



دانشگاه اصفهان

دانشکده مهندسی کامپیووتر

گروه مهندسی هوش مصنوعی

مبانی هوش محاسباتی

تمرین سری دوم

اعضاي گروه:

ابوالفضل رنجبر

میعاد کیمیاگری

امیر طاها نجف

استاد درس:

دکتر حسین کارشناس

۱۴۰۴ بهار

۱- تمرین اول: مفاهیم و روش‌های فازی

۱-۱- تابع عضویت یک مجموعه فازی (Fuzzy Set) چیست؟

عضویت یک مجموعه فازی (Fuzzy Set) به این معناست که یک عنصر می‌تواند با درجات مختلفی به یک مجموعه تعلق داشته باشد. برخلاف مجموعه‌های قطعی (Crisp Sets) که در آن‌ها عضویت یا صفر (غیرعضو) یا یک (عضو کامل) است، در مجموعه‌های فازی مقدار عضویت عددی بین صفر و یک است (مانند $0.2, 0.7, \dots$). این مقدار نشان‌دهنده میزان تعلق یا عضویت نسبی عنصر به مجموعه است.

۱-۲- آیا یک عضویت فازی می‌تواند همزمان درست (True) و نادرست (False) باشد؟ به طور خلاصه توضیح دهید.

عضویت فازی به جای درست مطلق یا نادرست مطلق، از مفهوم درجه درستی استفاده می‌کند. یک عنصر می‌تواند به طور همزمان تا حدی در یک مجموعه فازی عضو باشد (درجه عضویت بین 0 و 1)، اما این به معنی درست و نادرست بودن همزمان به شیوه دوتایی منطق کلاسیک نیست.

۱-۳- چگونه می‌توان یک مجموعه فازی را از یک مجموعه قطعی (Crisp) تشخیص داد؟

برای تشخیص یک مجموعه فازی از یک مجموعه قطعی، باید به درجه عضویت عناصر توجه کرد. در یک مجموعه قطعی، هر عنصر فقط می‌تواند عضویت کامل (1) یا عدم عضویت کامل (0) داشته باشد. اما در یک مجموعه فازی، درجه عضویت می‌تواند هر مقداری بین 0 و 1 باشد. بنابراین اگر حداقل یک عنصر دارای درجه عضویت غیر از 0 یا 1 باشد، مجموعه مورد نظر فازی است.

۱-۴- متغیرهای واقعی زیر را از زندگی روزمره در نظر بگیرید:

- درآمد بر حسب پوند انگلستان
- سرعت بر حسب متر بر ثانیه
- یک برنامه تلویزیونی بر حسب میزان علاقه شما به تماشای آن.
- یک وعده غذایی بر حسب میزان تمایل شما به خوردن آن.
- یک چراغ راهنمایی بر حسب رنگی که روشن است.

برای هر کدام یک متغیر فازی متناظر پیشنهاد دهید. به نظر شما استفاده از متغیر فازی برای کدام یک از این پنج متغیر واقعا ضروری نیست؟ چرا؟

برای هر یک از متغیرهای واقعی می‌توان متغیر فازی متناظری در نظر گرفت: درآمد به صورت کم، متوسط یا زیاد، سرعت به صورت آهسته، معمولی یا سریع، علاقه به برنامه تلویزیونی به صورت بی‌علاقه، نسیی یا خیلی علاقه‌مند، میل به وعده غذایی به صورت بی‌میل، معمولی یا بسیار مایل، و چراغ راهنمایی بر اساس رنگ‌های قرمز، زرد و سبز. از بین این موارد، فازی‌سازی برای درآمد، سرعت و چراغ راهنمایی چندان ضروری نیست، چون این متغیرها کمی یا قطعی هستند و نیازی به متغیر فازی ندارند. در مقابل، علاقه به برنامه یا میل به غذا ذاتاً مبهم‌اند و استفاده از متغیر فازی برای آن‌ها منطقی‌تر است.

۱-۵- سیستم خبره فازی زیر را برای پیش‌بینی وضعیت هوا در نظر بگیرید:

(الف) اگر فشار برابر 10~mbar باشد مقدار عضویت فلش در دسته‌های پایین بالا یا وسط چقدر است؟ از توابع عضویت روی نمودارها استفاده کنید.

دسته پایین نمودار را در ارتفاع ۰ قطع می‌کند (چون نمودار پایین از 1000 به بعد روی محور افقی است). پس مقدار عضویت صفر می‌شود.

دسته وسط نمودار را در ارتفاع تقریباً 25~m قطع می‌کند. پس مقدار عضویت 0.25 می‌شود.

دسته بالا نمودار را در ارتفاع تقریباً 50 قطع می‌کند. پس مقدار عضویت 0.5 می‌شود.

(ب) اگر فشار با سرعت 2~m/s باشد در ساعت تغییر کند مقدار عضویت فلش در دسته‌های حرکت به پایین یا حرکت به بالا چقدر است؟

در حرکت به سمت پایین مقدار 75~m دارد و در حرکت به سمت بالا مقدار صفر را دارد.

(ج) با استفاده از مقادیر عضویت به دست آمده در بالا و میزان اطمینان قوانین داده شده در جدول درجه اطمینان در مورد صاف یا ابری بودن آسمان را محاسبه کنید.

هر قاعده را با عملگر AND تابع (\min) ارزیابی کردیم و حاصل را در ضریب اطمینان آن قانون ضرب کردیم تا قدرت هر قاعده به دست آید:

- قانون اول (اگر فشار پایین، آسمان ابری؛ $\text{Conf}=0.8$) با عضویت فشار پایین برابر صفر فعال شد، پس قدرت آن 0 شد.
- قانون دوم (اگر فشار وسط و تغییر پایین، آسمان ابری؛ $\text{Conf}=0.6$) شرط‌ها $\min(0.25, 0.75)=0.25$ را داد؛ ضرب در 0.6 شد 0.15 .
- قانون سوم (اگر فشار وسط و تغییر بالا، آسمان صاف؛ $\text{Conf}=0.6$) شرط‌ها $\min(0.25, 0)=0$ بود؛ پس قدرت آن 0 .

- قانون چهارم (اگر فشار بالا، آسمان صاف؛ $Conf=0.8$) با عضویت فشار بالا 0.5 ، فعال شد؛ ضرب در 0.8 شد 0.40 .

در پایان برای هر خروجی (ابری یا صاف) بزرگترین قدرت میان قوانین مربوطه را برداشتیم؛ برای ابری بزرگترین مقدار 0.0 و برای صاف بزرگترین مقدار 0.40 . به این ترتیب سیستم با این ورودی فازی، درجه اطمینان 0.15 برای ابری بودن و 0.40 برای صاف بودن آسمان ارائه می‌دهد.

۶-۱ سه نقطه قوت و سه نقطه ضعف سیستم‌های خبره فازی را نام ببرید.

نقاط قوت:

- قابلیت مدل‌سازی عدم قطعیت و ابهام در داده‌ها.
- توانایی استفاده از دانش انسانی به صورت قوانین فازی.
- عملکرد مناسب در شرایطی که داده‌های دقیق در دسترس نیستند.

نقاط ضعف:

- عیین توابع عضویت مناسب ممکن است دشوار و زمان بر باشد.
- طراحی پایگاه قواعد فازی می‌تواند نیازمند دانش تخصصی بالا باشد.
- توانایی یادگیری و به روزرسانی اطلاعات به صورت خودکار محدود است (در روش‌های کلاسیک فازی).

۷-۱ تفاوت‌های کلیدی بین یک سیستم Mamdani و یک سیستم TSK را بیان کنید.

سیستم‌های فازی Mamdani و TSK (Takagi-Sugeno-Kang) دو رویکرد رایج در طراحی سیستم‌های فازی هستند که تفاوت‌های کلیدی میان آن‌ها بیشتر در نوع خروجی، ساختار قواعد و کاربردها نمایان می‌شود. در سیستم Mamdani، خروجی هر قاعده به صورت یک مقدار فازی بیان می‌شود و در نهایت برای رسیدن به یک خروجی عددی، مرحله‌ای به نام واهمی‌سازی (Defuzzification) لازم است. این نوع سیستم بیشتر مناسب مدل‌سازی دانش انسانی و کاربردهای کیفی است، زیرا ساختار قواعد آن شبیه به استدلال انسانی است. اما در مقابل، سیستم TSK خروجی هر قاعده را به صورت یکتابع ریاضی (معمولًاً خطی یا ثابت) تولید می‌کند و دیگر نیازی به واهمی‌سازی ندارد. به همین دلیل، سیستم‌های TSK از نظر محاسباتی سریع‌تر و برای کاربردهایی که نیاز به دقت عددی بالا دارند، مانند کنترل سیستم‌ها یا داده‌کاوی، مناسب‌تر هستند. به طور خلاصه، Mamdani بیشتر انسان‌محور و شهودی است، در حالی که TSK داده‌محور و عددی است.

۸-۱- تفاوت بین یک برازش خطی تکه‌ای (Piecewise Linear Fit) معمولی و روش TSK چیست؟

در روش TSK ، هر قاعده فازی نه با یک بازه‌ی قطعی که با یکتابع عضویت نرم در تمام دامنه ورودی‌ها فعال می‌شود و در بخش نتیجه یکتابع خطی از ورودی‌ها ارائه می‌کند. خروجی‌نها بی سیستم میانگین وزنی این توابع خطی است که وزن هر قاعده برابر درجه‌ی فراخوانی آن است؛ بنابراین انتقال بین قطعات خطی کاملاً نرم و پیوسته بوده و هیچگاه شکستگی در سطح خروجی ایجاد نمی‌شود. این ساختار، TSK را به یک تقریب‌کننده جهانی تبدیل می‌کند و نیازی به مرحله‌ی جداگانه‌ی defuzzification مانند مدل‌های ممدانی ندارد.

در مقابل، در برازش خطی تکه‌ای (Piecewise Linear Fit) دامنه‌ی ورودی به زیر بازه‌های مجزا و قطعی تقسیم می‌شود و در هر زیر بازه یک رگرسیون خطی مستقل برازش می‌شود. هر نقطه‌ی داده تنها به یکی از این زیر بازه‌ها تعلق دارد و خطوط در مرزها به صورت سخت (crisp) به هم می‌رسند؛ هیچ مکانیزم وزنی فازی برای ترکیب نتایج وجود ندارد و در نقاط شکست ممکن است ناگهان شیب یا مقدار خروجی تغییر کند.

۹-۱- چگونه می‌توان تضاد (Conflict) را در یک سیستم مبتنی بر دانش برطرف کرد؟

در سیستم‌های فازی، تضاد به شکل کلاسیک وجود ندارد چون قوانین به طور همزمان با درجات مختلف فعال می‌شوند و خروجی‌ها به صورت نرم ترکیب می‌شوند. با این حال، اگر پایگاه قواعد ضعیف یا ناسازگار باشد، ممکن است نتایج مبهم یا غیرمنطقی ایجاد شود. به طور کلی برای رفع تضاد در سیستم‌های مبتنی بر دانش می‌توان از روش‌های زیر استفاده کرد.

- **اولویت‌بندی قوانین:** به هر قانون یک اولویت داده می‌شود و قانونی با بالاترین اولویت اجرا می‌شود. این اولویت می‌تواند طبق اهمیت قانون تعیین شود.
- **ترتیب قوانین:** سامانه استنتاج، بر اساس ترتیب قوانین در پایگاه دانش عمل کند. مثلاً سیستم همیشه اولین قانونی که با ورودی تطبیق داشته باشد اجرا می‌کند. (یا همیشه آخرین قانون)
- **وزن دهنده بی قوانین و رأی‌گیری:** به هر قانون یک وزن داده می‌شود که بیانگر اعتماد یا اهمیت آن است. در صورت تضاد، نتایج قوانین بر اساس وزن آن‌ها ترکیب می‌شود (نه حذف کامل یک قانون).

۱۰-۱- به نظر شما، سیستم‌های TSK چگونه از مشاهدات یاد می‌گیرند؟

در سیستم‌های TSK، یادگیری از مشاهدات از طریق الگوریتم‌هایی مانند رگرسیون فازی یا الگوریتم‌های یادگیری ماشین (مثلاً الگوریتم‌های خوشه‌بندی، گرادیان نزولی، یا شبکه‌های عصبی) انجام می‌شود. این الگوریتم‌ها به سیستم کمک می‌کنند تا پارامترهای توابع خروجی و قواعد فازی را از داده‌های آموزشی استخراج کند و به صورت خودکار به روزرسانی شود.

۲- تمرین دوم کنترل کننده منطق فازی برای سیستم آبیاری خودکار گیاهان

۱-۲- توابع عضویت فازی زیر را با استفاده از زبان پایتون به طور واضح و دقیق تعریف و پیاده سازی کنید:

- سطح رطوبت خاک
 - شرایط آب و هوا
 - مقدار آبیاری
- ۲-۲- توابع عضویت را ترسیم و بصری سازی کنید.

```
# Antecedent: soil moisture (0-100)
soil = ctrl.Antecedent(np.arange(0, 101, 1), 'soil')

soil['dry']    = fuzz.trapmf(soil.universe, [0, 0, 25, 35])
soil['wet']     = fuzz.trapmf(soil.universe, [65, 75, 100, 100])
soil['medium'] = fuzz.gaussmf(soil.universe, 50, 10)
sunny = ctrl.Antecedent([0, 1], 'sunny')
cloudy = ctrl.Antecedent([0, 1], 'cloudy')
rainy = ctrl.Antecedent([0, 1], 'rainy')
# Membership: exactly 0→False, 1→True
for var in (sunny, cloudy, rainy):
    var['False'] = fuzz.trimf(var.universe, [0, 0, 1])
    var['True']  = fuzz.trimf(var.universe, [0, 1, 1])
# Consequent: irrigation amount (0-100)
irrig = ctrl.Consequent(np.arange(0, 101, 1), 'irrigation')
irrig['None']   = fuzz.trimf(irrig.universe, [0, 0, 0])
irrig['Low']    = fuzz.trimf(irrig.universe, [0, 25, 50])
irrig['Medium'] = fuzz.trimf(irrig.universe, [25, 50, 75])
irrig['High']   = fuzz.trimf(irrig.universe, [50, 100, 100])
```

رطوبت خاک (soil):

مقدار ورودی بین ۰ تا ۱۰۰ است (درصد). برای این متغیر، سه تابع عضویت تعریف شده:

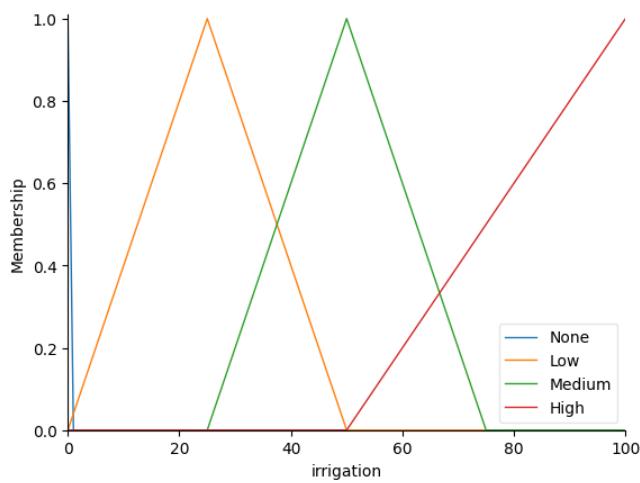
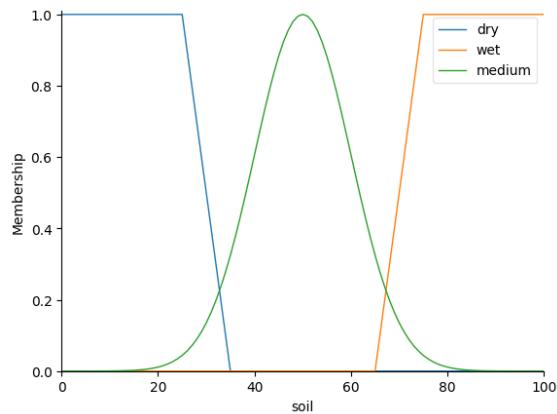
برای خاک خشک و مرطوب تابع مثلثی و برای حالت وسط از تابع گوسی استفاده کردیم.

وضعیت هوا (sunny, cloudy and rainy):

این ورودی‌ها را باینری در نظر گرفتیم و برای تعریف هر کدام از یک تابع مثلثی که به دو حالت True و False تقسیم می‌شوند استفاده کردیم.

متغیر خروجی، مقدار آبیاری (irrigation):

به چهار سطح زبانی تقسیم کردیم و با استفاده از توابع مثلثاتی تعریف شدند.



۴-۳ - قواعد منطق فازی زیر را پیاده سازی کنید:

- اگر خاک خشک است و هوا آفتابی است، آبیاری زیاد باشد.
- اگر خاک خشک و هوا ابری است، آب دهی متوسط باشد.
- اگر خاک خشک و هوا بارانی است، آب دهی کم.
- اگر خاک متوسط و هوا آفتابی است، مقدار آب متوسط.
- اگر خاک متوسط و هوا ابری است، مقدار آب کم.

- اگر خاک متوسط و هوا بارانی است نیازی به آب نیست.
- اگر خاک مرطوب و هوا آفتابی است، مقدار آب کم.
- اگر خاک مرطوب و هوا ابری است نیازی به آب نیست.
- اگر خاک مرطوب و هوا بارانی است نیازی به آب نیست.

```
rules = [
    ctrl.Rule(soil['dry'] & sunny['True'], irrig['High']),
    ctrl.Rule(soil['dry'] & cloudy['True'], irrig['Medium']),
    ctrl.Rule(soil['dry'] & rainy['True'], irrig['Low']),
    ctrl.Rule(soil['medium'] & sunny['True'], irrig['Medium']),
    ctrl.Rule(soil['medium'] & cloudy['True'], irrig['Low']),
    ctrl.Rule(soil['medium'] & rainy['True'], irrig['None']),
    ctrl.Rule(soil['wet'] & sunny['True'], irrig['Low']),
    ctrl.Rule(soil['wet'] & cloudy['True'], irrig['None']),
    ctrl.Rule(soil['wet'] & rainy['True'], irrig['None']),
]
```

۴-۲- با استفاده از روش مرکز ثقل (Centroid)، خروجی فازی را به یک مقدار واضح (Crisp) برای میزان آب تبدیل کنید.

ابتدا یک سیستم کنترل فازی رو براساس قواعد تعریف شده میسازیم و یک شبیه ساز از آن را اجرا میکنیم.

```
system = ctrl.ControlSystem(rules)
sim    = ctrl.ControlSystemSimulation(system)

def compute_irrigation(soil_val, weather_label):
    # Clear out any previous inputs/state
    sim.reset()
    # Set the soil moisture
    sim.input['soil'] = soil_val
    # One-hot encode weather
    sim.input['sunny'] = 1 if weather_label == 'Sunny' else 0
    sim.input['cloudy'] = 1 if weather_label == 'Cloudy' else 0
    sim.input['rainy'] = 1 if weather_label == 'Rainy' else 0
    # Run the fuzzy inference + defuzzification
    sim.compute()
    # Return the crisp irrigation amount plus the sim (for plotting)
    return sim.output['irrigation'], sim
```

تعریف تابعی برای اجرای سیستم و گرفتن خروجی:

ابتدا خروجی به صورت فازی محاسبه می‌شود و سپس با روش مرکز ثقل (که به صورت خودکار در تابع آماده است) به یک عدد واضح تبدیل می‌شود.

