به نام خدا

گزارش تمرین اول درس سیستم های تصمیم یار

نام و نام خانوادگی :

مبین رمضانی 40311415016

نام استاد :

آقای دکتر امین هاشمی

آبان 1404

سوال تمرین :

شرح مسئله: فرض کنید شما مدیر یک سایت فروش لپ تاپ هستید و میخواهید با توجه به اولویت های هر کاربر ترتیبی از گزینه های مختلف را به هر شخص ارائه دهید. در واقع شما به دنبال طراحی یک سیستم توصیه گر شخصی سازی شده برای هر کاربر هستید.

معیارهای تصمیم‌گیری به صورت زیر تعریف شده‌اند:

معیار	توضیح	نوع
C1	قیمت(Price)	منفی
C2	عمر باتری(Battery Life)	مثبت
C3	عملکرد پردازنده (Processor Performance)	مثبت
C4	وزن(Weight)	منفی
C5	کیفیت نمایشگر(Display Quality)	مثبت

پنج مدل لپ‌تاپ مختلف در نظر گرفته شده‌اند:

لپ‌تاپ	قیمت($)	باتری (ساعت)	CPU (امتیاز)	وزن (کیلوگرم)	نمایشگر (امتیاز)
A	950	8	7.5	1.6	8
B	1200	10	9.0	1.8	9
C	800	6	6.5	1.4	7
D	1000	9	8.0	1.5	8.5
E	1100	7	8.5	1.9	8

به زبان پایتون دو تابع برای سازی دو الگوریتم AHP و TOPSIS بنویسید و این دو الگوریتم را پیاده سازی کنید. قرار است که از روش AHP برای تعیین وزن هر شاخص و از روش TOPSIS برای تصمیم گیری چند شاخصه استفاده شود.

برای هر کاربر ابتدا ماتریس های مقایسات جفتی برای الگوریتم AHP را دریافت کنید. (با مقیاس ۱ تا ۹)

در نهایت برنامه شامل دو خروجی باشد:

مقادیر وزن ها

```
C1: 0.25
C2: 0.20
C3: 0.30
C4: 0.10
C5: 0.15
```

رتبه بندی نهایی

```
Laptop B: 0.85  → 1 رتبه
Laptop D: 0.78  → 2 رتبه
Laptop A: 0.65  → 3 رتبه
Laptop E: 0.52  → 4 رتبه
Laptop C: 0.35  → 5 رتبه
```

روند را مطابق با اسلایدهای درس انجام دهید. مجاز به استفاده از تابع آماده AHP و TOPSIS نیستید.

حل تمرین :

کدی که آماده کرده ام به‌صورت زیر است :

```
import numpy as np

# --------------------- AHP ---------------------
def pairwise_matrix(n, comparisons):
    A = np.ones((n, n), dtype=float)
    for i, j, val in comparisons:
        A[i, j] = float(val)
        A[j, i] = 1.0 / float(val)
    return A

def ahp_weights(A, max_iter=100):
    A = np.array(A, dtype=float)
    n = A.shape[0]
    w_first = np.ones(n) / n
    for _ in range(max_iter):
        B = A @ A
        row_sums = B.sum(axis=1)
        w_new = row_sums / row_sums.sum()
        if np.round(w_new, 4).tolist() == np.round(w_first, 4).tolist():
            return w_new
        A = B
        w_first = w_new
    return w_first

# --------------------- TOPSIS ---------------------
def topsis(X, weights, is_benefit):
    X = np.array(X, dtype=float)
    w = np.array(weights, dtype=float)
    is_benefit = np.array(is_benefit, dtype=bool)

    R = X / (np.sqrt((X ** 2).sum(axis=0)))
    V = R * w

    A_pos = np.where(is_benefit, V.max(axis=0), V.min(axis=0))
    A_neg = np.where(is_benefit, V.min(axis=0), V.max(axis=0))
```

```python
    S_pos = np.sqrt(((V - A_pos) ** 2).sum(axis=1))
    S_neg = np.sqrt(((V - A_neg) ** 2).sum(axis=1))
    C = S_neg / (S_neg + S_pos)

    ranking = np.argsort(C)[::-1]
    return C, ranking

# -------------------- main --------------------
if __name__ == "__main__":
    comparisons = [
        (0,1,1/5), (0,2,1/9), (0,3,7), (0,4,7),
        (1,2,1/9), (1,3,5), (1,4,5),
        (2,3,9), (2,4,9),
        (3,4,1),
    ]
    A = pairwise_matrix(5, comparisons)
    w = ahp_weights(A)

    print("Weights (AHP):")
    for i, weight in enumerate(w):
        print(f"C{i+1}: {weight:.2f}")

    # Price, Battery, CPU, Weight, Display
    X = np.array([
        [950, 8, 7.5, 1.6, 8],   # A
        [1200, 10, 9.0, 1.8, 9],   # B
        [800, 6, 6.5, 1.4, 7],   # C
        [1000, 9, 8.0, 1.5, 8.5], # D
        [1100, 7, 8.5, 1.9, 8],   # E
    ])
    is_benefit = np.array([False, True, True, False, True])
    C, ranking = topsis(X, w, is_benefit)
    alternatives = np.array(["A", "B", "C", "D", "E"])

    print("\nRelative Closeness (TOPSIS):")
    for i in range(len(C)):
        print(f"Laptop {alternatives[i]}: {C[i]:.2f}")

    print("\nRanking:")
    for rank, index in enumerate(ranking, start=1):
        print(f"Laptop {alternatives[index]}: {C[index]:.2f} -> {rank} رتبه")
```

