = (R_true \neq 0)$ 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.

```

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 λ_{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 می شود تا نتایج قابل تکرار باشند. سپس ابعاد ماتریس امتیازها (تعداد کاربران و تعداد آیتمها) استخراج شده و دو ماتریس U و V با ابعاد $m \times k$ و $n \times k$ به صورت تصادفی مقداردهی اولیه می شوند. این دو ماتریس نمایانگر بردارهای ویژگی پنهان کاربران و آیتمها هستند. علاوه بر این، با استفاده از $observed_mask = (R \neq 0)$ ماسکی تعریف می شود که محل درایه های دارای امتیاز واقعی (غیر صفر) را مشخص می کند تا در به روزرسانی ها فقط این داده های مشاهده شده در نظر گرفته شوند. هسته اصلی الگوریتم در یک حلقه ی تکرار به تعداد n_iters اجرا می شود. در هر تکرار، ابتدا ماتریس U به روزرسانی می شود؛ به این صورت که برای هر کاربر i ، ابتدا مجموعه ی آیتم هایی که آن کاربر به آن ها امتیاز داده است با استفاده از ماسک استخراج می گردد. در صورتی که کاربری هیچ امتیازی ثبت نکرده باشد، آن سطر نادیده گرفته می شود. سپس زیرماتریس V_i (شامل فاکتورهای آیتم های امتیاز داده شده) و بردار امتیازهای متناظر r_i از ماتریس R تشکیل می شوند. در ادامه، ماتریس

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

و بردار

$$b = V_i^T r_i$$

محاسبه می شوند که متناظر با حل مسئله ی کمترین مربعات با منظم سازی Ridge هستند. در نهایت، با حل دستگاه خطی $AU_i^T = b$ به وسیله ی تابع $np.linalg.solve$ ، بردار فاکتورهای نهان کاربر i در سطر متناظر ماتریس U به روزرسانی می شود.

در گام بعدی و در همان تکرار، ماتریس V به صورت مشابه به روزرسانی می گردد. برای هر آیتم j ، مجموعه ی کاربرانی که به آن آیتم امتیاز داده اند تعیین می شود و در صورت فقدان امتیاز، آن آیتم نیز نادیده گرفته می شود. سپس زیرماتریس U_j (شامل بردارهای ویژگی کاربران امتیاز دهنده) و بردار امتیازهای متناظر r_j استخراج می شوند. 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 به روزرسانی می شود. بدین ترتیب، در هر تکرار، ابتدا بردارهای ویژگی کاربران و سپس بردارهای ویژگی آیتمها به صورت متناوب (Alternating) و بر اساس داده های مشاهده شده به روزرسانی می شوند.

پس از اتمام به‌روزرسانی‌های U و V در هر تکرار، ابتدا ماتریس تقریبی امتیازها به صورت $\hat{R} = U @ V.T$ محاسبه می‌شود. سپس برای جلوگیری از خروج مقادیر پیش‌بینی‌شده از بازه مجاز امتیازدهی، این مقادیر در بازه $[1, 5]$ بریده (clip) می‌شوند و با استفاده از عبارت $\text{np.floor}(\hat{R}_{\text{clipped}} + 0.5)$ 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 باز می‌گرداند 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 = 70 # تعداد فاکتورهای نهان
LAMBDA = 0.1 # ضریب منظم‌سازی ( $\lambda$ )
N_ITERS = 20 # تعداد تکرار ALS
```

در این بخش سه هاپرپارامتر اصلی الگوریتم تعیین شده‌اند. متغیر K تعداد فاکتورهای نهان (ابعاد فضای پنهان و ویژگی‌ها) را مشخص می‌کند؛ هرچه مقدار K بزرگ‌تر انتخاب شود، مدل توانایی بیشتری برای مدل کردن 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 قابل توجهی 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 (λ) 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 به دست آمدن این دو ماتریس، ماتریس تقریب‌زدهی امتیازها با ضرب ماتریسی U در ترانهادهی V ساخته می‌شود ($\hat{R} = U @ V.T$) که در واقع پیش‌بینی مدل از امتیاز هر کاربر به هر آیتم را ارائه می‌کند. از آنجا که مقادیر پیش‌بینی شده در \hat{R} 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 به عنوان ماتریس امتیازهای اولیه (که در آن فقط درایه‌های غیرصفر به عنوان داده‌های واقعی در نظر گرفته می‌شوند) و `R_hat_round` به عنوان 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 به عنوان یک عدد اسکالر برمی‌گرداند که نشان‌دهنده‌ی میزان خطای کلی مدل روی داده‌های مشاهده‌شده است. دستور `print("Final SSE:", sse)` نیز این مقدار را به عنوان 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)
```

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؛ در نتیجه، فایل خروجی شامل یک ماتریس عددی خالص از امتیازهای پیش‌بینی‌شده است که هر سطر آن به جز سطر اول، 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 1/20 => SSE : 97174.00
Iteration 2/20 => SSE : 80574.00
Iteration 3/20 => SSE : 72050.00
Iteration 4/20 => SSE : 67104.00
Iteration 5/20 => SSE : 63997.00
Iteration 6/20 => SSE : 61827.00
Iteration 7/20 => SSE : 59880.00
Iteration 8/20 => SSE : 58319.00
Iteration 9/20 => SSE : 57112.00
Iteration 10/20 => SSE : 56252.00
Iteration 11/20 => SSE : 55466.00
Iteration 12/20 => SSE : 54753.00
Iteration 13/20 => SSE : 53849.00
Iteration 14/20 => SSE : 53238.00
Iteration 15/20 => SSE : 52708.00
Iteration 16/20 => SSE : 52242.00
Iteration 17/20 => SSE : 51768.00
Iteration 18/20 => SSE : 51439.00
Iteration 19/20 => SSE : 51079.00
Iteration 20/20 => SSE : 50866.00
Final SSE: 50866.0
```

در این خروجی ابتدا دیده می شود که ماتریس ورودی با ابعاد 1000×200 به درستی خوانده شده و سپس مقدار SSE در طول 20 تکرار الگوریتم ALS گزارش شده است. در تکرار اول خطا حدود 97174 است و در هر تکرار به صورت نسبتاً یکنواخت کاهش پیدا می کند؛ به طوری که بعد از چند گام اول افت شدیدی دارد و از حدود تکرار 10 به بعد کاهش ها آرام و جزئی می شود (مثلاً از حدود 56252 در تکرار 10 به حدود 50866 در تکرار 20 می رسد). این رفتار نشان می دهد که الگوریتم به خوبی در حال یادگیری ساختار داده است و با هر تکرار تقریب بهتری از ماتریس امتیازها به دست می آورد، اما پس از مدتی روند کاهش خطا به حالت تقریباً پایدار می رسد و افزایش تعداد تکرارها فقط بهبودهای کوچک ایجاد می کند. مقدار «Final SSE: 50866.0» نیز خطای نهایی مدل روی امتیازهای واقعی (فقط درایه های غیر صفر) را نشان می دهد و بیانگر این است که مدل هنوز کمی خطا دارد، اما نسبت به حالت اولیه به میزان قابل توجهی بهبود یافته است.

Thank You...