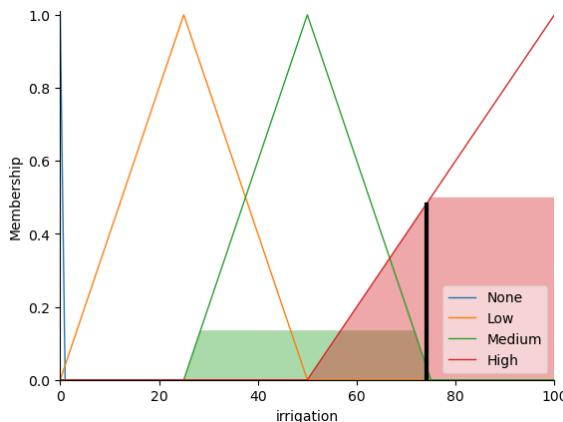
```
soil_val      = 30
weather_val   = 'Sunny'
crisp, sim_state = compute_irrigation(soil_val, weather_val)
print(f"Crisp irrigation for {soil_val}% & {weather_val} → {crisp:.2f}")
```

plot aggregated output with centroid

```
irrig.view(sim=sim_state)
plt.show()
```

Out:

Crisp irrigation for 30% & Sunny → 74.02



۲-۵- حداقل با سه روش دیگر یا بیشتر خروجی فازی را به یک مقدار واضح برای میزان آب تبدیل کنید سپس نتایج را برای هر یک مقایسه کنید.

```
def compare_and_plot(soil_val, weather_label):
    flags = {
        'sunny': 1 if weather_label=='Sunny' else 0,
        'cloudy': 1 if weather_label=='Cloudy' else 0,
        'rainy': 1 if weather_label=='Rainy' else 0,
    }
    methods = ['centroid', 'bisector', 'mom', 'som', 'lom']
    results = {}
    for m in methods:
        irrig.defuzzify_method = m
        sim.reset()
        sim.input['soil']    = soil_val
        sim.input['sunny']   = flags['sunny']
        sim.input['cloudy']  = flags['cloudy']
        sim.input['rainy']   = flags['rainy']
```

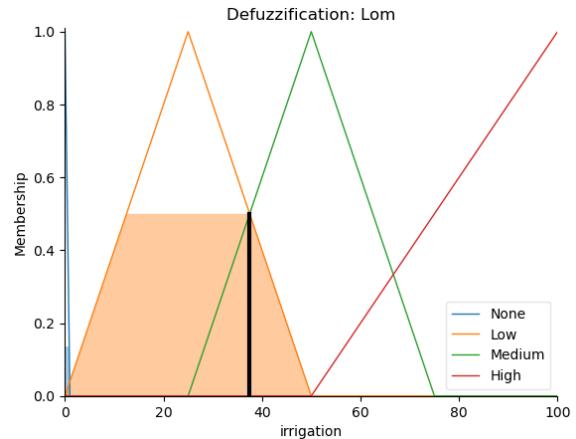
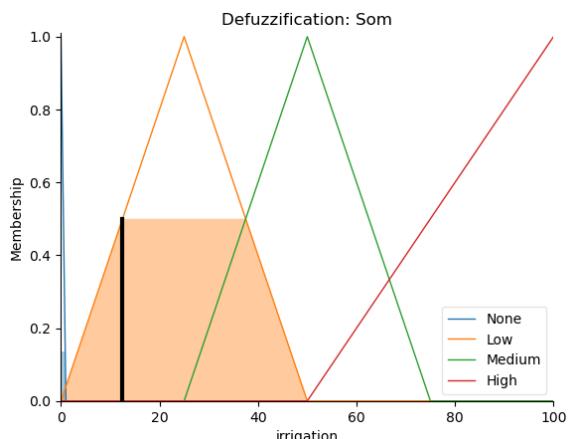
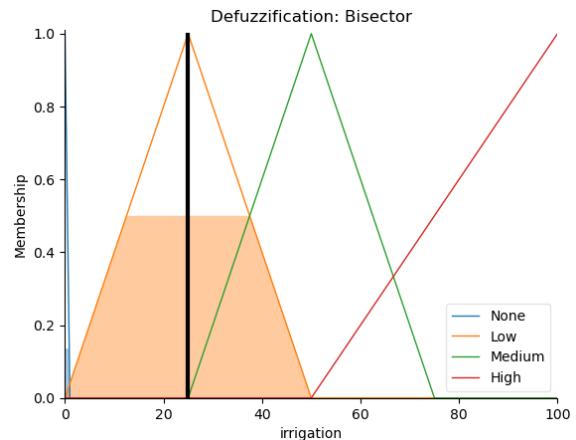
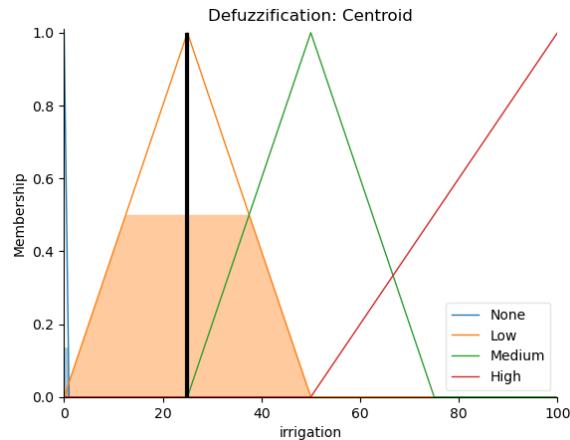
```

        sim.compute()
        results[m] = sim.output['irrigation']

    irrig.view(sim=sim)
    plt.title(f"Defuzzification: {m.capitalize()}")
    plt.show()

    return results
res = compare_and_plot(soil_val=30, weather_label='Rainy')
for method, value in res.items():
    print(f"{method:8s} → {value:.2f}")
out:
centroid → 24.86
bisector → 24.89
mom → 25.00
som → 12.50
lom → 37.50

```



۴-۶- کنترل کننده فازی طراحی شده را در یک بازه زمانی مثلاً ۱۰ روز با شرایط متغیر آب و هوایی مثل‌اً ابتدا آفتایی سپس ابری و در نهایت بارانی شبیه سازی کنید. در هر گام زمانی رطوبت خاک را بر اساس تصمیم کنترل کننده به روز کنید.

```

import random

# 5) RUN A 10-DAY SIMULATION
def simulate_over_time(initial_soil=40.0):
    beta = 0.6 # irrigation → soil moisture loss
    weather_effects = {
        'Sunny': -0.05,
        'Cloudy': -0.02,
        'Rainy': +0.05
    }

    days = np.arange(1, 11)
    weather_seq = ['Sunny']*6 + ['Cloudy']*2 + ['Rainy']*2
    random.shuffle(weather_seq)

    soil_history = [initial_soil]
    irrigation_history = []

    for day, weather in zip(days, weather_seq):
        current_soil = soil_history[-1]
        irrig_amt, _ = compute_irrigation(current_soil, weather)
        irrigation_history.append(irrig_amt)

        next_soil = current_soil + weather_effects[weather]*current_soil + beta*irrig_amt
        next_soil = np.clip(next_soil, 0, 100)

        soil_history.append(next_soil)

    # Run 10-day simulation
    simulate_over_time(initial_soil=40.0)

```

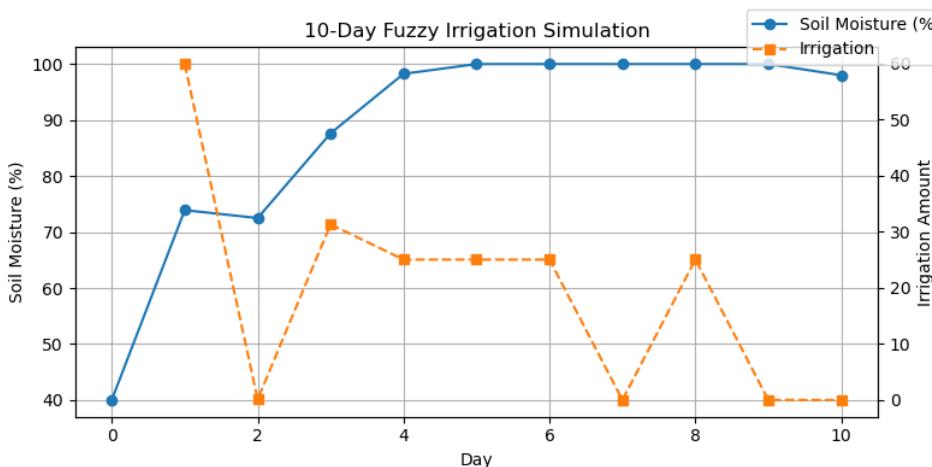
در ابتدای تابع ضرایب ثابت و پارامترها را تعریف کردیم سپس یک لیست از وضعیت آب و هوایی که در هر فراخوانی تابع شافل میشود قرار دادیم.

سپس با استفاده از فرمول داده شده رطوبت جدید خاک را در هر ایتریشن حساب میکنیم.

۴-۷ - موارد زیر را ترسیم و بررسی کنید.

- تغییرات رطوبت خاک در طول زمان

- میزان آبدهی تعیین شده توسط کنترل کننده فازی در طول زمان



همانطور که در نمودار مشاهده میشود، وقتی رطوبت پایین است مقدار آبیاری به مقدار زیادی افزایش پیدا میکند ولی در ادامه با اینکه رطوبت خاک به حد زیادی میرسد همچنان آبیاری انجام میشود، این مشکل میتواند بخاطر پارامترها و یا کم بودن تنوع در فازی سازی و قوانین فازی باشد. (که در پیاده سازی بعدی اصلاح شده است).

۴-۸ - نتایج را تحلیل و بررسی کنید کننده فازی طراحی شده تا چه حد در حفظ سطح بهینه رطوبت خاک موفق است؟

همان طور که در نمودار مشاهده میشود، وقتی رطوبت پایین است مقدار آبیاری به مقدار زیادی افزایش پیدا میکند ولی در ادامه با اینکه رطوبت خاک به حد زیادی میرسد همچنان آبیاری انجام میشود، این مشکل میتواند بخاطر پارامترها و یا کم بودن تنوع در فازی سازی و قوانین فازی باشد. (که در پیاده سازی بعدی اصلاح شده است).

۴-۹ - قواعد فازی اولیه را بررسی کنید و قواعد جدیدی را برای افزایش دقت و جزئیات کنترل اضافه کنید به عنوان مثال برای سطوح رطوبت و شرایط آب و هوایی که در قواعد اولیه در نظر گرفته نشده بودند قواعد جدیدی پیشنهاد دهید.

```
# Antecedent: soil moisture (0-100)
soil = ctrl.Antecedent(np.arange(0, 101, 1), 'soil')

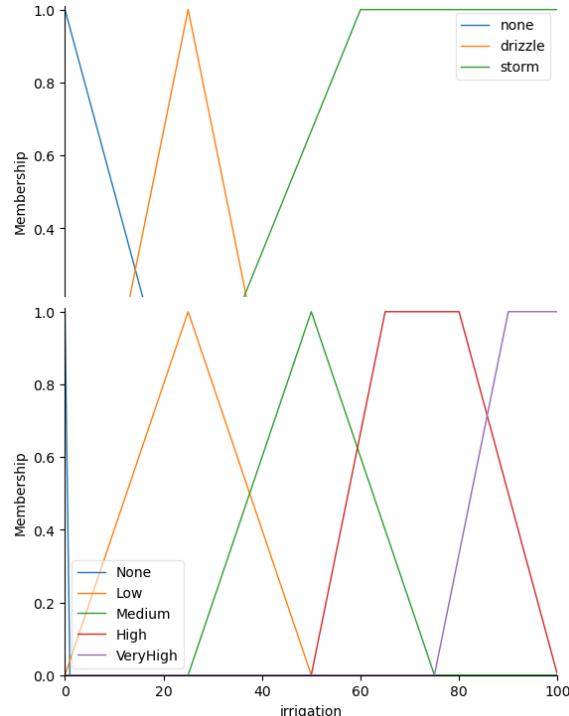
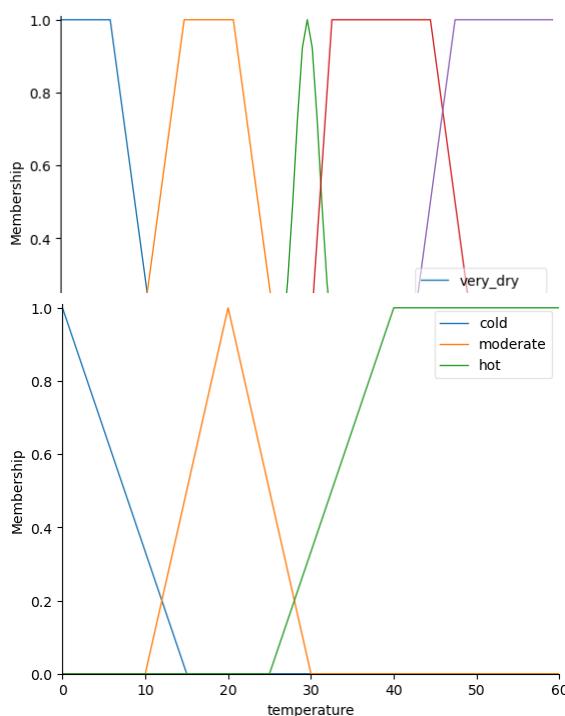
soil['very_dry']      = fuzz.trapmf(soil.universe, [0, 0, 10, 20])
soil['slightly_dry']  = fuzz.trapmf(soil.universe, [15, 25, 35, 45])
soil['medium']         = fuzz.gaussmf(soil.universe, 50, 2.5)
soil['slightly_wet']   = fuzz.trapmf(soil.universe, [50, 55, 75, 85])
soil['very_wet']        = fuzz.trapmf(soil.universe, [70, 80, 100, 100])
```

```

# Antecedent: rain intensity (0-100)
rain = ctrl.Antecedent(np.arange(0, 101, 1), 'rain_intensity')
rain['none']      = fuzz.trimf(rain.universe, [0, 0, 20])
rain['drizzle']   = fuzz.trimf(rain.universe, [10, 25, 40])
rain['storm']     = fuzz.trapmf(rain.universe, [30, 60, 100, 100])
# Antecedent: temperature (°C, 0-60)
temp = ctrl.Antecedent(np.arange(0, 61, 1), 'temperature')
temp['cold']      = fuzz.trimf(temp.universe, [0, 0, 15])
temp['moderate']  = fuzz.trimf(temp.universe, [10, 20, 30])
temp['hot']        = fuzz.trapmf(temp.universe, [25, 40, 60, 60])
# Antecedent: crop water requirement (0-2)
water_req = ctrl.Antecedent(np.arange(0, 3, 1), 'water_req')
water_req['low']   = fuzz.trimf(water_req.universe, [0, 0, 1])
water_req['medium'] = fuzz.trimf(water_req.universe, [0, 1, 2])
water_req['high']  = fuzz.trimf(water_req.universe, [1, 2, 2])
# Consequent: irrigation amount (0-100)
irrig = ctrl.Consequent(np.arange(0, 101, 1), 'irrigation')
irrig['None']      = fuzz.trimf(irrig.universe, [0, 0, 0])
irrig['Low']        = fuzz.trimf(irrig.universe, [0, 25, 50])
irrig['Medium']    = fuzz.trimf(irrig.universe, [25, 50, 75])
irrig['High']       = fuzz.trapmf(irrig.universe, [50, 65, 80, 100])
irrig['VeryHigh']  = fuzz.trapmf(irrig.universe, [75, 90, 100, 100])

```

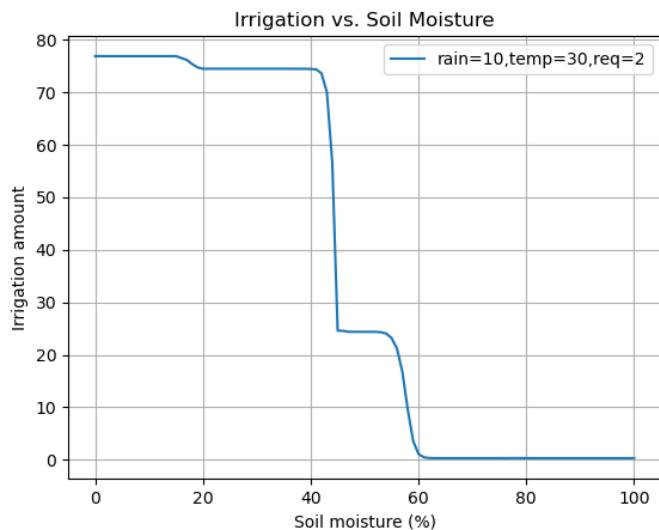
در فاز جدید پیاده سازی برای منعطف کردن سیستم فازی تعداد متغیرها (دما، سطح نیاز به آبیاری برای محصول و شدت باران) و تعداد سطوح متغیرهای قبلی را افزایش دادیم. نمایش توابع استفاده شده برای مقادیر عضویت:



قوانين جديد:

```
rules = [
    # Very dry + no rain + hot + high-demand → emergency watering
    ctrl.Rule(soil['very_dry'] & rain['none'] & temp['hot'] & water_req['high'],
              irrig['VeryHigh']),
    ctrl.Rule(soil['very_dry'] & rain['none'] & water_req['high'],
              irrig['VeryHigh']),
    ctrl.Rule(soil['very_dry'] & rain['none'],
              irrig['High']),
    # Very dry + light drizzle → still substantial watering
    ctrl.Rule(soil['very_dry'] & rain['drizzle'] & water_req['medium'],
              irrig['Medium']),
    # Very dry + heavy storm → minimal top-up
    ctrl.Rule(soil['very_dry'] & rain['storm'],
              irrig['Low']),
    # — Mild Dryness —
    # Slightly dry + no rain → high watering
    ctrl.Rule(soil['slightly_dry'] & rain['none'],
              irrig['High']),
    # Slightly dry + drizzle → low watering
    ctrl.Rule(soil['slightly_dry'] & rain['drizzle'],
              irrig['Low']),
    # Slightly dry + storm → skip watering
    ctrl.Rule(soil['slightly_dry'] & rain['storm'],
              irrig['None']),
    # — Near-Optimal & Wet —
    # Medium soil + no rain → medium watering
    ctrl.Rule(soil['medium'] & rain['none'],
              irrig['None']),
    ctrl.Rule(soil['medium'] & rain['none'] & temp['hot'],
              irrig['Low']),
    # Medium soil + any rain → low or none
    ctrl.Rule(soil['medium'] & rain['drizzle'],
              irrig['Low']),
    ctrl.Rule(soil['medium'] & rain['storm'],
              irrig['None']),
    # Slightly wet or very wet → no watering
    ctrl.Rule(soil['slightly_wet'], irrig['None']),
    ctrl.Rule(soil['very_wet'], irrig['None']),
]
```

۱۰-۲- پس از تغییر قواعد دوباره شبیه سازی را اجرا کنید و نتایج را مقایسه کنید آیا افزودن قواعد بیشتر و دقیق تر باعث بهبود در کنترل و حفظ رطوبت بهینه خاک می شود؟



این بار سیستم را طوری طراحی کردیم که وقتی رطوبت به بالای ۶۰ واحد برسد آبیاری متوقف شود.