توضیحات کد :

تابع pairwise_matrix :

```
def pairwise_matrix(n, comparisons):
    A = np.ones((n, n), dtype=float)
    for i, j, val in comparisons:
        A[i, j] = float(val)
        A[j, i] = 1.0 / float(val)
    return A
```

تابع pairwise_matrix مسئول ساخت ماتریس مقایسه‌های جفتی در روش AHP است. این تابع ابتدا با استفاده از دستور np.ones یک ماتریس اولیه به نام A با ابعاد $n \times n$ ایجاد می‌کند که تمام عناصر آن مقدار 1 دارند. سپس لیست ورودی comparisons که شامل سه تایی‌های (i, j, val) است پردازش می‌شود. در حلقه‌ی تابع، مقدار val به‌عنوان میزان اهمیت معیار i نسبت به j در خانه‌ی A[i, j] قرار داده می‌شود. برای حفظ خاصیت تقارن معکوسی در مقایسه‌های جفتی، معکوس همین مقدار یعنی 1/val نیز در موقعیت A[j, i] ذخیره می‌شود. به این ترتیب، تمام مقایسه‌های جفتی وارد ماتریس می‌شوند و در نهایت ماتریس کامل‌شده‌ی A توسط تابع بازگردانده می‌شود.

تابع ahp_weights :

```
def ahp_weights(A, max_iter=100):
    A = np.array(A, dtype=float)
    n = A.shape[0]
    w_first = np.ones(n) / n
    for _ in range(max_iter):
        B = A @ A
        row_sums = B.sum(axis=1)
        w_new = row_sums / row_sums.sum()
        if np.round(w_new, 4).tolist() == np.round(w_first, 4).tolist():
            return w_new
        A = B
        w_first = w_new
    return w_first
```

تابع ahp_weights برای محاسبه‌ی وزن نهایی معیارها در روش AHP طراحی شده است. در ابتدای تابع، ماتریس ورودی A با استفاده از np.array به آرایه‌ای از نوع عددی تبدیل می‌شود تا عملیات محاسباتی روی آن بدون خطا انجام گیرد. سپس با استخراج مقدار A.shape[0] تعداد معیارها در متغیر n ذخیره می‌شود. وزن

اولیه‌ی معیارها در متغیر w_first به صورت یک بردار یکنواخت (تمام عناصر برابر 1/n) با استفاده از np.ones(n) تنظیم می‌شود. تابع در ادامه وارد یک حلقه تکراری می‌شود که حداکثر max_iter بار اجرا خواهد شد. در هر تکرار، ابتدا با عملیات ضرب ماتریسی A @ A یک ماتریس جدید به نام B محاسبه می‌شود. سپس مجموع عناصر هر ردیف ماتریس B با دستور B.sum(axis=1) در متغیر row_sums ذخیره می‌گردد. وزن‌های جدید معیارها در متغیر w_new تولید می‌شوند؛ به این صورت که هر مجموع ردیف بر مجموع کل ردیف‌ها تقسیم شده و بردار نرمال‌شده‌ی وزن‌ها به دست می‌آید.

در ادامه، شرط توقف الگوریتم بررسی می‌شود: اگر مقدارهای w_new پس از گرد کردن تا چهار رقم اعشار با مقدارهای w_first برابر باشند، یعنی وزن‌ها به حالت پایدار رسیده‌اند و تابع همان w_new را به‌عنوان خروجی بازمی‌گرداند. در غیر این صورت، ماتریس A با مقدار B جایگزین شده و وزن قبلی w_first نیز برابر با w_new قرار می‌گیرد تا تکرار بعدی ادامه یابد. در صورتی که حلقه بدون رسیدن به شرط توقف پایان یابد، مقدار نهایی w_first به عنوان وزن معیارها بازگردانده می‌شود.

تابع topsis :

```
def topsis(X, weights, is_benefit):
    X = np.array(X, dtype=float)
    w = np.array(weights, dtype=float)
    is_benefit = np.array(is_benefit, dtype=bool)

    R = X / (np.sqrt((X ** 2).sum(axis=0)))
    V = R * w

    A_pos = np.where(is_benefit, V.max(axis=0), V.min(axis=0))
    A_neg = np.where(is_benefit, V.min(axis=0), V.max(axis=0))

    S_pos = np.sqrt(((V - A_pos) ** 2).sum(axis=1))
    S_neg = np.sqrt(((V - A_neg) ** 2).sum(axis=1))
    C = S_neg / (S_neg + S_pos)

    ranking = np.argsort(C)[::-1]
    return C, ranking
```

تابع topsis برای اجرای مراحل روش TOPSIS و محاسبه‌ی نزدیکی نسبی هر گزینه نسبت به ایده‌آل مثبت و منفی به کار می‌رود. در ابتدا ماتریس تصمیم X با استفاده از np.array به آرایه‌ای عددی تبدیل می‌شود تا عملیات

ریاضی روی آن قابل انجام باشد. بردار وزن‌ها نیز در متغیر w به آرایه عددی و بردار نوع معیارها (سود یا هزینه) در is_benefit به آرایه بولی تبدیل می‌شوند.

در گام بعد، ماتریس نرمال‌شده‌ی TOPSIS با نام R با استفاده از روش vector normalization بصورت فرمول زیر محاسبه می‌شود:

$$r_{ij} = \frac{r_{ij}}{\sqrt{\sum_{k=1}^{n} x_{ik}^2}}$$

درواقع این کار با تقسیم هر ستون ماتریس X بر ریشه‌ی مجموع مربعات همان ستون انجام می‌گیرد تا همه معیارها در یک مقیاس قرار گیرند. سپس ماتریس وزن‌دار V با ضرب کردن ماتریس R در بردار وزن‌ها (w) به صورت ضرب عنصر به عنصر ساخته می‌شود.

پس از آن، دو بردار ایده‌آل مثبت و ایده‌آل منفی با نام‌های A_pos و A_neg به کمک دستور np.where استخراج می‌شوند. برای معیارهای سود (که مقدار بیشتر بهتر است)، ایده‌آل مثبت برابر با بیشترین مقدار ستون و ایده‌آل منفی برابر با کمترین مقدار آن است؛ و برای معیارهای هزینه برعکس عمل می‌شود.

در ادامه، فاصله هر گزینه از ایده‌آل مثبت در متغیر S_pos و فاصله از ایده‌آل منفی در S_neg محاسبه می‌شود. این فاصله‌ها با استفاده از ریشه‌ی مجموع مربع اختلاف عناصر ماتریس V با بردارهای ایده‌آل انجام می‌گیرد. سپس نزدیکی نسبی هر گزینه به ایده‌آل، با فرمول C = S_neg / (S_neg + S_pos) به دست می‌آید؛ به‌طوری که مقدار بالاتر C نشان‌دهنده‌ی مطلوبیت بیشتر گزینه است.

در پایان، با استفاده از np.argsort(C)[::-1] گزینه‌ها بر اساس مقدار C از بیشترین به کمترین مرتب شده و در متغیر ranking ذخیره می‌شوند. تابع در نهایت بردار نزدیکی نسبی (C) و رتبه‌بندی به‌دست‌آمده (ranking) را بازمی‌گرداند.

بخش اصلی اجرای کد :

```
if __name__ == "__main__":
    comparisons = [
        (0,1,1/5), (0,2,1/9), (0,3,7), (0,4,7),
        (1,2,1/9), (1,3,5), (1,4,5),
        (2,3,9), (2,4,9),
        (3,4,1),
    ]
    A = pairwise_matrix(5, comparisons)
    w = ahp_weights(A)

    print("Weights (AHP):")
    for i, weight in enumerate(w):
        print(f"C{i+1}: {weight:.2f}")

    # Price, Battery, CPU, Weight, Display
    X = np.array([
        [950, 8, 7.5, 1.6, 8],   # A
        [1200, 10, 9.0, 1.8, 9],   # B
        [800, 6, 6.5, 1.4, 7],   # C
        [1000, 9, 8.0, 1.5, 8.5], # D
        [1100, 7, 8.5, 1.9, 8],   # E
    ])
    is_benefit = np.array([False, True, True, False, True])
    C, ranking = topsis(X, w, is_benefit)
    alternatives = np.array(["A", "B", "C", "D", "E"])

    print("\nRelative Closeness (TOPSIS):")
    for i in range(len(C)):
        print(f"Laptop {alternatives[i]}: {C[i]:.2f}")

    print("\nRanking:")
    for rank, index in enumerate(ranking, start=1):
        print(f"Laptop {alternatives[index]}: {C[index]:.2f} -> {rank} رتبه")
```