شرایط محیط در این نمودار:

۱۰ واحد باران (none) •

۳۰ واحد دما (گرم) •

۲ واحد نیاز به آبیاری (محصول نیاز

به آبیاری زیادی دارد)

شبیه سازی ۱۰ روزه:

```
weather_effects = {
    'Sunny': -0.05,
    'Cloudy': -0.02,
    'Rainy': +0.05
}
β = 0.6 # irrigation effectiveness
temp_w = 0.25
rain_w = 0.5

for day, (r, t) in enumerate(zip(rain_seq, temp_seq), start=1):
    current = soil_hist[-1]
    irr, _ = compute_irrigation(current, r, t, req)

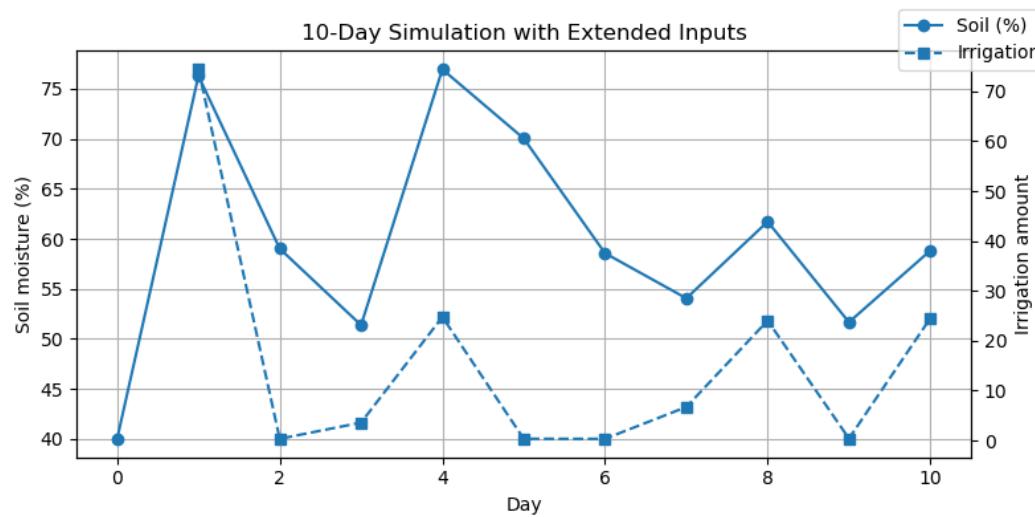
    next_soil = current + (r)*(rain_w) -(t/60)*(temp_w)*current + β*irr

rain_seq = [0, 0, 5, 30, 5, 0, 0, 0, 0, 0]
temp_seq = [50, 55, 50, 20, 30, 40, 35, 30, 40, 35]
simulate_10_days(initial_soil=40.0, rain_seq=rain_seq, temp_seq=temp_seq, req=1)
```

تغییر فرمول بروزرسانی رطوبت:

رطوبت جدید = رطوبت فعلی + مقدار واحد بارش * نسبت دما * وزن بارش - تاثیر آبیاری * مقدار آبیاری

با تغییر فرمول و دادن اطلاعات بیشتر تونستیم سیستم فازی رو به رطوبت بهینه نزدیکتر کنیم:



۳- تمرین سوم: طراحی سیستم منطق فازی برای توصیه تمرینات ورزشی

۱-۳- تعریف متغیرهای ورودی فازی:

شما باید حداقل پنج متغیر ورودی را برای سیستم خود تعریف کنید:

- سطح آمادکی جسمانی: (مبتدی، متوسط، پیشرفته)

- سطح انرژی: (کم، متوسط، زیاد)

- هدف ورزشی: (کاهش وزن، افزایش عضله، تناسب عمومی)

- سن: (جوان، میانه‌سال، سالمند)

- وزن: (کمبود وزن، نرمال، اضافه وزن)

۲-۳- تعریف متغیرهای خروجی فازی:

- شدت تمرین: (کم، متوسط، زیاد)

- مدت زمان تمرین: (کوتاه، متوسط، طولانی)

```
physical_readiness = ctrl.Antecedent(np.arange(0, 11, 1), "physical_readiness")
energy_level = ctrl.Antecedent(np.arange(0, 11, 1), "energy_level")
fitness_goal = ctrl.Antecedent(np.arange(0, 13, 1), "fitness_goal")
age = ctrl.Antecedent(np.arange(18, 101, 1), "age")
weight = ctrl.Antecedent(np.arange(30, 151, 1), "weight")
motivation = ctrl.Antecedent(np.arange(0, 11, 1), 'motivation')
exercise_intensity = ctrl.Consequent(np.arange(0, 11, 1), "exercise_intensity")
exercise_duration = ctrl.Consequent(np.arange(0, 61, 1), "exercise_duration")
```

آمادگی جسمانی و انرژی: مقیاس ۰-۱۰ امکان سنجش دقیق وضعیت کاربر را می‌دهد. به عنوان مثال، عدد ۷ در انرژی نشان‌دهنده انرژی نسبتاً بالا است.

هدف ورزشی: برای پوشش طیف متنوع اهداف، از کدگذاری گسسته ۰-۱۲ استفاده شده است که هر عدد نمایانگر یک هدف مشخص مثل کاهش وزن یا انعطاف‌پذیری است.

سن و وزن: محدوده‌های بزرگ‌سال معمول (۱۸-۱۰۰ سال و ۳۰-۱۵۰ کیلوگرم) انتخاب شده تا برای اکثر کاربران کاربردی باشد.

انگیزه: فاکتوری روان‌شناسی که تأثیر زیادی بر تعهد به تمرین دارد.

متغیرهای خروجی:

شدت تمرین عددی بین ۰ (بدون فشار) تا ۱۰ (حداکثر فشار) که اجازه می‌دهد توصیه‌ای متناسب با توان کاربر ارائه شود. مدت تمرین بر حسب دقیقه که از ۰ تا ۶۰ دقیقه متغیر است، و دسته‌بندی سه‌سطحی آن کمک می‌کند کاربر به سادگی زمان مناسب جلسه را انتخاب کند.

۳-۳- تعریف توابع عضویت فازی برای هر متغیر:

برای هر یک از متغیرهای ورودی و خروجی باید توابع عضویت مناسب تعریف کنید.

```
physical_readiness["beginner"] = fuzz.trimf(physical_readiness.universe,
[0, 0, 4])
physical_readiness["intermediate"] =
fuzz.trimf(physical_readiness.universe, [3, 5, 7])
physical_readiness["advanced"] = fuzz.trimf(physical_readiness.universe,
[6, 10, 10])

energy_level["low"] = fuzz.trimf(energy_level.universe, [0, 0, 4])
energy_level["mid"] = fuzz.trimf(energy_level.universe, [3, 5, 7])
energy_level["high"] = fuzz.trimf(energy_level.universe, [6, 10, 10])

fitness_goal["maintenance"] = fuzz.trimf(fitness_goal.universe, [0, 2, 4])
fitness_goal["weight_loss"] = fuzz.trimf(fitness_goal.universe, [3, 6, 9])
fitness_goal["muscle_gain"] = fuzz.trimf(fitness_goal.universe, [8, 11,
12])

age["young"] = fuzz.trimf(age.universe, [18, 25, 30])
age["middle_aged"] = fuzz.trimf(age.universe, [30, 45, 60])
age["old"] = fuzz.trimf(age.universe, [55, 70, 80])

weight["underweight"] = fuzz.trimf(weight.universe, [30, 40, 55])
weight["normal"] = fuzz.trimf(weight.universe, [50, 65, 80])
weight["overweight"] = fuzz.trimf(weight.universe, [75, 90, 110])
weight["obese"] = fuzz.trimf(weight.universe, [100, 120, 150])

exercise_intensity["low"] = fuzz.trapmf(exercise_intensity.universe, [0,
0, 2, 4])
exercise_intensity["mid"] = fuzz.trapmf(exercise_intensity.universe, [3,
4, 6, 7])
exercise_intensity["high"] = fuzz.trapmf(exercise_intensity.universe, [6,
8, 10, 10])
```

```

exercise_duration["short"] = fuzz.trapmf(exercise_duration.universe, [0,
0, 10, 20])
exercise_duration["mid"] = fuzz.trapmf(exercise_duration.universe, [15,
25, 35, 45])
exercise_duration["long"] = fuzz.trapmf(exercise_duration.universe, [40,
50, 60, 60])

motivation['low'] = fuzz.trimf(motivation.universe, [0, 0, 4])
motivation['medium'] = fuzz.trimf(motivation.universe, [3, 5, 7])
motivation['high'] = fuzz.trimf(motivation.universe, [6, 10, 10])

```

- برای متغیرهای ورودی از توابع مثلثی (trimf) استفاده شده تا هر عبارت زبانی (مثل beginner یا intermediate) باشد. پارامترها نقاط پایه و رأس مثلث را تعیین می‌کنند.
- برای متغیرهای خروجی از توابع ذوزنقه‌ای (trapmf) استفاده کردیم تا سطوح پایدار در میانه دامنه داشته باشیم و تغییرات تدریجی در لبه‌ها اعمال شود.
- انتخاب بازه‌ها و پارامترها بر اساس منطق، همپوشانی جزئی بین سطوح مجاور برای انتقال نرم و جلوگیری از جهش ناگهانی در دی‌فازی‌سازی است.

۴-۳- تعریف قواعد فازی:

```

from skfuzzy import control as ctrl

rules = [
    # Physical Readiness vs. Energy Level
    ctrl.Rule(physical_readiness['beginner'] & energy_level['low'],
              consequent=[exercise_duration['short'],
exercise_intensity['low']]],
    ctrl.Rule(physical_readiness['beginner'] & energy_level['mid'],
              consequent=[exercise_duration['short'],
exercise_intensity['low']]],
    ctrl.Rule(physical_readiness['beginner'] & energy_level['high'],
              consequent=[exercise_duration['mid'],
exercise_intensity['mid']]],
    ctrl.Rule(physical_readiness['intermediate'] & energy_level['low'],
              consequent=[exercise_duration['mid'],
exercise_intensity['low']]],
    ctrl.Rule(physical_readiness['intermediate'] & energy_level['mid'],
              consequent=[exercise_duration['mid'],
exercise_intensity['mid']]]
]

```

```

        consequent=[exercise_duration['mid'],
exercise_intensity['mid']]),
ctrl.Rule(physical_readiness['intermediate'] & energy_level['high'],
          consequent=[exercise_duration['long'],
exercise_intensity['high']]),

ctrl.Rule(physical_readiness['advanced'] & energy_level['low'],
          consequent=[exercise_duration['mid'],
exercise_intensity['mid']]),
ctrl.Rule(physical_readiness['advanced'] & energy_level['mid'],
          consequent=[exercise_duration['long'],
exercise_intensity['mid']]),
ctrl.Rule(physical_readiness['advanced'] & energy_level['high'],
          consequent=[exercise_duration['long'],
exercise_intensity['high']]),

# Fitness Goal vs. Age and Weight
ctrl.Rule(fitness_goal['weight_loss'] & age['young'] &
weight['underweight'],
          consequent=[exercise_intensity['low'],
exercise_duration['mid']]),
ctrl.Rule(fitness_goal['weight_loss'] & age['young'] &
weight['normal'],
          consequent=[exercise_intensity['mid'],
exercise_duration['long']]),
ctrl.Rule(fitness_goal['weight_loss'] & age['young'] &
weight['overweight'],
          consequent=[exercise_intensity['high'],
exercise_duration['long']]),
ctrl.Rule(fitness_goal['weight_loss'] & age['middle_aged'] &
weight['normal'],
          consequent=[exercise_intensity['mid'],
exercise_duration['mid']]),
ctrl.Rule(fitness_goal['weight_loss'] & age['middle_aged'] &
weight['overweight'],
          consequent=[exercise_intensity['high'],
exercise_duration['mid']]),
ctrl.Rule(fitness_goal['weight_loss'] & age['old'] &
weight['overweight'],
          consequent=[exercise_intensity['mid'],
exercise_duration['short']]),

ctrl.Rule(fitness_goal['muscle_gain'] & age['young'] &
weight['normal'],

```

```

        consequent=[exercise_intensity['mid'],
exercise_duration['mid'])],
ctrl.Rule(fitness_goal['muscle_gain'] & age['young'] &
weight['overweight'],
        consequent=[exercise_intensity['high'],
exercise_duration['long'])],
ctrl.Rule(fitness_goal['muscle_gain'] & age['middle_aged'] &
weight['normal'],
        consequent=[exercise_intensity['mid'],
exercise_duration['mid'])],
ctrl.Rule(fitness_goal['muscle_gain'] & age['middle_aged'] &
weight['overweight'],
        consequent=[exercise_intensity['high'],
exercise_duration['long'])],
ctrl.Rule(fitness_goal['muscle_gain'] & age['old'] & weight['normal'],
        consequent=[exercise_intensity['mid'],
exercise_duration['short'])],

ctrl.Rule(fitness_goal['maintenance'] & age['young'] &
weight['normal'],
        consequent=[exercise_intensity['mid'],
exercise_duration['mid'])],
ctrl.Rule(fitness_goal['maintenance'] & age['middle_aged'] &
weight['normal'],
        consequent=[exercise_intensity['mid'],
exercise_duration['long'])],
ctrl.Rule(fitness_goal['maintenance'] & age['old'] & weight['normal'],
        consequent=[exercise_intensity['low'],
exercise_duration['mid'])],
ctrl.Rule(fitness_goal['maintenance'] & age['old'] &
weight['overweight'],
        consequent=[exercise_intensity['mid'],
exercise_duration['short'])],


# Weight vs. Physical Readiness
ctrl.Rule(weight['underweight'] & physical_readiness['beginner'],
        consequent=[exercise_intensity['low'],
exercise_duration['mid'])],
ctrl.Rule(weight['underweight'] & physical_readiness['intermediate'],
        consequent=[exercise_intensity['low'],
exercise_duration['mid'])],
ctrl.Rule(weight['underweight'] & physical_readiness['advanced'],
        consequent=[exercise_intensity['mid'],
exercise_duration['mid'])],

```

```

ctrl.Rule(weight['normal'] & physical_readiness['beginner'],
          consequent=[exercise_intensity['low'],
exercise_duration['mid']]),
ctrl.Rule(weight['normal'] & physical_readiness['intermediate'],
          consequent=[exercise_intensity['mid'],
exercise_duration['mid']]),
ctrl.Rule(weight['normal'] & physical_readiness['advanced'],
          consequent=[exercise_intensity['mid'],
exercise_duration['long']]),

ctrl.Rule(weight['overweight'] & physical_readiness['beginner'],
          consequent=[exercise_intensity['low'],
exercise_duration['short']]),
ctrl.Rule(weight['overweight'] & physical_readiness['intermediate'],
          consequent=[exercise_intensity['mid'],
exercise_duration['mid']]),
ctrl.Rule(weight['overweight'] & physical_readiness['advanced'],
          consequent=[exercise_intensity['high'],
exercise_duration['long']]),

ctrl.Rule(weight['obese'] & physical_readiness['beginner'],
          consequent=[exercise_intensity['low'],
exercise_duration['short']]),
ctrl.Rule(weight['obese'] & physical_readiness['intermediate'],
          consequent=[exercise_intensity['mid'],
exercise_duration['short']]),
ctrl.Rule(weight['obese'] & physical_readiness['advanced'],
          consequent=[exercise_intensity['high'],
exercise_duration['mid']]),

# Age-specific rules
ctrl.Rule(age['young'] & fitness_goal['weight_loss'],
          consequent=[exercise_intensity['mid'],
exercise_duration['long']]),
ctrl.Rule(age['middle_aged'] & fitness_goal['weight_loss'],
          consequent=[exercise_intensity['high'],
exercise_duration['mid']]),
ctrl.Rule(age['old'] & fitness_goal['weight_loss'],
          consequent=[exercise_intensity['mid'],
exercise_duration['short']]),

# motivation
ctrl.Rule(motivation['low'], consequent=[exercise_intensity['low'],
exercise_duration['short']]),

```

```

ctrl.Rule(motivation['high'], consequent=[exercise_intensity['high'],
exercise_duration['long']]),

]

fis = ctrl.ControlSystem(rules)
sim = ctrl.ControlSystemSimulation(fis)

```

- قواعد بر اساس ترکیب ورودی‌های کلیدی تنظیم شده‌اند: آمادگی جسمانی و انرژی برای تنظیم اولیه، سپس هدف تناسب‌اندام در ترکیب با سن و وزن برای شخصی‌سازی بیشتر.
- وزن و آمادگی جسمانی نیز مستقل برای اصلاح شدت و مدت اضافه شده‌اند.
- قواعد انگیزشی نهایی میزان درگیری ذهنی را در نظر می‌گیرند تا اگر انگیزه کم است زمان و شدت کاهش یابد و بالعکس.
- در زیر چند قانون مهم با مثال آورده شده است.

قاعده ترکیب اولیه (Beginner & Low Energy)

: physical_readiness['beginner'] & energy_level['low'] شرط

exercise_duration = short و exercise_intensity = low خروجی

مثال: فردی تازه‌کار پس از یک روز کاری خسته، بهترین شروع برای او یک جلسه ۱۰ دقیقه‌ای با شدت کم است تا بدن به تدریج گرم شود.

قاعده شخصی‌سازی وزن و هدف (Weight Loss, Old & Overweight)

& fitness_goal['weight_loss'] & age['old'] & weight['overweight'] شرط

exercise_duration = short و exercise_intensity = mid خروجی

مثال: برای یک فرد ۶۵ ساله با اضافه‌وزن که هدف کاهش وزن دارد، توصیه می‌شود ۲۰ دقیقه پیاده‌روی با شدت متوسط انجام دهد تا مفاصل تحت فشار زیاد قرار نگیرند.

قاعده اصلاح مستقل (Obese & Advanced)

: weight['obese'] & physical_readiness['advanced'] شرط

`exercise_duration = mid` و `exercise_intensity = high` خروجی

مثال: اگر فردی با وزن بالا اما سطح آمادگی بدنی بالا باشد، یک جلسه ۳۰ دقیقه‌ای با شدت بالا (مانند دوچرخه‌سواری سریع) مناسب است تا کالری‌سوزی افزایش یابد.

قاعده انگیزشی (Motivation High)

`motivation['high']` شرط

`exercise_duration = long` و `exercise_intensity = high` خروجی

مثال: شخصی که انگیزه زیادی دارد، می‌تواند زمان طولانی با شدت بالا ورزش کند.

۵-۳- استفاده از استنتاج فازی:

پس از تعریف متغیرها و قواعد فازی، سیستم باید با استفاده از استنتاج فازی تصمیمات مناسب را برای شدت و مدت زمان تمرین ارائه دهد.

```
def recommend_exercise(workout_style, fitness_goal_value,
physical_readiness_value, energy_level_value, age_value, weight_value,
motivation_value):
    fitness_goal_crisp_map = {"Maintenance": 0, "Weight Loss": 6, "Muscle
Gain": 12}

    sim.input['physical_readiness'] = physical_readiness_value
    sim.input['energy_level'] = energy_level_value
    sim.input['fitness_goal'] = fitness_goal_crisp_map[fitness_goal_value]
    sim.input['age'] = age_value
    sim.input['weight'] = weight_value
    sim.input['motivation'] = motivation_value

    sim.compute()

    intensity = sim.output['exercise_intensity']
    duration = sim.output['exercise_duration']

    # multiplexers
    if workout_style == 'Relaxed':
        sim.output['exercise_intensity'] *= 0.8
        sim.output['exercise_duration'] *= 0.8

    elif workout_style == 'Intense':
```

```

sim.output['exercise_intensity'] *= 1.2
sim.output['exercise_duration'] *= 1.2

intensity = min(sim.output['exercise_intensity'], 10)
duration = min(sim.output['exercise_duration'], 60)

exercise_intensity.view(sim)
exercise_duration.view(sim)

```

در پیاده‌سازی سیستم، ابتدا ورودی‌های عددی (مانند آمادگی جسمانی، سطح انرژی، هدف ورزشی، سن، وزن و انگیزه) به موتور فازی داده می‌شوند. در مرحله‌ی فازی‌سازی، هر کدام از این مقادیر با استفاده از توابع عضویت مثلثی یا ذوزنقه‌ای به درجات «کم»، «متوسط» و «زیاد» تبدیل می‌گردد. سپس برای هر یک از قواعد تعریف شده، میزان تحقق آنها به کمک عملگر «AND» (عملگر Min) محاسبه می‌شود و خروجی فازی هر قاعده مشخص می‌شود. در ادامه همه‌ی این خروجی‌های فازی با هم تلفیق می‌شوند (روش Max) تا یک توزیع فازی یکپارچه برای شدت تمرین و مدت‌زمان آن شکل بگیرد. این توزیع نهایی با استفاده از روش مرکز ثقل (centroid) دی‌فازی‌سازی شده و به دو عدد قطعی «شدت تمرین» و «مدت تمرین» تبدیل می‌شود. در پایان، متناسب با سبک تمرین (آرام، متعادل یا شدید) هر یک از مقادیر ۲۰٪ افزایش یا کاهش می‌یابد و نهایتاً در بازه‌های مجاز [۰ تا ۶۰] برای شدت و [۰ تا ۱۰] برای مدت‌زمان محدود می‌شود تا خروجی نهایی به کاربر نمایش داده شود.

۳-۶-۳- تست و اعتبارسنجی سیستم:

پس از پیاده‌سازی، سیستم خود را با چندین ورودی مختلف آزمایش کنید و نتایج آن را با انتظارات منطقی مقایسه کنید.

```

import ipywidgets as widgets
from ipywidgets import interact
interact(recommend_exercise,
         workout_style=widgets.ToggleButtons(
             options=['Relaxed', 'Balanced', 'Intense'],
             value='Balanced',
             description='Workout Style'
         ),
         fitness_goal_value=widgets.ToggleButtons(
             options=["Maintenance", "Weight Loss", "Muscle Gain"],
             value="Weight Loss",
             description="Fitness Goal"
         ),
         physical_readiness_value=widgets.IntSlider(min=0, max=10, step=1,
value=5, description="Readiness"),
         energy_level_value=widgets.IntSlider(min=0, max=10, step=1,
value=5, description="Energy Level"),

```

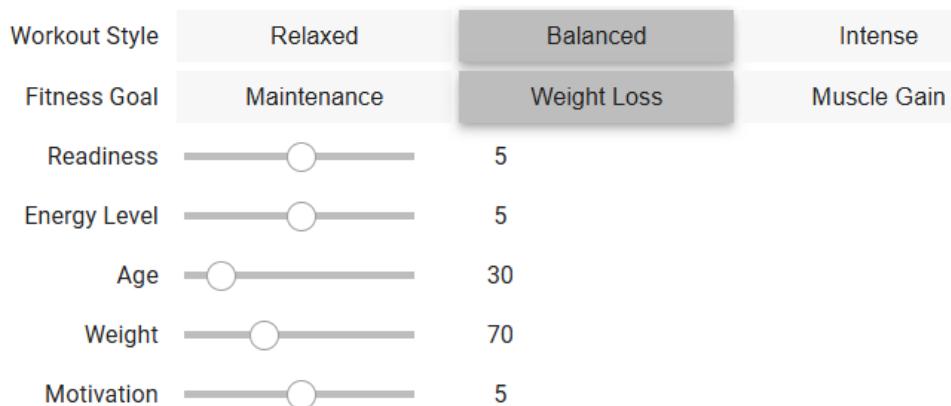
```

        age_value=widgets.IntSlider(min=18, max=100, step=1, value=30,
description="Age"),
        weight_value=widgets.IntSlider(min=30, max=150, step=1, value=70,
description="Weight"),
        motivation_value=widgets.IntSlider(min=0, max=10, step=1,
value=5, description="Motivation")
)

```

برای تست و ارزیابی، چند سناریوی متفاوت مثل «فرد جوان با انرژی بالا»، «فرد میانسال با انگیزه متوسط» و «فرد مسن با انرژی و انگیزه کم» تعریف و مقادیر ورودی آنها به سیستم داده شد. خروجی‌های حاصل (شدت و مدت تمرین) با انتظارات منطقی مقایسه شدند؛ برای مثال در حالت متعادل با هدف کاهش وزن، مقدار متوسطی از شدت و مدت تمرین به دست آمد که با شرایط ورودی همخوانی داشت، و در سناریوی شدید با انگیزه بالا، خروجی نیز افزایش منطقی نشان داد.

به ازای ورودی‌های زیر

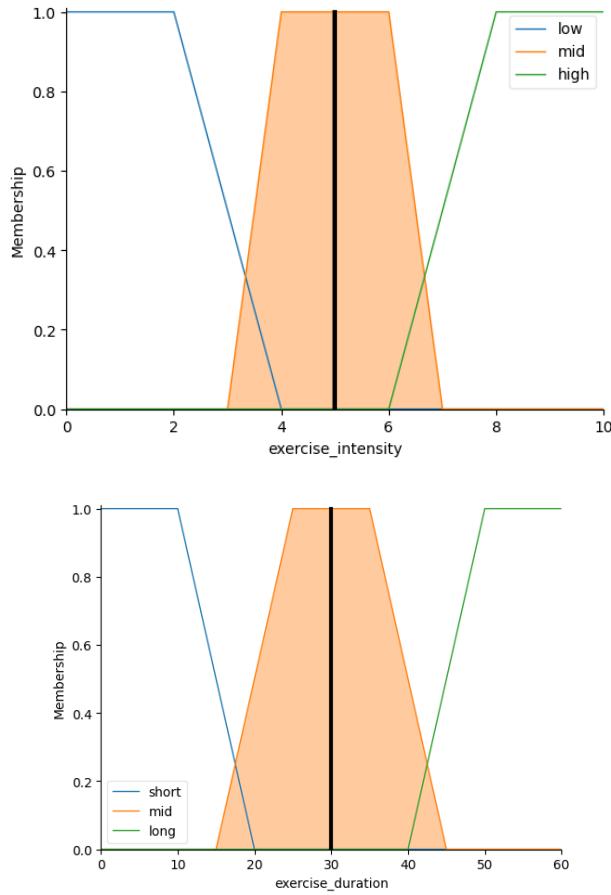


🏋️ Personalized Exercise Recommendation

خروجی و نمودارها به صورت زیر می باشد

📈 Recommended Intensity: 5.00 / 10

⌚ Recommended Duration: 30.00 minutes



۷-۳- نمایش گرافیکی:

از کتابخانه‌های موجود مانند **matplotlib** برای رسم توابع عضویت و همچنین نتایج استنتاج فازی استفاده کنید تا فرایند تصمیمگیری سیستم را به صورت گرافیکی نمایش دهید.

```
def plot_membership_funcs(variables):
    num_vars = len(variables)
    cols = 2
    rows = (num_vars + 1) // cols

    fig, axes = plt.subplots(rows, cols, figsize=(15, 3 * rows))
    axes = axes.flatten()

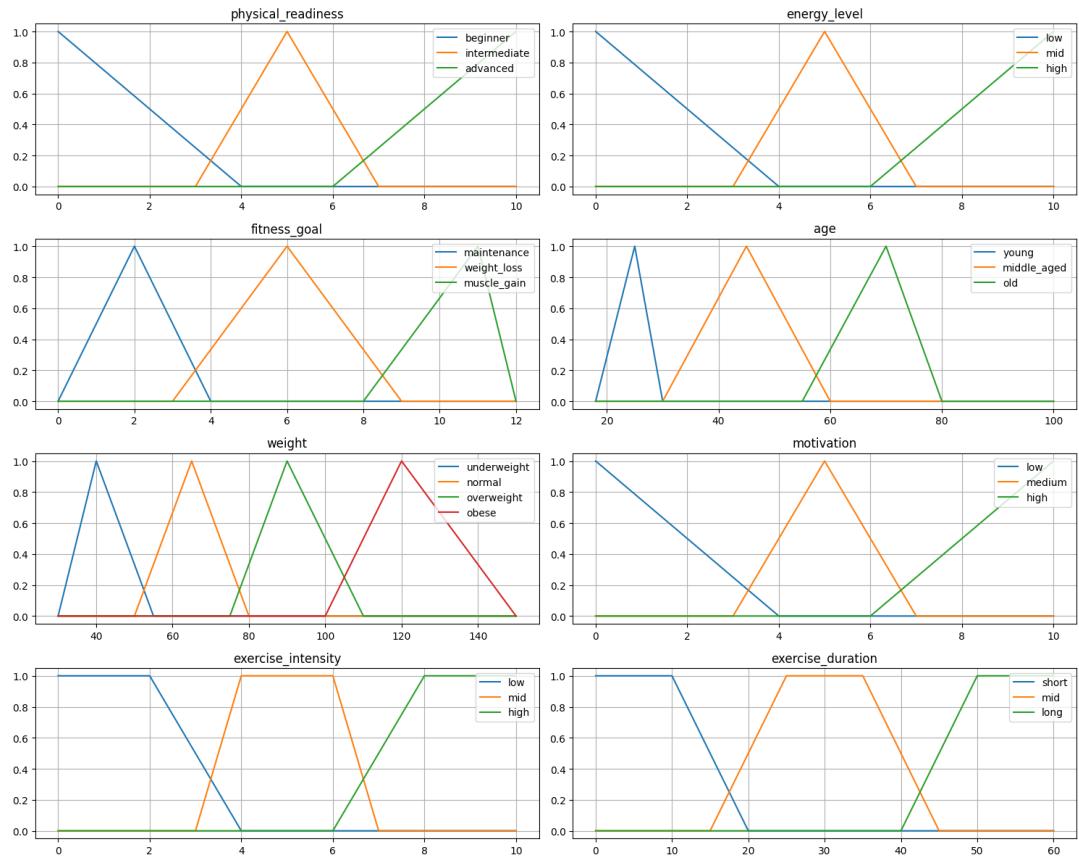
    for i, var in enumerate(variables):
        ax = axes[i]
        for term_name in var.terms:
            ax.plot(var.universe, var[term_name].mf, label=term_name)
        ax.set_title(f'{var.label}', fontsize=12)
        ax.set_ylim([-0.05, 1.05])
```

```
    ax.legend(loc='upper right')
    ax.grid(True)

    # hide unused subplots
    for j in range(i + 1, len(axes)):
        fig.delaxes(axes[j])

plt.tight_layout()
plt.show()

plot_membership_funcs([
    physical_readiness,
    energy_level,
    fitness_goal,
    age,
    weight,
    motivation,
    exercise_intensity,
    exercise_duration
])
```



تمرین چهارم: طبقه بند مبتنی بر قوانین فازی برای پیش بینی نتایج تحصیلی

وظایف پروژه:

وظیفه ۱: پیش پردازش و اکتشاف داده ها

- بارگذاری دیتاست و بررسی مشکلات قالبندی.
- پاکسازی و پیش پردازش داده ها (مثل مدیریت مقادیر گمشده، تبدیل انواع داده ها).
- انجام تحلیل اکتشافی داده (EDA) و ترسیم الگوهای کلیدی.
- تقسیم دیتاست به مجموعه آموزش (۸۰٪) و آزمون (۲۰٪) با استفاده از نمونه گیری طبقه بندی شده.

خروجی مورد انتظار: گزارش مختصر EDA با مشاهدات و دیتاست تمیز و تقسیم شده.

بارگذاری و تحلیل اولیه داده ها

بررسی اولیه داده ها با استفاده از توابع پایه پانداس مانند `head()`, `info()` و `describe()` انجام شد تا درک کلی از ساختار داده ها، انواع ویژگی ها و آمار توصیفی آن ها بدست آید. این بررسی نشان داد که مجموعه داده شامل ویژگی های متنوعی از اطلاعات تحصیلی، جمعیت شناختی و اقتصادی دانشجویان است که می تواند در پیش بینی احتمال ترک تحصیل مورد استفاده قرار گیرد. همچنین با بررسی توزیع متغیر هدف، مشخص شد که عدم توازن در کلاس ها وجود دارد که باید در مراحل بعدی مدل سازی مورد توجه قرار گیرد.

تقسیم بندی داده ها

برای اطمینان از ارزیابی دقیق مدل، داده ها به دو بخش آموزش (۸۰٪) و آزمون (۲۰٪) تقسیم شدند. در این فرایند، از روش نمونه گیری طبقه بندی شده استفاده شد تا توزیع متغیر هدف (ترک تحصیل) در هر دو مجموعه حفظ شود. این کار با استفاده از پارامتر `stratify` در تابع `train_test_split` انجام شد که برای داده های نام توازن بسیار مهم است، زیرا اطمینان می دهد که هر دو مجموعه نمایندگی خوبی از داده های اصلی باشند.

```
train_set, test_set = train_test_split(  
    df,  
    test_size=0.2,  
    random_state=SEED,  
    stratify=df[ "Target" ]  
)
```

شناسایی ویژگی های با چولگی بالا

تحلیل چولگی برای ویژگی‌های عددی انجام شد تا توزیع غیرنرمال شناسایی شوند. ویژگی‌هایی با چولگی مطلق بیشتر از ۰.۷ به عنوان ویژگی‌های با چولگی بالا در نظر گرفته شدند. این تحلیل برای انتخاب تبدیل‌های مناسب در مراحل بعدی پیش‌پردازش اهمیت دارد.

```
numeric_cols = df[numeric_features]
skews = numeric_cols.apply(skew, bias=False)
high_skew = skews[skews.abs() > 0.7].index.tolist()
low_skew = [col for col in numeric_features if col not in high_skew]
print(skews)
```

مهندسی ویژگی

برای افزایش قدرت پیش‌بینی مدل، ویژگی‌های جدیدی از ترکیب ویژگی‌های موجود ایجاد شدند. این ویژگی‌های مهندسی شده شامل نسبت‌های موقوفیت تحصیلی (مانند تعداد واحدهای قبول شده به تعداد واحدهای ثبت‌نام شده)، شاخص‌های کارایی (ترکیب نمره و نسبت موقوفیت)، تأثیر شهریه و بورس تحصیلی بر عملکرد، ارتباط تحصیلات والدین با عملکرد دانشجو، و معیارهای مشارکت تحصیلی بودند. این ویژگی‌های ترکیبی، روابط پیچیده‌تری را که ممکن است در پیش‌بینی ترک تحصیل مؤثر باشند، کشف می‌کنند.

```
engineered_features = [
    "approved_per_enrolled_1st_sem", "approved_per_enrolled_2nd_sem",
    "avg_approved_per_enrolled",
    "efficiency_grade_1st", "efficiency_grade_2nd", "tuition_impact",
    "scholarship_boost",
    "mother_qual_grade", "father_qual_grade", "age_efficiency",
    "total_approved", "total_enrolled", "global_approval_ratio",
    "eval_engagement"
]

def add_engineered_features(df):
    df = df.copy()
    df["approved_per_enrolled_1st_sem"] = df["Curricular units 1st sem (approved)"] / df["Curricular units 1st sem (enrolled)"]
    df["approved_per_enrolled_2nd_sem"] = df["Curricular units 2nd sem (approved)"] / df["Curricular units 2nd sem (enrolled)"]
    df['avg_approved_per_enrolled'] = (df["approved_per_enrolled_1st_sem"] + df["approved_per_enrolled_2nd_sem"]) / 2
    df['efficiency_grade_1st'] = df["approved_per_enrolled_1st_sem"] * df["Curricular units 1st sem (grade)"]
    df['efficiency_grade_2nd'] = df["approved_per_enrolled_2nd_sem"] * df["Curricular units 2nd sem (grade)"]
```

```

df['tuition_impact'] = df["Tuition fees up to date"] *
df["avg_approved_per_enrolled"]
df['scholarship_boost'] = df["Scholarship holder"] *
df["avg_approved_per_enrolled"]
df['mother_qual_grade'] = df["Mother's qualification"] *
df["avg_approved_per_enrolled"]
df['father_qual_grade'] = df["Father's qualification"] *
df["avg_approved_per_enrolled"]
df['age_efficiency'] = df["Age at enrollment"] *
df["avg_approved_per_enrolled"]
df['total_approved'] = df["Curricular units 1st sem (approved)"] +
df["Curricular units 2nd sem (approved)"]
df['total_enrolled'] = df["Curricular units 1st sem (enrolled)"] +
df["Curricular units 2nd sem (enrolled)"]
df['global_approval_ratio'] = df["total_approved"] /
df["total_enrolled"]
df['eval_engagement'] = df["Curricular units 1st sem (evaluations)"] +
df["Curricular units 2nd sem (evaluations)"]
return df

feature_engineering_transformer =
FunctionTransformer(add_engineered_features, validate=False)

```

حذف داده‌های پرت

برای مدیریت داده‌های پرت، یک کلاس سفارشی OutlierClipper پیاده‌سازی شد که از روش دامنه میان‌چارکی (IQR) برای شناسایی و محدود کردن مقادیر پرت استفاده می‌کند. این روش مقادیر خارج از محدوده $Q1 - 1.5 \times IQR$ و $Q3 + 1.5 \times IQR$ را شناسایی کرده و آنها را به مرزهای این محدوده محدود می‌کند. این رویکرد به جای حذف کامل نمونه‌های حاوی مقادیر پرت، مقادیر آنها را به مقادیر قابل قبولی تبدیل می‌کند که باعث حفظ تعداد نمونه‌های داده می‌شود.

```

class OutlierClipper(BaseEstimator, TransformerMixin):
    def __init__(self, factor=1.5):
        self.factor = factor
        self.lower_bounds_ = None
        self.upper_bounds_ = None

    def fit(self, X, y=None):
        Q1 = np.percentile(X, 25, axis=0)
        Q3 = np.percentile(X, 75, axis=0)

```

```

IQR = Q3 - Q1
self.lower_bounds_ = Q1 - self.factor * IQR
self.upper_bounds_ = Q3 + self.factor * IQR
return self

def transform(self, X):
    init_len = len(X)
    X_clipped = np.clip(X, self.lower_bounds_, self.upper_bounds_)
    print(f"\n outliers removed: {init_len - len(X_clipped)}")
    return X_clipped