این بخش از کد، قسمت اصلی اجرای برنامه است و در واقع نشان می‌دهد که چطور روش‌های AHP و TOPSIS روی یک مسئله‌ی انتخاب لپ‌تاپ اعمال شده‌اند. در ابتدا شرط if __name__ == "__main__": به این معنی است که کد داخل این بلاک فقط زمانی اجرا شود که این فایل مستقیما اجرا شود و نه وقتی که به عنوان یک ماژول در فایل دیگری ایمپورت می‌شود. در ادامه، لیستی به نام comparisons تعریف شده است که شامل

تعدادی سه‌تایی (i, j, val) است؛ این سه‌تایی‌ها مقایسه‌های جفتی بین معیارها را نشان می‌دهند. سپس با استفاده از تابع pairwise_matrix(5, comparisons) یک ماتریس مقایسه‌ی جفتی در متغیر A ساخته می‌شود که در آن عدد 5 نشان‌دهنده‌ی تعداد معیارها است. بعد از آن، با فراخوانی تابع ahp_weights(A) وزن نهایی هر معیار محاسبه شده و در متغیر w ذخیره می‌شود.

در ادامه، با دستور print("Weights (AHP):") عنوان مربوط به وزن‌ها چاپ می‌شود و سپس با استفاده از یک حلقه‌ی for و تابع enumerate(w)، وزن هر معیار به‌صورت جداگانه نمایش داده می‌شود. در این قسمت، متغیر i شماره‌ی معیار و متغیر weight مقدار وزن همان معیار است و با استفاده از رشته‌ی فرمت‌شده‌ی f"C{i+1}: {weight:.2f}" وزن هر معیار با دو رقم اعشار چاپ می‌شود.

در بخش بعدی، ماتریس تصمیم مربوط به گزینه‌ها در متغیر X تعریف شده است. این ماتریس با استفاده از np.array ساخته شده و هر سطر آن نشان‌دهنده‌ی یک لپ‌تاپ (A تا E) و هر ستون نشان‌دهنده‌ی یک معیار تصمیم‌گیری است. در کامنت بالای ماتریس X معیارها مشخص شده‌اند: Price (قیمت)، Battery (باتری)، CPU، Weight (وزن)، و Display (صفحه‌نمایش). بعد از آن، برداری به نام is_benefit تعریف می‌شود که مشخص می‌کند هر معیار از نوع سود (بهتر بودن با مقدار بیشتر) است یا هزینه (بهتر بودن با مقدار کمتر). در این بردار، مقدار True یعنی معیار سود و مقدار False یعنی معیار هزینه است؛ مثلا برای قیمت و وزن مقدار False در نظر گرفته شده چون مقدار کمتر مطلوب‌تر است.

در گام بعد، تابع topsis(X, w, is_benefit) فراخوانی می‌شود و خروجی آن، یعنی نزدیکی نسبی هر گزینه به ایده‌آل (C) و رتبه‌بندی گزینه‌ها (ranking) در دو متغیر جداگانه ذخیره می‌شود. سپس آرایه‌ای به نام alternatives تعریف می‌شود که برچسب متنی هر لپ‌تاپ (A تا E) را نگه می‌دارد تا در زمان چاپ نتایج از آن استفاده شود.

در ادامه با دستور print("\nRelative Closeness (TOPSIS):") عنوان مربوط به نزدیکی نسبی هر لپ‌تاپ چاپ شده و در یک حلقه‌ی for، مقدار C[i] برای هر گزینه همراه با نام آن در خروجی نمایش داده می‌شود. در پایان نیز با دستور print("\nRanking:") عنوان رتبه‌بندی چاپ شده و سپس با استفاده از یک حلقه‌ی for روی متغیر ranking، گزینه‌ها بر اساس مقدار نزدیکی نسبی مرتب شده و به ترتیب رتبه‌ها نمایش داده می‌شوند. در اینجا متغیر rank رتبه‌ی فعلی (از ۱ شروع می‌شود) و index شماره‌ی گزینه در آرایه‌ی alternatives و

بردار C است و با استفاده از رشته‌ی فرمت‌شده‌ی f"Laptop {alternatives[index]}: {C[index]:.2f} -> رتبه "هم نام لپ‌تاپ، هم مقدار نزدیکی نسبی و هم رتبه‌ی نهایی آن چاپ می‌شود. {rank}

خروجی نهایی کد به صورت زیر است :

```
Weights (AHP):
C1: 0.11
C2: 0.19
C3: 0.64
C4: 0.03
C5: 0.03

Relative Closeness (TOPSIS):
Laptop A: 0.43
Laptop B: 0.84
Laptop C: 0.16
Laptop D: 0.62
Laptop E: 0.65

Ranking:
Laptop B: 0.84 -> 1 رتبه
Laptop E: 0.65 -> 2 رتبه
Laptop D: 0.62 -> 3 رتبه
Laptop A: 0.43 -> 4 رتبه
Laptop C: 0.16 -> 5 رتبه
```