```

نمونه‌گیری متوازن (SMOTE) (امتیازی)

برای مقابله با عدم توازن کلاس‌ها، امکان استفاده از تکنیک Synthetic Minority Over-sampling (SMOTE) در کد پیش‌بینی شده است. این تکنیک با الگوریتم kNN نمونه‌های مصنوعی برای کلاس‌هایی که داده کمتری دارند، تولید می‌کند تا توزیع کلاس‌ها متعادل‌تر شود.

```

from imblearn.over_sampling import SMOTE

if USE_SMOTE:
    smote = SMOTE(random_state=SEED)
    X_train, y_resampled = smote.fit_resample(X_train,
Y_train[["Target_Ordinal"]])

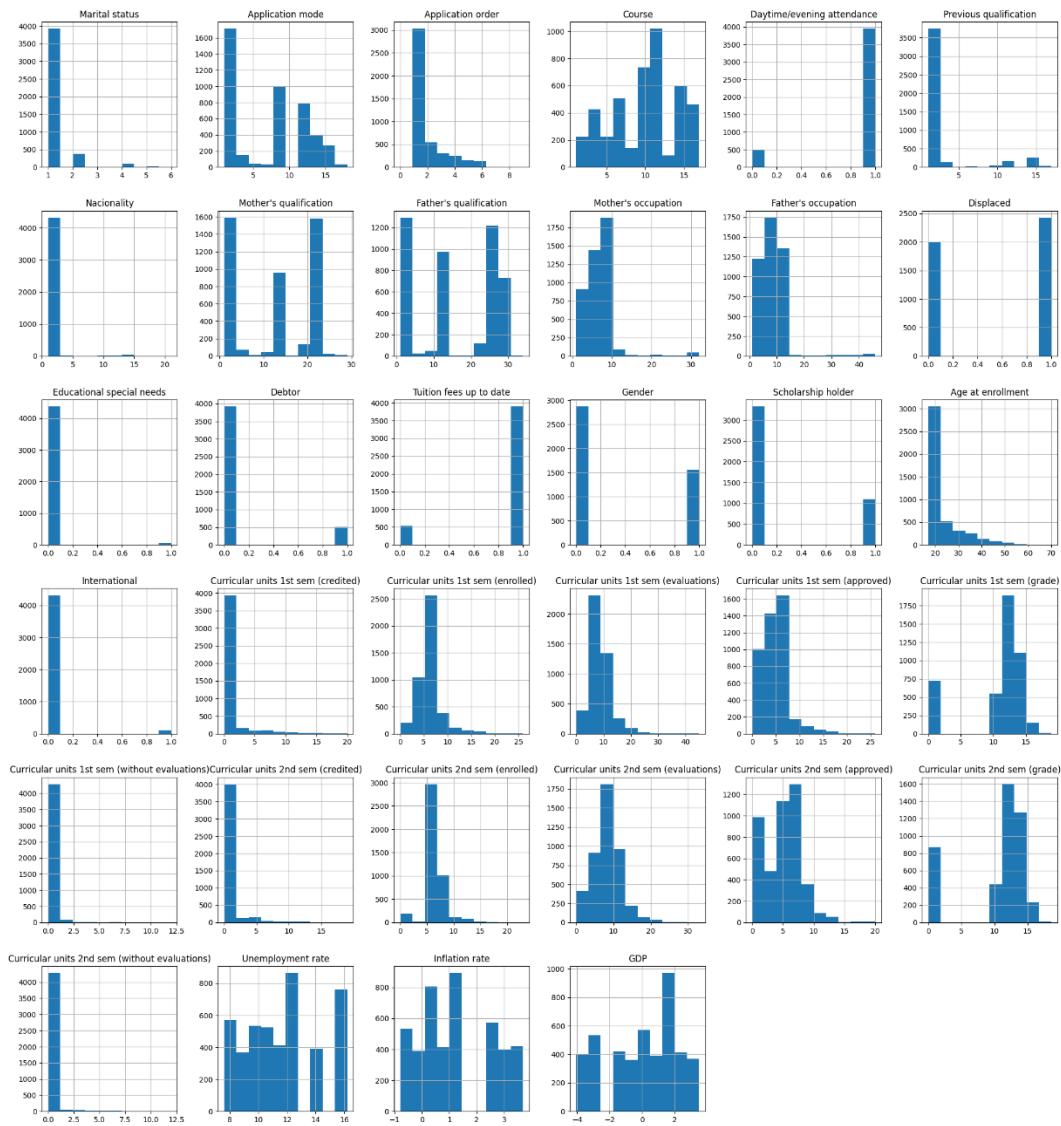
    Y_train = pd.DataFrame({ "Target_Ordinal": y_resampled},
index=np.arange(len(y_resampled)))

    train_set_processed = pd.concat([pd.DataFrame(X_train,
index=Y_train.index), Y_train], axis=1)

```

در زیر نمودارها قبل پیش‌پردازش و بعد پیش‌پردازش آورده شده است.

قبل:



بعد:



انتخاب ویژگی

```

class FeatureSelector:
    random_state = SEED

    @staticmethod
    def _calculate_vif(X_df):
        vif_data = pd.DataFrame()
        vif_data["feature"] = X_df.columns
        vif_data["VIF"] = [variance_inflation_factor(X_df.values, i) for i
in range(X_df.shape[1])]
        return vif_data

```

```

    @staticmethod
    def _find_mi(X_df, Y_df) -> pd.Series:
        mi = mutual_info_classif(X_df, Y_df, discrete_features='auto',
random_state=FeatureSelector.random_state)
        mi_series = pd.Series(mi,
index=X_df.columns).sort_values(ascending=False)

        return mi_series

    @staticmethod
    def _find_colinear_groups(df: pd.DataFrame, threshold=0.9):
        corr_matrix = df.corr().abs()
        upper_triangle =
corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

        G = nx.Graph()
        G.add_nodes_from(df.columns)

        for col in upper_triangle.columns:
            for row in upper_triangle.index:
                corr_value = upper_triangle.loc[row, col]
                if pd.notnull(corr_value) and corr_value > threshold:
                    G.add_edge(row, col)

        groups = list(nx.connected_components(G))

        return groups

    @staticmethod
    def _remove_colinears(feature_groups, mi_series: pd.Series):
        selected_features = []

        for group in feature_groups:
            group_mi = mi_series[list(group)]
            best_feature = group_mi.idxmax()
            selected_features.append((best_feature, mi_series[best_feature]))

        selected_features_sorted = sorted(selected_features, key=lambda x:
x[1], reverse=True)
        selected_features = [f[0] for f in selected_features_sorted]

        return selected_features

    @staticmethod

```

```

def get_top_features_dict(X_df, Y_df, top_n=15, verbose=True):
    mi_series = FeatureSelector._find_mi(X_df, Y_df)
    groups = FeatureSelector._find_colinear_groups(X_df)
    selected_features = FeatureSelector._remove_colinears(groups,
mi_series)
    top_features = selected_features[:top_n]

    # For verbose
    log_df = pd.DataFrame()
    log_df['feature'] = top_features

    vif_df = FeatureSelector._calculate_vif(X_df[top_features])
    log_df = log_df.merge(vif_df, on="feature", how="left")

    log_df['mi'] = log_df['feature'].map(mi_series)

    # corr with target
    corrs = X_df[top_features].apply(lambda x: x.corr(Y_df))
    log_df['corr'] = log_df['feature'].map(corrs)
    log_df = log_df.round(3)

    if verbose:
        print(log_df.to_string(index=True))

    return top_features, log_df

```

```

top_features, info = FeatureSelector.get_top_features_dict(X_train,
Y_train["Target_Ordinal"], top_n=12)

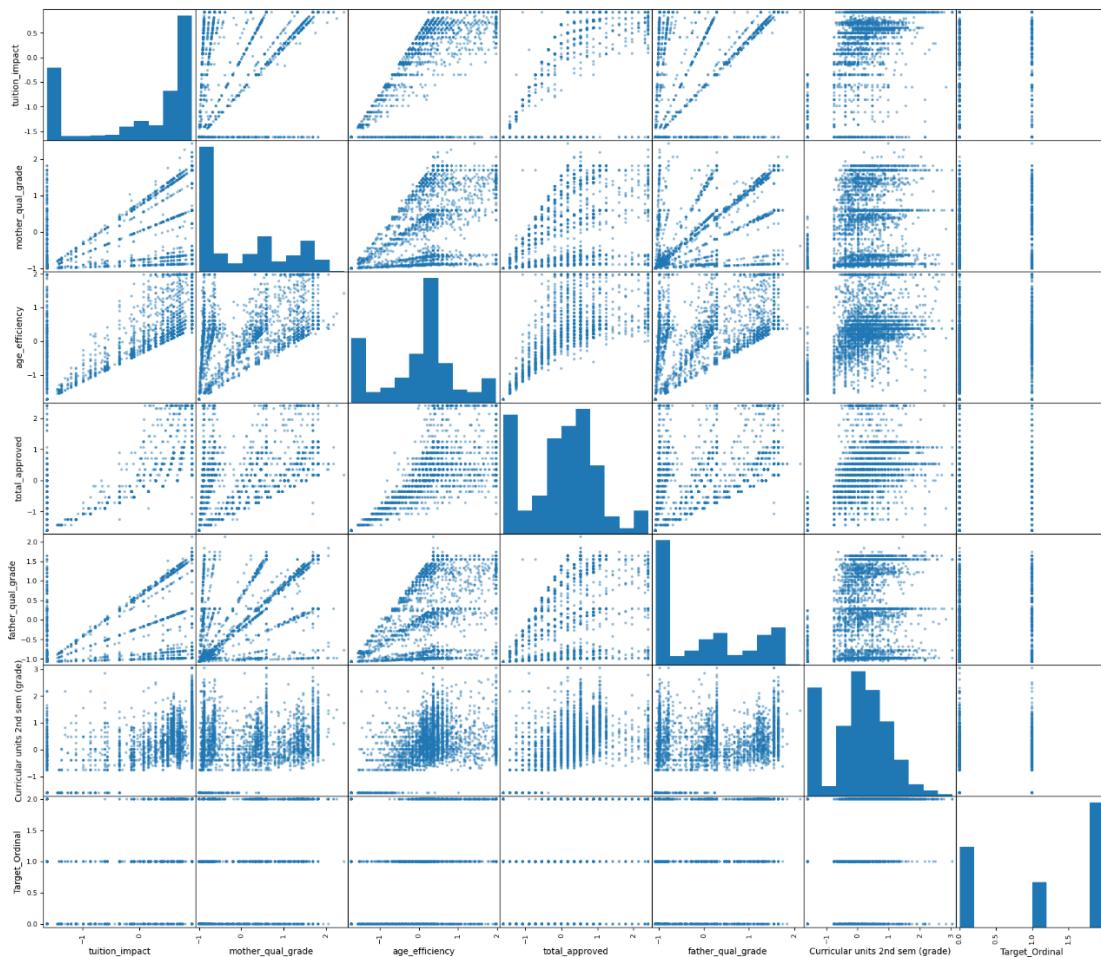
```

برای کاهش ابعاد و بهبود عملکرد مدل، فرایند انتخاب ویژگی پیچیده‌ای پیاده‌سازی شد:

۱. ابتدا اطلاعات بین هر ویژگی و متغیر هدف محاسبه کردیم تا قدرت پیش‌بینی هر ویژگی ارزیابی شود.
۲. سپس یک روش مبتنی بر گراف برای گروه‌بندی ویژگی‌های همبسته پیاده‌سازی کردیم. در این روش، یک گراف ایجاد می‌شود که ویژگی‌ها گره‌های آن هستند و لبه‌ها بین ویژگی‌هایی با همبستگی بالاتر از یک آستانه ایجاد می‌شود.
۳. از هر گروه همبسته، ویژگی با بیشترین اطلاعات متقابل با متغیر هدف انتخاب شد.
۴. ویژگی‌های انتخاب شده بر اساس اطلاعات متقابل مرتب شدند و تعداد مشخصی از برترین‌ها (۱۲ ویژگی) انتخاب شدند.

	feature	VIF	mi	corr
0	tuition_impact	5.811	0.385	0.716
1	mother_qual_grade	1.965	0.354	0.360
2	age_efficiency	8.776	0.330	0.562
3	total_approved	11.321	0.321	0.614
4	father_qual_grade	2.023	0.319	0.402
5	Curricular units 2nd sem (grade)	3.822	0.240	0.572
6	Curricular units 1st sem (grade)	3.166	0.185	0.502
7	Curricular units 2nd sem (evaluations)	1.572	0.094	0.093
8	Tuition fees up to date_1	4.043	0.093	0.403
9	Scholarship holder_0	3.925	0.063	-0.309
10	total_enrolled	3.846	0.060	0.196
11	Age at enrollment	2.272	0.052	-0.303

در نهایت ارتباط بین ویژگی‌های برتر در نمودار زیر قابل مشاهده است.



ارزیابی مدل‌های پایه

```
def evaluate_classifiers_from_split(X_train, X_test, y_train_df,
y_test_df, target_col="Target_Ordinal"):
    models = {
        "Random (Stratified)": DummyClassifier(strategy="stratified",
random_state=SEED),
        "Decision Tree": DecisionTreeClassifier(random_state=SEED),
        "K-Nearest Neighbors": KNeighborsClassifier(n_neighbors=5),
        "Support Vector Machine": SVC(kernel='rbf', C=1.0, gamma='scale',
random_state=SEED),
        "MLP Neural Network": MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='adam',
                                         max_iter=1000,
early_stopping=True, random_state=SEED
    )
}
```

برای ایجاد یک خط مبنا جهت مقایسه عملکرد سیستم فازی، چندین مدل یادگیری ماشین متداول روی داده‌های پیش‌پردازش شده آموزش داده شدند و ارزیابی شدند:

- یک مدل پایه تصادفی (با استراتژی طبقه‌بندی شده)
- درخت تصمیم
- K نزدیک‌ترین همسایه
- ماشین بردار پشتیبان
- شبکه عصبی پرسپترون چندلایه

عملکرد هر مدل با استفاده از معیارهای دقت، گزارش طبقه‌بندی و confusion matrix ارزیابی شد.

```
==== Random (Stratified) ====
Accuracy: 0.4011

Classification Report:
precision    recall    f1-score   support
      0.0      0.34      0.35      0.34      284
      1.0      0.20      0.21      0.20      159
      2.0      0.52      0.50      0.51      442

accuracy                           0.40      885
```

macro avg	0.35	0.35	0.35	885
weighted avg	0.40	0.40	0.40	885

Confusion Matrix:

```
[ [ 99  49 136]
 [ 54  33  72]
 [138  81 223]]
```

==== Decision Tree ===

Accuracy: 0.6983

Classification Report:

	precision	recall	f1-score	support
0.0	0.71	0.73	0.72	284
1.0	0.39	0.42	0.41	159
2.0	0.81	0.78	0.80	442
accuracy			0.70	885
macro avg	0.64	0.64	0.64	885
weighted avg	0.70	0.70	0.70	885

Confusion Matrix:

```
[ [206  44  34]
 [ 46  67  46]
 [ 37  60 345]]
```

==== K-Nearest Neighbors ===

Accuracy: 0.7684

Classification Report:

	precision	recall	f1-score	support
0.0	0.80	0.77	0.79	284
1.0	0.55	0.37	0.44	159
2.0	0.80	0.91	0.85	442
accuracy			0.77	885
macro avg	0.72	0.68	0.69	885
weighted avg	0.75	0.77	0.76	885

Confusion Matrix:

```
[ [220  22  42]
 [ 40  59  60]
 [ 15  26 401]]
```

==== Support Vector Machine ===

Accuracy: 0.7898

Classification Report:

	precision	recall	f1-score	support
0.0	0.84	0.79	0.81	284
1.0	0.59	0.36	0.45	159

```

2.0      0.80      0.94      0.87      442
accuracy
macro avg      0.74      0.70      0.71      885
weighted avg    0.78      0.79      0.77      885

Confusion Matrix:
[[225  21  38]
 [ 37  57  65]
 [  7  18 417]]

==== MLP Neural Network ====
Accuracy: 0.7819

Classification Report:
precision      recall      f1-score      support
0.0          0.79      0.80      0.79      284
1.0          0.66      0.40      0.50      159
2.0          0.80      0.91      0.85      442
accuracy
macro avg      0.75      0.70      0.71      885
weighted avg    0.77      0.78      0.77      885

Confusion Matrix:
[[228  15  41]
 [ 38  63  58]
 [ 24  17 401]]

```

وظیفه ۲: فازی‌سازی و بیزگی‌ها

- اعمال توابع عضویت مثلثی بر ویژگی‌های پیوسته (مثلاً نمره پذیرش، نرخ بیکاری).
- تخصیص برچسب‌های فازی (مثلاً کم، متوسط، زیاد) به ویژگی‌های طبقه‌ای و باینتری.

```

import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder

class Fuzzifier:
    def __init__(self, features: list[str]):
        self.features = features
        self.label_encoder = None
        self.min_max_values = None
        self.bin_feats = []
        self.cont_feats = []

    def fit(self, df: pd.DataFrame, target_col: str):

```

```

"""
Fit the fuzzifier on the training data by computing the min and
max values
for each continuous feature.
"""

self.label_encoder = LabelEncoder()
self.y_true = self.label_encoder.fit_transform(df[target_col])
self.min_max_values =
self._get_min_max_for_continuous_features(df)
    self.bin_feats = [f for f in self.features if
self.is_binary(df[f])]
    self.cont_feats = [f for f in self.features if not
self.is_binary(df[f])]

def transform(self, df: pd.DataFrame):
    memberships = pd.DataFrame(index=df.index)

    bin_feats = self.bin_feats
    cont_feats = self.cont_feats

    # Binary features (crisp-like)
    for B in bin_feats:
        memberships[f"{B}_is_1"] = (df[B] == 1).astype(float)
        memberships[f"{B}_is_0"] = (df[B] == 0).astype(float)
        memberships[f"{B}_fuzzy_label"] = np.where(
            memberships[f"{B}_is_1"] >= memberships[f"{B}_is_0"], '1',
            '0'
        )

    # Continuous features (triangular)
    for X in cont_feats:
        x = df[X].astype(float).to_numpy()
        m, M = self.min_max_values[X]

        if M == m:
            mu_low = np.zeros_like(x)
            mu_high = np.zeros_like(x)
            mu_med = np.ones_like(x)

        else:
            a = m
            b = (m + M) / 2
            c = M

```

```

        mu_low = self.triangular_mf(x, a=a, b=a, c=b)
        mu_med = self.triangular_mf(x, a=a, b=b, c=c)
        mu_high = self.triangular_mf(x, a=b, b=c, c=c)

        memberships[f"X_Low"] = mu_low
        memberships[f"X_Medium"] = mu_med
        memberships[f"X_High"] = mu_high

        cols = [f"X_{lbl}" for lbl in ["Low", "Medium", "High"]]
        memberships[f"X_fuzzy_label"] = (
            memberships[cols].idxmax(axis=1).str.replace(f"X_", "")
        )

    # for target

    return memberships

def fit_transform(self, df: pd.DataFrame, target_col: str):
    self.fit(df, target_col)
    return self.transform(df)

def _get_min_max_for_continuous_features(self, df: pd.DataFrame) -> dict:
    min_max_values = {}
    for feature in self.features:
        if not self.is_binary(df[feature]):
            min_max_values[feature] = (df[feature].min(),
df[feature].max())

    return min_max_values

@staticmethod
def triangular_mf(x: np.ndarray, a: float, b: float, c: float) -> np.ndarray:
    out = np.zeros_like(x, dtype=float)

    # For the rising edge
    rise = (x >= a) & (x < b)
    out[rise] = (x[rise] - a) / (b - a)

    # For the falling edge
    fall = (x >= b) & (x <= c)

    if b != c:

```

```

        out[fall] = (c - x[fall]) / (c - b)

    else:
        out[fall] = 1 # Handle the case where b == c by setting it to
1

    return out

def is_binary(self, series: pd.Series) -> bool:
    """
    Check if a series is binary (0, 1 values).
    """
    return set(series.dropna().unique()) <= {0, 1}

```

فرآیند فازی‌سازی داده‌های عددی دقیق را به مقادیر فازی با درجات عضویت تبدیل می‌کند تا بتوان از منطق فازی برای پیش‌بینی استفاده کرد. برای این کار ویژگی‌ها را به دو دسته کلی تقسیم می‌شوند: ویژگی‌های باینری (دو دویی) که فقط مقادیر صفر و یک دارند و ویژگی‌های پیوسته که طیفی از مقادیر را می‌پذیرند.

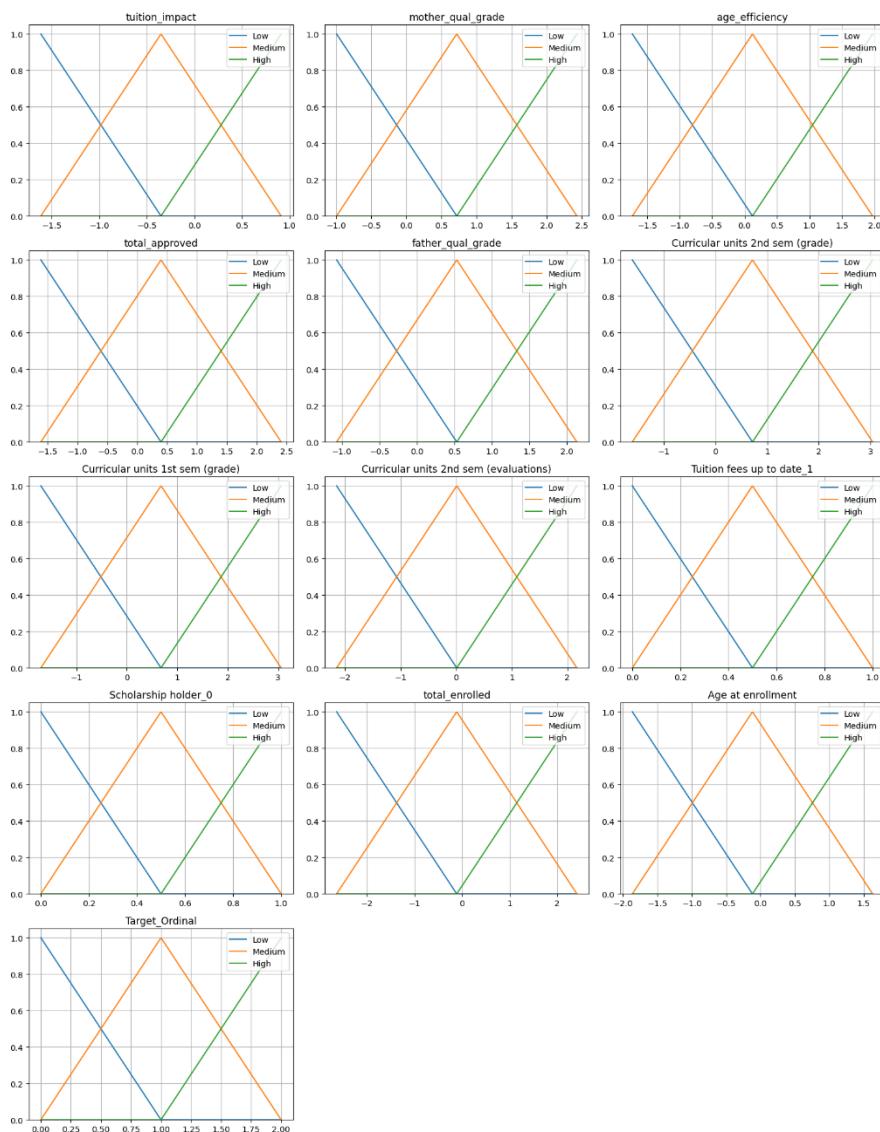
در مرحله آماده‌سازی، فازی‌ساز آمارهای اساسی مانند مقادیر حداقل و حداکثر برای هر ویژگی پیوسته را محاسبه می‌کند تا بتواند توابع عضویت مثلثی را مناسب تعریف کند. همچنین یک کدگذار برچسب برای متغیر هدف تنظیم می‌شود تا بتواند بین برچسب‌های فازی و مقادیر عددی تبدیل انجام دهد.

برای ویژگی‌های باینری مانند جنسیت یا داشتن بورس تحصیلی، فازی‌سازی نسبتاً ساده است. از آنجا که این ویژگی‌ها فقط دو مقدار دارند، برای هر ویژگی دو ستون درجه عضویت ایجاد می‌شود که نشان می‌دهد هر نمونه با چه درجه‌ای به مجموعه‌های صفر یا یک تعلق دارد. در اینجا اگر مقدار اصلی صفر باشد، درجه عضویت در مجموعه صفر برابر با یک و در مجموعه یک برابر با صفر خواهد بود و بالعکس. همچنین یک ستون برچسب فازی نیز ایجاد می‌شود که مقدار ۰ یا ۱ را بر اساس بزرگترین درجه عضویت نگه می‌دارد.

برای ویژگی‌های پیوسته، فازی‌سازی با استفاده از توابع عضویت مثلثی انجام می‌شود. دامنه مقادیر هر ویژگی به سه قسمت تقسیم می‌شود: کم (از حداقل مقدار تا میانه دامنه)، متوسط (از حداقل تا جداکثر مقدار، با اوج در میانه) و زیاد (از میانه تا جداکثر مقدار). تابع عضویت مثلثی به گونه‌ای تعریف می‌شود که در نقطه اوج، درجه عضویت برابر با یک و در نقاط ابتدایی و انتهایی برابر با صفر است. در نقاط بین این مقادیر، درجه عضویت به صورت خطی تغییر می‌کند.

برای هر ویژگی پیوسته، سه ستون درجه عضویت ایجاد می‌شود که نشان می‌دهد هر نمونه با چه درجه‌ای به مجموعه‌های کم، متوسط و زیاد تعلق دارد. همچنین یک ستون برچسب فازی نیز ایجاد می‌شود که مقدار Low، Medium یا High را بر اساس بزرگترین درجه عضویت نگه می‌دارد. این برچسب‌ها در مراحل بعدی برای ساخت قوانین فازی استفاده می‌شوند.

در موارد خاص مانند ویژگی‌های پیوسته با مقدار ثابت (که همه نمونه‌ها در آن مقدار یکسانی دارند)، درجه عضویت متوسط برابر با یک و سایر درجات صفر در نظر گرفته می‌شوند.
نمودار توابع عضویت در زیر آمده است.



وظیفه ۳: استخراج قوانین فازی (روش Wang-Mendel)

- تولید قوانین فازی از نوع IF-THEN از داده‌های آموزشی.
- هر قانون شامل یک مقدمه (Antecedent)، برچسب کلاس (نتیجه) و وزن اطمینان باشد.
- حفظ مهم ترین قانون برای هر مجموعه مقدمه منحصر به فرد بر اساس وزن.

```
def generate_wang_mendel_rules(df: pd.DataFrame, fuzzy_features: list,
target_col: str):
    """
    Generate candidate fuzzy IF-THEN rules using Wang-Mendel method.

    Args:
        df: DataFrame with fuzzy memberships and fuzzy labels.
        fuzzy_features: list of feature names (without suffixes).
        target_col: target column name ("Target").

    Returns:
        List of tuples (antecedents, consequent, weight)
    """
    rules = []

    for idx, row in df.iterrows():
        antecedents = []
        degrees = []

        for feature in fuzzy_features:
            label = row[f'{feature}_fuzzy_label']
            membership_degree = row[f'{feature}_is_{label}'] if label in
['0', '1'] else row[f'{feature}_{label}']

            antecedents.append((feature, label))
            degrees.append(membership_degree)

        consequent = row[f'{target_col}_fuzzy_label']

        # print(degrees)

        weight = np.min(degrees)

        rules.append((antecedents, consequent, weight))
```

```

    return rules

def prune_rules(rules):
    """
    Keep only the highest-weight rule per unique antecedent.

    Args:
        rules: List of (antecedents, consequent, weight)

    Returns:
        Pruned list of rules
    """
    antecedent_groups = {}

    for antecedents, consequent, weight in rules:
        antecedents_key = tuple(antecedents)

        if antecedents_key not in antecedent_groups:
            antecedent_groups[antecedents_key] = (consequent, weight)

        else:
            # Keep rule with higher weight
            if weight > antecedent_groups[antecedents_key][1]:
                antecedent_groups[antecedents_key] = (consequent, weight)

    pruned_rules = []
    for antecedents_key, (consequent, weight) in antecedent_groups.items():
        pruned_rules.append((antecedents_key, consequent, weight))

    return pruned_rules

target_col = "Target_Ordinal"

wang_mendel_rules = generate_wang_mendel_rules(memberships_train,
                                                top_features, target_col)
pruned_rules = prune_rules(wang_mendel_rules)

print(f"\n rules: {len(pruned_rules)}")
print("Top 5 rules:")
for antecedents, consequent, weight in sorted(pruned_rules, key=lambda x:
                                              x[2], reverse=True)[:5]:

```

```

rule_text = " AND ".join([f"{feat} IS {label}" for feat, label in
antecedents])
print(f"IF {rule_text} THEN {consequent} [weight={weight:.4f}]")

```

در این روش، برای هر نمونه در مجموعه داده آموزشی، یک قانون فازی تولید می‌شود. هر قانون شامل سه بخش اصلی است:

۱. مقدمه (**Antecedent**): شامل ترکیبی از عبارات فازی برای هر ویژگی (مانند "اگر global_approval_ratio کم است و efficiency_grade_1st متوسط است و ...")

۲. نتیجه (**Consequent**): برچسب کلاس خروجی برای آن نمونه که نشان‌دهنده وضعیت ترک تحصیل دانشجو است

۳. وزن اطمینان (**Weight**): عددی بین ۰ و ۱ که اطمینان به این قانون را نشان می‌دهد

برای هر نمونه آموزشی، ابتدا برچسب فازی هر ویژگی (مانند کم، متوسط یا زیاد) و درجه عضویت آن در مجموعه فازی مربوطه استخراج می‌شود. سپس یک قانون با استفاده از برچسب‌های فازی همه ویژگی‌ها به عنوان مقدمه و برچسب کلاس نمونه به عنوان نتیجه ایجاد می‌شود. وزن اطمینان قانون بر اساس حداقل درجه عضویت در میان تمام ویژگی‌های موجود در مقدمه تعیین می‌شود، که این روش منطق عملگر AND فازی را منعکس می‌کند.

از آنجا که تعداد زیادی نمونه آموزشی وجود دارد، تعداد زیادی قانون تولید می‌شود که بسیاری از آنها ممکن است مقدمه‌های مشابهی داشته باشند اما نتایج متفاوتی ارائه دهند. برای مدیریت این تضاد و کاهش تعداد قوانین، یک مرحله هرس‌کردن اجرا می‌شود. در این مرحله، قوانین بر اساس مقدمه‌های یکسان گروه‌بندی می‌شوند و از هر گروه، قانونی با بالاترین وزن اطمینان حفظ می‌شود و بقیه حذف می‌شوند. این کار با استفاده از یک ساختار داده دیکشنری انجام می‌شود که مقدمه قوانین را به عنوان کلید و زوج (نتیجه، وزن) را به عنوان مقدار نگه می‌دارد.

پس از پردازش تمام نمونه‌های آموزشی و هرس‌کردن قوانین، قوانین نهایی بر اساس وزن مرتب می‌شوند و پنج قانون برتر نمایش داده می‌شوند. در نمونه اجرای کد، مشاهده می‌شود که تعداد بزرگی از قوانین اولیه به تعداد کمتری از قوانین غیرتکراری و با وزن بالا کاهش یافته است.

این قوانین تصویری نسبتاً شفاف از عوامل موثر بر ترک تحصیل دانشجویان ارائه می‌دهند و می‌توانند به مدیران آموزشی کمک کنند تا دانشجویان در معرض خطر را شناسایی کرده و اقدامات پیشگیرانه را پیاده‌سازی کنند.

در مجموع ۷۲۹ قانون ایجاد شد که ۵ مورد برتر آنها در زیر قابل مشاهده است.

```

n rules: 729
Top 5 rules:
IF tuition_impact IS Low AND mother_qual_grade IS Low AND age_efficiency
IS Low AND total_approved IS Low AND father_qual_grade IS Low AND
Curricular units 2nd sem (grade) IS Low AND Curricular units 1st sem

```

(grade) IS Low AND Curricular units 2nd sem (evaluations) IS Low AND Tuition fees up to date_1 IS 1 AND Scholarshipp holder_0 IS 1 AND total_enrolled IS Low AND Age at enrollment IS Low THEN Medium [weight=1.0000]

IF tuition_impact IS Low AND mother_qual_grade IS Low AND age_efficiency IS Low AND total_approved IS Low AND father_qual_grade IS Low AND Curricular units 2nd sem (grade) IS Low AND Curricular units 1st sem (grade) IS Low AND Curricular units 2nd sem (evaluations) IS Low AND Tuition fees up to date_1 IS 0 AND Scholarshipp holder_0 IS 1 AND total_enrolled IS Medium AND Age at enrollment IS High THEN Low [weight=1.0000]

IF tuition_impact IS Low AND mother_qual_grade IS Low AND age_efficiency IS Low AND total_approved IS Low AND father_qual_grade IS Low AND Curricular units 2nd sem (grade) IS Low AND Curricular units 1st sem (grade) IS Low AND Curricular units 2nd sem (evaluations) IS Medium AND Tuition fees up to date_1 IS 0 AND Scholarshipp holder_0 IS 1 AND total_enrolled IS Medium AND Age at enrollment IS High THEN Low [weight=1.0000]

IF tuition_impact IS Low AND mother_qual_grade IS Low AND age_efficiency IS Low AND total_approved IS Low AND father_qual_grade IS Low AND Curricular units 2nd sem (grade) IS Low AND Curricular units 1st sem (grade) IS Low AND Curricular units 2nd sem (evaluations) IS Low AND Tuition fees up to date_1 IS 1 AND Scholarshipp holder_0 IS 1 AND total_enrolled IS Medium AND Age at enrollment IS High THEN Low [weight=1.0000]

IF tuition_impact IS Low AND mother_qual_grade IS Low AND age_efficiency IS Low AND total_approved IS Low AND father_qual_grade IS Low AND Curricular units 2nd sem (grade) IS Low AND Curricular units 1st sem (grade) IS Low AND Curricular units 2nd sem (evaluations) IS Low AND Tuition fees up to date_1 IS 1 AND Scholarshipp holder_0 IS 1 AND total_enrolled IS Low AND Age at enrollment IS High THEN Low [weight=1.0000]

وظیفه ۴: انتخاب قوانین با استفاده از الگوریتم زنتیک (GA)

- کدگذاری قوانین به صورت کروموزوم باینری برای انتخاب.
- تعریفتابع برازنده‌گی که دقت طبقه‌بندی را پاداش داده و تعداد زیاد قوانین را جریمه کند.
- اجرای GA برای انتخاب بهترین زیرمجموعه قوانین.

این مرحله مهم است چون روش Wang-Mendel تعداد زیادی قانون تولید می‌کند که بسیاری از آنها می‌توانند کم‌اهمیت باشند. هدف، یافتن مجموعه کوچکی از قوانین با بیشترین دقت طبقه‌بندی است. با استفاده از الگوریتم GA برای بهینه سازی.

کلاس FIS (سیستم استنتاج فازی)

```
class FIS:
    def __init__(self, rule_base, fuzzy_df, target_col,
label_encoder=None, inference=False):
        self.label_encoder = label_encoder

        if not label_encoder:
            self.label_encoder = LabelEncoder()

        self.fuzzy_df = fuzzy_df
        if not inference:
            self.y_true =
self.label_encoder.fit_transform(fuzzy_df[target_col])

        else:
            self.y_true = self.label_encoder.transform(fuzzy_df[target_col])

        self.majority_class = np.bincount(self.y_true).argmax()

        self.rule_base = rule_base
        self.n_rules = len(rule_base)

        self.rule_fires = np.empty((self.n_rules, len(fuzzy_df)),
dtype=bool)    # boolean mat, which rules fire for each data point?
        self.firing_degrees = np.empty_like(self.rule_fires,
dtype=float)    # how much each rule fires for each data point
        self.rule_weights = np.empty(self.n_rules)
        self.rule_consequents = np.empty(self.n_rules, dtype=int)      #
labels

        self.compute_inference_matrices()
def compute_inference_matrices(self):
    for i, (ants, consq, weight) in enumerate(self.rule_base):    # for
each rule in the base
        # check this rule against all data points
        fire_bool = np.ones(len(self.fuzzy_df), dtype=bool)
        fire_degree = np.ones(len(self.fuzzy_df), dtype=float)

        # for each feature in ants
        for feat, lbl in ants:
            col = f"{feat}_is_{lbl}" if lbl in ['0', '1'] else
f"{feat}_{lbl}"
```

```

        fire_bool = fire_bool & (self.fuzzy_df[f"{{feat}}_fuzzy_label"] == lbl)
        fire_degree = np.minimum(fire_degree, self.fuzzy_df[col].values)

        self.rule_fires[i] = fire_bool
        self.firing_degrees[i] = fire_degree
        self.rule_weights[i] = weight
        self.rule_consequents[i] =
self.label_encoder.transform([conseq])[0]

        # normalize weights to [0, 1]
        self.rule_weights = (self.rule_weights - self.rule_weights.min()) /
(self.rule_weights.max() - self.rule_weights.min() + 1e-6)

def predict(self, selected_rules_mask=None):
    if selected_rules_mask is None:
        selected_rules_mask = np.ones(self.n_rules, dtype=bool)
        print("Warning: using all the rules.")

    assert len(selected_rules_mask) == self.n_rules

    selected_rules_mask = selected_rules_mask.astype(bool)

    # If no rules is selected
    # return the majority class
    if not selected_rules_mask.any():
        return np.full_like(self.y_true, self.majority_class),
(self.majority_class == self.y_true).mean()

    rule_weights_selected = self.rule_weights[selected_rules_mask][:,
None]
    firing_degrees_selected = self.firing_degrees[selected_rules_mask]

    weighted_firing_degrees = firing_degrees_selected *
rule_weights_selected

    predicted_classes =
self.rule_consequents[selected_rules_mask][np.argmax(weighted_firing_degrees, axis=0)]
    accuracy = (predicted_classes == self.y_true).mean()

    return predicted_classes, accuracy

```

ابتدا کلاس FIS (Fuzzy Inference System) تعریف شده که مسئول اعمال قوانین فازی روی داده‌ها و تولید پیش‌بینی‌ها را بر عهده دارد. این کلاس عملکردهای اصلی زیر را انجام می‌دهد:

۱. محاسبه ماتریس‌های استنتاج: در تابع `compute_inference_matrices`, برای هر قانون در پایگاه قوانین، محاسبه می‌کند که کدام نمونه‌های داده با این قانون مطابقت دارند و درجه فعال‌سازی هر قانون برای هر نمونه چقدر است. این محاسبات شامل:

- برسی مطابقت برچسب‌های فازی با مقدمه قانون
- استفاده از عملگر MIN فازی برای محاسبه درجه فعال‌سازی کلی قانون
- نرمال‌سازی وزن‌های قوانین

۲. پیش‌بینی: در تابع `predict`, با استفاده از ماسک انتخاب قوانین (که توسط الگوریتم ژنتیک تعیین می‌شود)، استنتاج فازی را انجام می‌دهد:

- برای هر نمونه داده، درجه فعال‌سازی هر قانون منتخب را با وزن آن قانون ضرب می‌کند
- قانونی که بالاترین درجه فعال‌سازی وزن‌دار را دارد برای تعیین کلاس انتخاب می‌شود
- اگر هیچ قانونی فعال نشود، کلاس اکثریت به عنوان پیش‌بینی استفاده می‌شود
- دقت پیش‌بینی را محاسبه و همراه با پیش‌بینی‌ها برمی‌گرداند

۳. ارزیابی: تابع `evaluate` معیارهای مختلف ارزیابی مانند دقت، F1 Score، دقت طبقه‌بندی، فراخوانی و ماتریس آشتفتگی را برای پیش‌بینی‌های انجام شده محاسبه می‌کند تا کیفیت سیستم استنتاج فازی ارزیابی شود.

کد مربوط به ژنتیک در زیر آمده است:

```
MAX_N_RULES = 500
POP_SIZE      = 160
GENERATIONS   = 300
PARENTS       = 50
MUT_PROB      = 0.05
ALPHA         = 0.5
SAFE_MARGIN    = 0.02
PROG_EVERY    = 5

rule_base = sorted(pruned_rules, key=lambda r: r[2],
reverse=True) [:MAX_N_RULES]
```

```

train_FIS = FIS(rule_base, memberships_train,
"Target_Ordinal_fuzzy_label")

def fuzzy_fitness(selected_rules_mask, alpha=ALPHA, margin=SAFE_MARGIN):
    preds, acc = train_FIS.predict(selected_rules_mask)

    rule_overuse_penalty = max(0.0, selected_rules_mask.mean() - margin)

    return acc * (1.0 - alpha * rule_overuse_penalty)

def fitness_func(ga, sol, _):
    return fuzzy_fitness(np.asarray(sol, int))

def on_gen(ga):
    if ga.generations_completed % PROG_EVERY == 0:
        sol, net, _ = ga.best_solution()
        mask = np.asarray(sol, int)
        acc = fuzzy_fitness(mask, alpha=0)
        k = mask.sum()
        pen = max(0.0, mask.mean() - SAFE_MARGIN) * ALPHA
        print(f"[{time.strftime('%H:%M:%S')}] gen
{ga.generations_completed:3d} "
              f"acc={acc:.4f}  k={k}/{train_FIS.n_rules}  pen={pen:.3f}  n
et={net:.4f}")

def run_ga():
    ga = pygad.GA(num_generations      = GENERATIONS,
                  sol_per_pop       = POP_SIZE,
                  num_parents_mating = PARENTS,
                  num_genes         = train_FIS.n_rules,
                  gene_space        = [0, 1],
                  parent_selection_type = "tournament", K_tournament=6,
                  keep_elitism       = POP_SIZE // 10,
                  crossover_type     = "two_points",
                  mutation_type      = "random",
                  mutation_probability = MUT_PROB,
                  mutation_by_replacement= True,
                  fitness_func       = fitness_func,
                  on_generation      = on_gen,
                  parallel_processing = ("process", mp.cpu_count()),
                  stop_criteria       = ["saturate_25"],
                  random_seed=SEED)

```

```

start_time = time.time()
ga.run()
print(f"GA runtime: {time.time() - start_time} seconds")

ga.plot_fitness(title="GA convergence - fuzzy compact rules")

return ga

```

ابتدا پارامترهای الگوریتم ژنتیک تنظیم می‌شوند: اندازه جمعیت ۱۶۰، حداکثر ۳۰۰ نسل، ۵۰ والد برای تولید مثل، احتمال جهش ۰.۰۵، ضریب جریمه ۰.۰۲ و حد آستانه جریمه ۰.۰۵ تعیین شده است. قوانین با بیشترین وزن (حداکثر ۵۰۰ قانون) به عنوان پایگاه قوانین اولیه انتخاب می‌شوند.

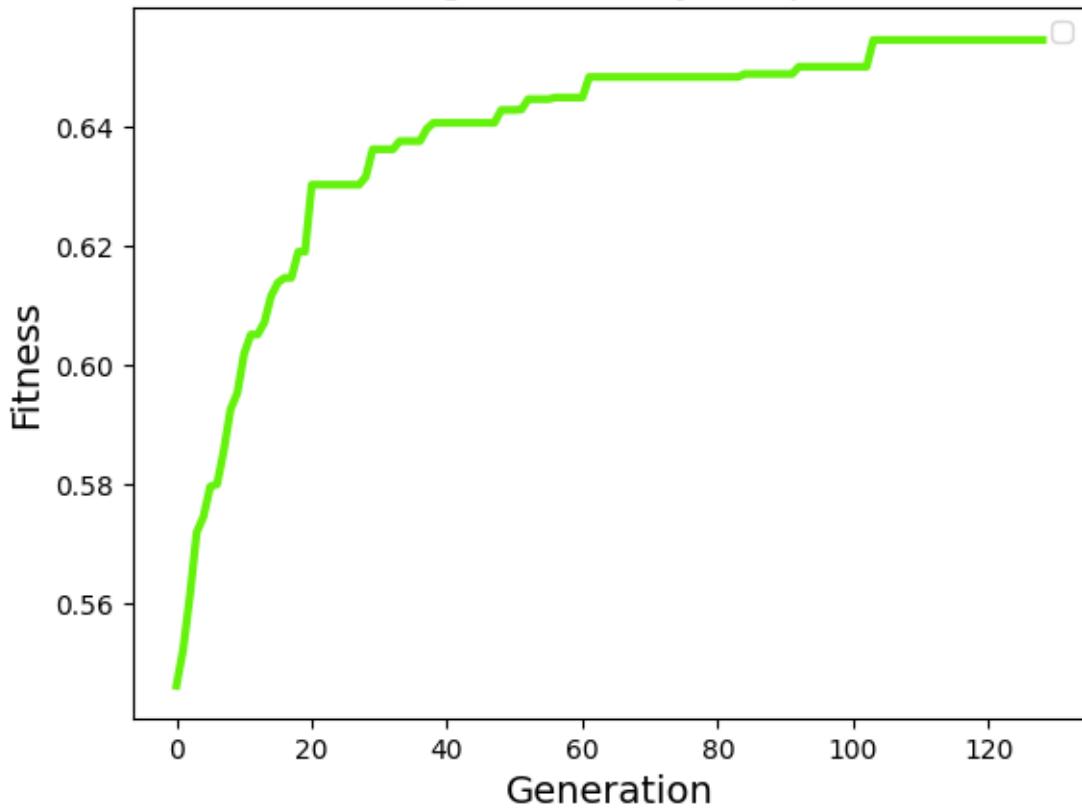
نمایش کروموزوم‌ها به صورت آرایه‌های باینری با طول برابر با تعداد قوانین است. هر ژن (۰ یا ۱) نشان می‌دهد آیا قانون متناظر باید در مدل نهایی استفاده شود یا خیر.تابع برازنده‌گی ترکیبی از دقت طبقه‌بندی و جریمه‌ای برای استفاده از تعداد زیاد قوانین است. این جریمه زمانی اعمال می‌شود که نسبت قوانین انتخاب شده از حد آستانه بیشتر شود.

برای انتخاب والدین از روش تورنمنت با اندازه ۶ استفاده می‌شود. در این روش ۶ کروموزوم به صورت تصادفی انتخاب شده و بهترین آنها به عنوان والد انتخاب می‌شود. عملگر تقاطع از نوع دو نقطه‌ای است که در آن دو نقطه تصادفی انتخاب شده و قسمت میانی کروموزوم‌های والدین با هم تعویض می‌شوند. جهش به صورت تصادفی با احتمال ۰.۰۵ برای هر ژن اتفاق می‌افتد. همچنین ۱۰٪ از بهترین کروموزوم‌ها در هر نسل به طور مستقیم به نسل بعد منتقل می‌شوند (نخبه‌گرایی).

تابع `on_gen` برای نمایش پیشرفت الگوریتم در هر ۵ نسل استفاده می‌شود و اطلاعاتی مانند دقت، تعداد قوانین انتخاب شده و میزان جریمه را نمایش می‌دهد. الگوریتم زمانی متوقف می‌شود که ۲۵ نسل متوالی بدون بهبود قابل توجه در بهترین برازنده‌گی اتفاق بیفتد (معیار توقف "saturate_25").

پردازش موازی با استفاده از تمام هسته‌های پردازنده انجام می‌شود تا سرعت اجرای الگوریتم افزایش یابد. پس از پایان اجرای الگوریتم، بهترین کروموزوم استخراج شده و قوانین متناظر با ژن‌های ۱ در این کروموزوم به عنوان مجموعه قوانین نهایی انتخاب می‌شوند.

GA convergence - fuzzy compact rules



```

Selected 135 / 500 rules
train metrics =
Accuracy: 0.7480
F1 Score: 0.6461
Precision: 0.6938
Recall: 0.6416
Confusion Matrix:
[[1627  89  51]
 [ 191  866  80]
 [ 332  149  154]]

```

در نهایت، عملکرد سیستم استنتاج فازی با مجموعه قوانین انتخاب شده روی داده‌های آموزش ارزیابی شده و گزارشی از دقت، نمره F1، دقت دسته‌بندی، فراخوانی و ماتریس آشفتگی ارائه می‌شود. نتایج نشان می‌دهد که الگوریتم ژنتیک توانسته تعداد قوانین را به طور قابل توجهی از ۱۳۵ مورد به ۷۲۹ مورد کاهش دهد در حالی که دقت طبقه‌بندی را حفظ کرده است.

به دلیل زیاد بودن قوانین پنج مورد در زیر آمده است:

```

01) IF tuition_impact IS Low AND mother_qual_grade IS Low AND
age_efficiency IS Low AND total_approved IS Low AND father_qual_grade IS
Low AND Curricular units 2nd sem (grade) IS Low AND Curricular units 1st
sem (grade) IS Low AND Curricular units 2nd sem (evaluations) IS Medium

```

AND Tuition fees up to date_1 IS 0 AND Scholarship holder_0 IS 1 AND total_enrolled IS Medium AND Age at enrollment IS High → THEN Low (w=1.000)

02) IF tuition_impact IS Low AND mother_qual_grade IS Low AND age_efficiency IS Low AND total_approved IS Low AND father_qual_grade IS Low AND Curricular units 2nd sem (grade) IS Low AND Curricular units 1st sem (grade) IS Low AND Curricular units 2nd sem (evaluations) IS Low AND Tuition fees up to date_1 IS 1 AND Scholarship holder_0 IS 1 AND total_enrolled IS Low AND Age at enrollment IS High → THEN Low (w=1.000)

03) IF tuition_impact IS Low AND mother_qual_grade IS Low AND age_efficiency IS Low AND total_approved IS Low AND father_qual_grade IS Low AND Curricular units 2nd sem (grade) IS Low AND Curricular units 1st sem (grade) IS Low AND Curricular units 2nd sem (evaluations) IS Low AND Tuition fees up to date_1 IS 0 AND Scholarship holder_0 IS 1 AND total_enrolled IS Low AND Age at enrollment IS High → THEN Low (w=0.955)

04) IF tuition_impact IS Low AND mother_qual_grade IS Low AND age_efficiency IS Low AND total_approved IS Low AND father_qual_grade IS Low AND Curricular units 2nd sem (grade) IS Low AND Curricular units 1st sem (grade) IS Low AND Curricular units 2nd sem (evaluations) IS Low AND Tuition fees up to date_1 IS 1 AND Scholarship holder_0 IS 1 AND total_enrolled IS Low AND Age at enrollment IS Medium → THEN Low (w=0.953)

05) IF tuition_impact IS Low AND mother_qual_grade IS Low AND age_efficiency IS Low AND total_approved IS Low AND father_qual_grade IS Low AND Curricular units 2nd sem (grade) IS Low AND Curricular units 1st sem (grade) IS Low AND Curricular units 2nd sem (evaluations) IS Low AND Tuition fees up to date_1 IS 0 AND Scholarship holder_0 IS 1 AND total_enrolled IS Medium AND Age at enrollment IS Medium → THEN Low (w=0.953)

این رویکرد منجر به ایجاد یک سیستم استنتاج فازی بهینه می‌شود که هم دقیق دارد و هم با تعداد کمتری از قوانین، سادگی و قابلیت تفسیر بیشتری ارائه می‌دهد.

وظیفه ۵: استنتاج فازی برای طبقه‌بندی و وظیفه ۶: ارزیابی مدل

- اعمال قوانین فازی انتخاب شده برای طبقه‌بندی نمونه‌های موجود در مجموعه تست.
- تجمعیع مشارکت قوانین و تعیین کلاس پیش‌بینی شده.

کد استنتاج فازی در بالا آورده شده است، کلاس Fis

نتیجه ارزیابی مدل:

```

Test metrics:
Accuracy: 0.7164
F1 Score: 0.5892
Precision: 0.6093
Recall: 0.5984
Confusion Matrix:
[[398 20 24]
 [ 49 213 22]
 [ 82 54 23]]

```

تحلیل معیارهای ارزیابی:

۱. دقت (Accuracy): مدل توانسته ۷۱.۶۴٪ از نمونه‌های آزمون را به درستی طبقه‌بندی کند که نتیجه نسبتاً خوبی است.
۲. مقدار F1 Score: نشان می‌دهد که مدل تعادل نسبی بین دقت و فراخوانی دارد، اما کمتر از دقت کلی است.
۳. دقت دسته‌بندی (Precision): مقدار ۶۰.۹۳٪ بیانگر این است که از کل پیش‌بینی‌های مثبت مدل، حدود ۶۰.۹۳٪ درست بوده‌اند.
۴. فراخوانی (Recall): مقدار ۵۹.۸۴٪ نشان می‌دهد که مدل توانسته حدود ۵۹.۸۴٪ از موارد مثبت واقعی را شناسایی کند.

۵. ماتریس آشنتگی (Confusion Matrix):

- سطر اول: از ۴۴۲ دانشجوی ادامه تحصیل، ۳۹۸ مورد درست تشخیص داده شده‌اند.
- سطر دوم: از ۲۸۴ دانشجوی ترک تحصیل، ۲۱۳ مورد درست تشخیص داده شده‌اند.
- سطر سوم: از ۱۵۹ دانشجوی فارغ‌التحصیل، تنها ۲۳ مورد درست تشخیص داده شده‌اند.

عملکرد ضعیف در کلاس اقلیت: از ماتریس آشنتگی مشخص است که مدل در تشخیص دانشجویان فارغ‌التحصیل (کلاس ۳) ضعیف عمل کرده و تنها ۲۳ مورد از ۱۵۹ مورد را درست تشخیص داده است (۱۴.۵٪).

بهترین عملکرد در کلاس اکثیریت: مدل در تشخیص دانشجویان ادامه تحصیل (کلاس ۱) که تعداد نمونه‌های بیشتری دارند، بهتر عمل کرده است (۹۰٪).

این مقادیر در شرایطی است که از smote استفاده نکرده باشیم.

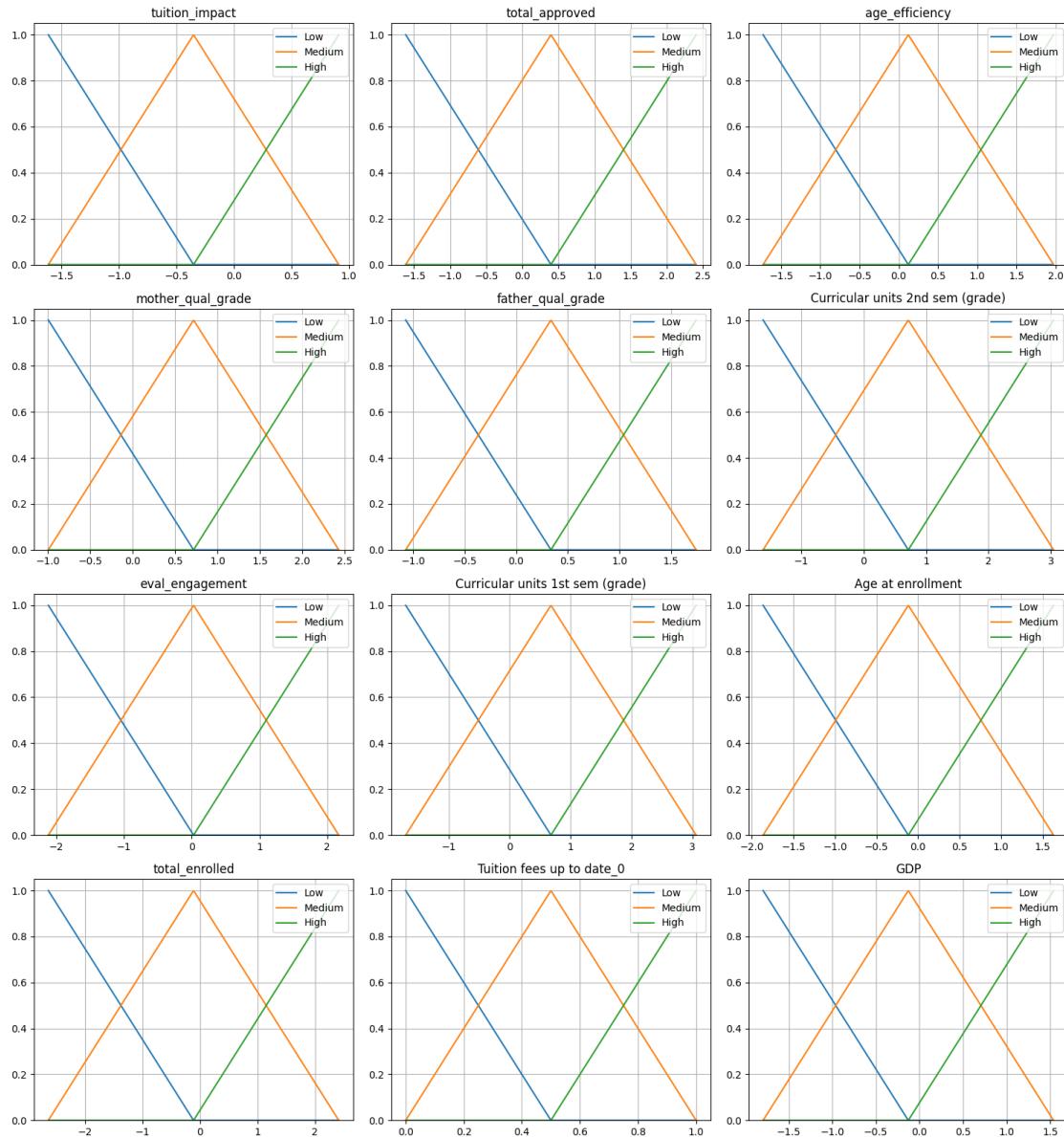
اختلاف بین Accuracy و F1: تفاوت بین دقت کلی (۷۱.۶۴٪) و امتیاز F1 (۵۸.۹۲٪) نشان‌دهنده تأثیر عدم توان اکلاس‌هاست. دقت کلی تحت تأثیر عملکرد خوب در کلاس اکثیریت است.

در زیر مقادیر مرتبط وقتی که از `smote` استفاده شده است:

```
Test metrics:  
Accuracy: 0.7062  
F1 Score: 0.6526  
Precision: 0.6569  
Recall: 0.6518  
Confusion Matrix:  
[[351 16 75]  
 [ 40 203 41]  
 [ 54 34 71]]
```

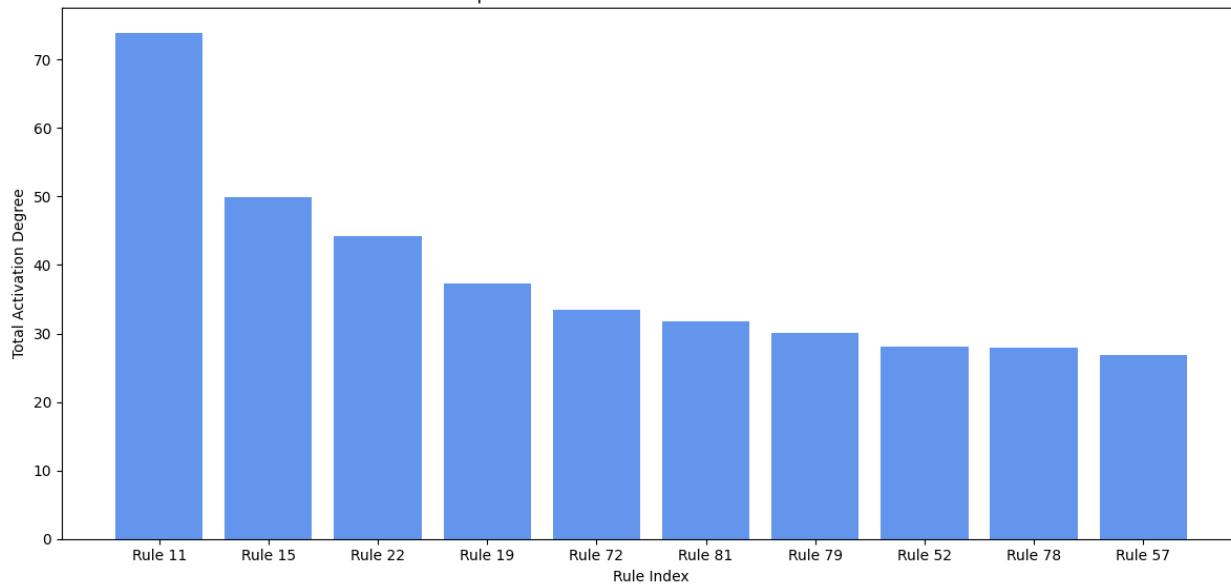
استفاده از تکنیک SMOTE برای متعادل‌سازی داده‌ها در سیستم استنتاج فازی، تأثیر قابل توجهی بر عملکرد مدل در شناسایی وضعیت تحصیلی دانشجویان داشته است. اگرچه دقت کلی اندکی کاهش یافته (از ۷۱.۶۴٪ به ۷۰.۶۲٪)، اما بهبود چشمگیری در شناسایی کلاس اقلیت (دانشجویان فارغ‌التحصیل) مشاهده می‌شود که از ۱۴.۵٪ به ۴۴.۷٪ افزایش یافته است. همزمان، معیارهای مهم دیگر نیز بهبود یافته‌اند: نمره F1 از ۰.۵۸۹ به ۰.۶۵۳، دقت دسته‌بندی از ۰.۶۰۹ به ۰.۶۵۷ و فراخوانی از ۰.۵۹۸ به ۰.۶۵۲ افزایش پیدا کرده است. ماتریس آشفتگی نیز نشان می‌دهد مدل با استفاده از SMOTE توانسته تعادل بهتری بین کلاس‌ها ایجاد کند؛ بطوری که فاصله بین عملکرد بهترین و بدترین کلاس از ۷۵.۵٪ به ۳۴.۷٪ کاهش یافته است. این بهبودها تأیید می‌کنند که SMOTE روشی مؤثر برای مقابله با مشکل عدم توازن داده در این سیستم استنتاج فازی بوده است، به ویژه در مواردی که شناسایی دقیق تمام وضعیت‌های ممکن (نه فقط وضعیت اکثریت) اهمیت دارد، مانند پیش‌بینی دانشجویان در معرض خطر ترک تحصیل یا فارغ‌التحصیلی که برای برنامه‌ریزی آموزشی حیاتی است.

وظیفه ۷: تفسیر و بصری‌سازی



توابع عضویت نمایش داده شده، ویژگی‌های اصلی مدل را به مجموعه‌های فازی کم، متوسط و زیاد تبدیل می‌کنند. این توابع مثلثی با درجات عضویت بین ۰ تا ۱، امکان تعلق نسبی یک مقدار به چند مجموعه را فراهم می‌سازند. محل تقاطع منحنی‌ها، نواحی گذار بین مفاهیم فازی را نشان می‌دهد و این ساختار، پایه استنتاج فازی برای پیش‌بینی وضعیت تحصیلی دانشجویان را شکل می‌دهد.

Top 10 Most Activated Rules on Test Dataset



تحلیل نمودارها و قوانین فازی نشان می‌دهد که سیستم استنتاج فازی طراحی شده برای پیش‌بینی وضعیت تحصیلی دانشجویان (موفقیت، ترک تحصیل یا فارغ‌التحصیلی) الگوهای مهمی را شناسایی کرده است. نمودار اول، ۱۰ قانون با بیشترین میزان فعال‌سازی را نشان می‌دهد که قانون ۱۱ با اختلاف قابل توجهی (۷۳.۸۳) بیشترین میزان فعال‌سازی را دارد. این قانون پیش‌بینی می‌کند دانشجویانی که شهریه خود را پرداخت کرده‌اند، بورسیه نیستند و نمرات متوسطی در دروس دارند، احتمالاً با موفقیت فارغ‌التحصیل خواهند شد.

Key Fuzzy Rules by Activation Level:

1. Rule 11: IF tuition_impact IS High AND mother_qual_grade IS Medium AND age_efficiency IS Medium AND total_approved IS Medium AND father_qual_grade IS Medium AND Curricular units 2nd sem (grade) IS Medium AND Curricular units 1st sem (grade) IS Medium AND Curricular units 2nd sem (evaluations) IS Medium AND Tuition fees up to date_1 IS 1 AND Scholarship holder_0 IS 1 AND total_enrolled IS Medium AND Age at enrollment IS Medium THEN High (Total Activation: 73.83)
2. Rule 15: IF tuition_impact IS High AND mother_qual_grade IS Medium AND age_efficiency IS Medium AND total_approved IS Medium AND father_qual_grade IS Medium AND Curricular units 2nd sem (grade) IS Medium AND Curricular units 1st sem (grade) IS Medium AND Curricular units 2nd sem (evaluations) IS Medium AND Tuition fees up to date_1 IS 1 AND Scholarship holder_0 IS 0 AND total_enrolled IS Medium AND Age at enrollment IS Medium THEN High (Total Activation: 49.85)
3. Rule 22: IF tuition_impact IS Medium AND mother_qual_grade IS Medium AND age_efficiency IS Medium AND total_approved IS Medium AND father_qual_grade IS Medium AND Curricular units 2nd sem (grade) IS Medium AND Curricular units 1st sem (grade) IS Medium AND Curricular units 2nd sem (evaluations) IS Medium AND Tuition fees up to date_1 IS 1 AND Scholarship holder_0 IS 1 AND total_enrolled IS Medium AND Age at enrollment IS Medium THEN Low (Total Activation: 44.21)
4. Rule 19: IF tuition_impact IS High AND mother_qual_grade IS Medium AND age_efficiency IS Medium AND total_approved IS Medium AND father_qual_grade IS High AND Curricular units 2nd sem (grade) IS Medium AND Curricular units

1st sem (grade) IS Medium AND Curricular units 2nd sem (evaluations) IS Medium AND Tuition fees up to date_1 IS 1 AND Scholarship holder_0 IS 1 AND total_enrolled IS Medium AND Age at enrollment IS Medium THEN High (Total Activation: 37.26)

5. Rule 72: IF tuition_impact IS High AND mother_qual_grade IS Medium AND age_efficiency IS Medium AND total_approved IS Medium AND father_qual_grade IS Medium AND Curricular units 2nd sem (grade) IS Low AND Curricular units 1st sem (grade) IS Medium AND Curricular units 2nd sem (evaluations) IS Medium AND Tuition fees up to date_1 IS 1 AND Scholarship holder_0 IS 1 AND total_enrolled IS Medium AND Age at enrollment IS Medium THEN High (Total Activation: 33.50)

6. Rule 81: IF tuition_impact IS Medium AND mother_qual_grade IS Medium AND age_efficiency IS Medium AND total_approved IS Medium AND father_qual_grade IS Medium AND Curricular units 2nd sem (grade) IS Medium AND Curricular units 1st sem (grade) IS Low AND Curricular units 2nd sem (evaluations) IS Medium AND Tuition fees up to date_1 IS 1 AND Scholarship holder_0 IS 1 AND total_enrolled IS Medium AND Age at enrollment IS Medium THEN Medium (Total Activation: 31.80)

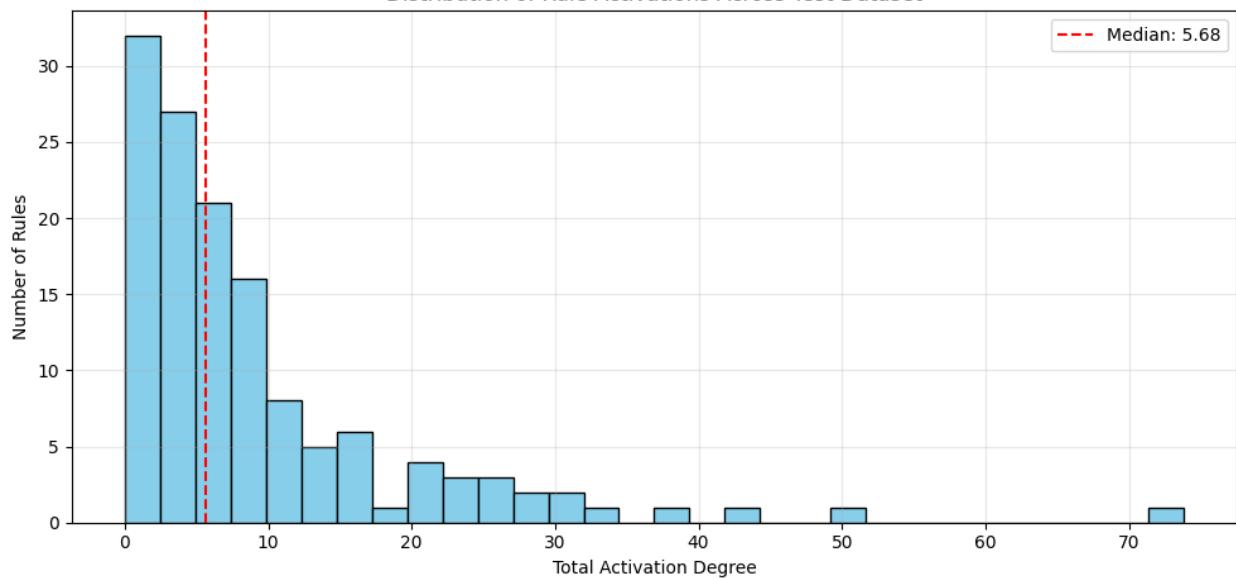
7. Rule 79: IF tuition_impact IS Medium AND mother_qual_grade IS Medium AND age_efficiency IS Medium AND total_approved IS Medium AND father_qual_grade IS Medium AND Curricular units 2nd sem (grade) IS Low AND Curricular units 1st sem (grade) IS Medium AND Curricular units 2nd sem (evaluations) IS Medium AND Tuition fees up to date_1 IS 1 AND Scholarship holder_0 IS 1 AND total_enrolled IS Medium AND Age at enrollment IS Medium THEN Low (Total Activation: 30.16)

8. Rule 52: IF tuition_impact IS Medium AND mother_qual_grade IS Medium AND age_efficiency IS Medium AND total_approved IS Medium AND father_qual_grade IS Low AND Curricular units 2nd sem (grade) IS Medium AND Curricular units 1st sem (grade) IS Medium AND Curricular units 2nd sem (evaluations) IS Medium AND Tuition fees up to date_1 IS 1 AND Scholarship holder_0 IS 1 AND total_enrolled IS Medium AND Age at enrollment IS Medium THEN Medium (Total Activation: 28.09)

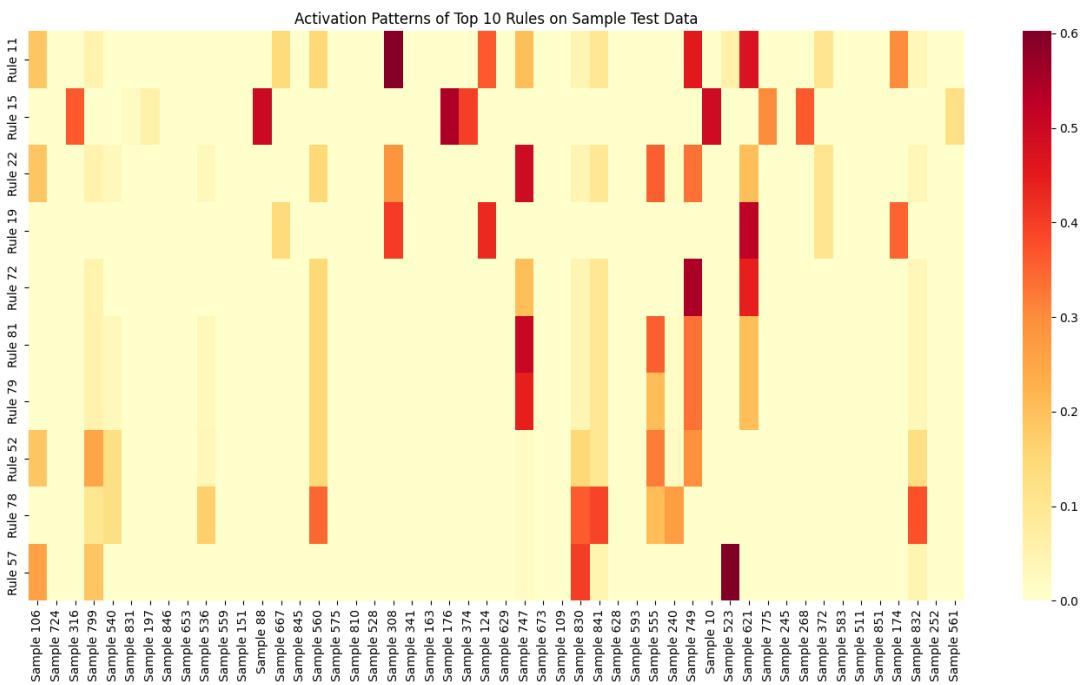
9. Rule 78: IF tuition_impact IS Medium AND mother_qual_grade IS Low AND age_efficiency IS Medium AND total_approved IS Medium AND father_qual_grade IS Low AND Curricular units 2nd sem (grade) IS Low AND Curricular units 1st sem (grade) IS Medium AND Curricular units 2nd sem (evaluations) IS Medium AND Tuition fees up to date_1 IS 1 AND Scholarship holder_0 IS 1 AND total_enrolled IS Medium AND Age at enrollment IS Medium THEN Medium (Total Activation: 27.89)

10. Rule 57: IF tuition_impact IS High AND mother_qual_grade IS Low AND age_efficiency IS Medium AND total_approved IS Medium AND father_qual_grade IS Low AND Curricular units 2nd sem (grade) IS Medium AND Curricular units 1st sem (grade) IS Medium AND Curricular units 2nd sem (evaluations) IS Medium AND Tuition fees up to date_1 IS 1 AND Scholarship holder_0 IS 1 AND total_enrolled IS Medium AND Age at enrollment IS Low THEN High (Total Activation: 26.78)

Distribution of Rule Activations Across Test Dataset



نمودار دوم توزیع فعال‌سازی قوانین را نشان می‌دهد که اکثر قوانین دارای فعال‌سازی کمتر از میانه (۵.۶۸) هستند، در حالی که تعداد کمی از قوانین (مانند قانون ۱۱) فعال‌سازی بالایی دارند. این نشان می‌دهد که سیستم از تعداد کمی قوانین کلیدی برای تصمیم‌گیری استفاده می‌کند که موجب بهینه‌سازی عملکرد و قابلیت تفسیر می‌شود.



نمودار هیتمپ الگوی فعال‌سازی قوانین را برای نمونه‌های مختلف نشان می‌دهد. مشاهده می‌شود قوانین مختلف برای نمونه‌های متفاوت با شدت‌های متعددی فعال می‌شوند که انعطاف‌پذیری سیستم را نشان می‌دهد. بررسی قوانین کلیدی

نشان می‌دهد پرداخت به موقع شهریه و عملکرد متوسط در واحدهای درسی فاکتورهای بسیار مهمی هستند. همچنین تاثیر شهریه (tuition_impact) که از ترکیب پرداخت به موقع و نسبت واحدهای قبول شده به دست می‌آید، و سطح تحصیلات والدین نیز متغیرهای تاثیرگذاری می‌باشند.

رابطه گرافیکی

با استفاده از gradio رابطه گرافیکی زیر را پیاده سازی کردیم.

The image shows a Gradio interface with two main sections: 'input' on the left and 'output' on the right.

input section:

- Curricular units 1st sem (approved): 10
- Curricular units 1st sem (enrolled): 20
- Curricular units 2nd sem (approved): 15
- Curricular units 2nd sem (enrolled): 10
- Curricular units 1st sem (grade): 10
- Curricular units 2nd sem (grade): 10
- Mother's qualification: 11th Year of Schooling—not completed
- Father's qualification: Secondary Education—12th Year of Schooling or Equivalent
- Tuition fees up to date
- Scholarship holder
- Age at enrollment: 25

output section:

- output: Dropout
- Flag