Q1)

کد حل این سوال در فایل pendulum.ipynb موجود است. ابتدا گزارشات تئوری را شرح میدهم و سپس کد را توضیح میدهم.

تعریف متغیرهای زبانی و نحوهی بازه بندی آن ها

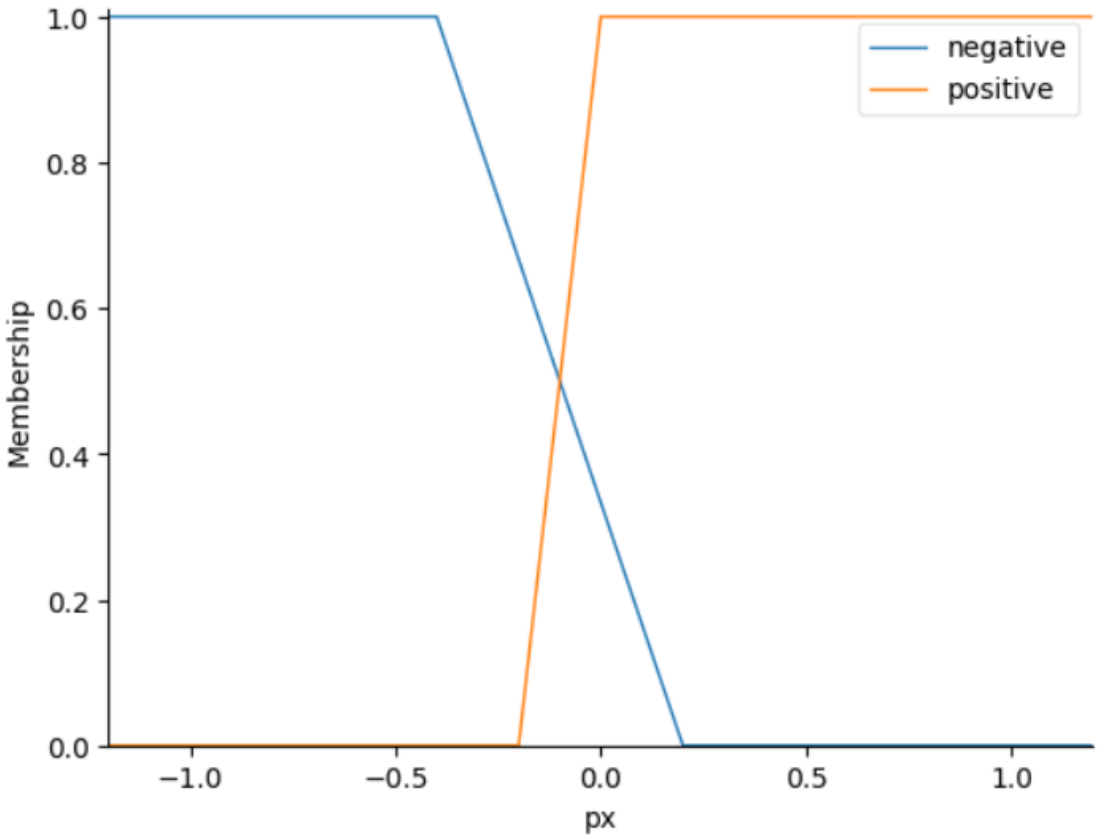
برای انتخاب متغیرهای زبانی، از پارامترهای action و observation خود بازی pendulum استفاده کردم. سیستم کنترلی قرار است با دریافت مشاهدات، یک Action را تولید کند پس، observation های بازی ورودی سیستم کنترلی و action بازی خروجی سیستم کنترلی تعریف میشود.

بازه های متغیرهای زبانی را مطابق با بازه مشخص شده در داکیومنت gym مشخص کردم. البته برای پرهیز از خطا درحالات خاص، تمامی بازه ها را کمی باز تر از داکیومنت gym در نظر گرفتم.

برای تعریف ترم ها، سعی کردم مسئله را پیچیده نکنم و در حالت مینیمال مسئله را حل کنم. برای همین برای هر متغیر تنها دو ترم positive و negative در نظر گرفتم. مسئله با همین ترم ها حل شد و نیازی به پیچیده ترکردن فاز ها نبود.

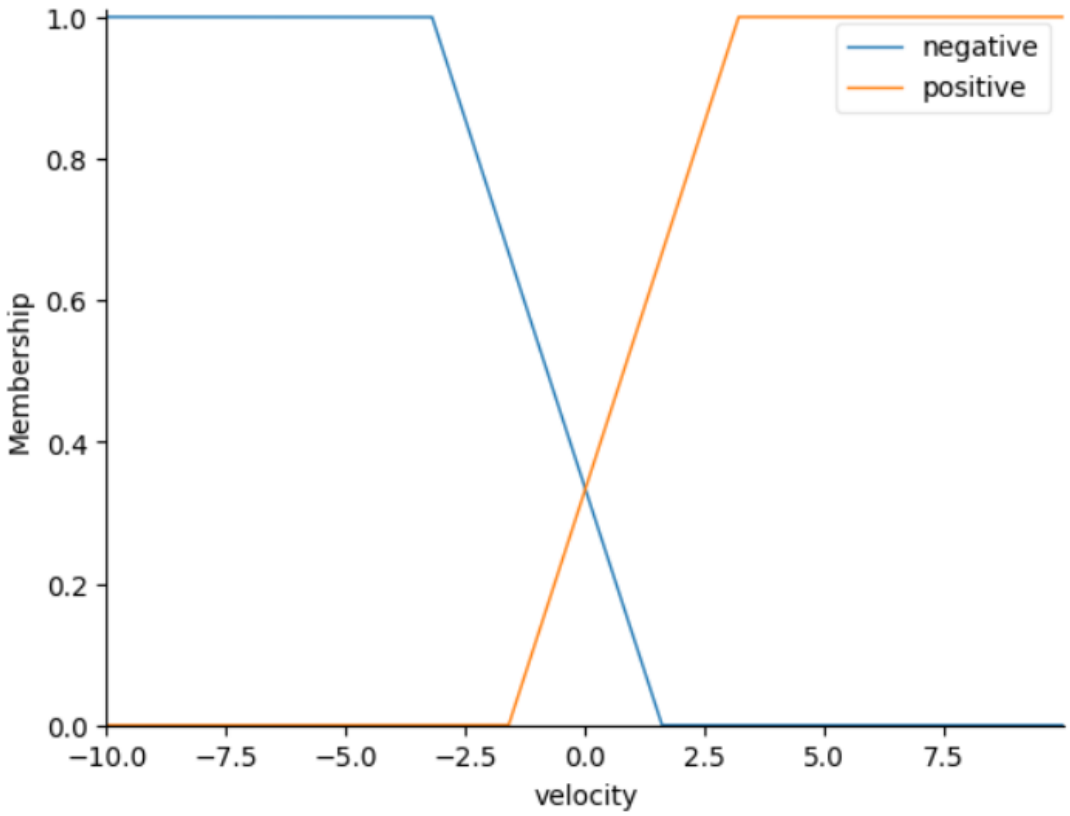
متغیرهای antecedent:

1- Px: مختصات انتهای پاندول در محور x (عمودی)



بازه بندی این متغیر را با آزمون و خطا به دست آوردم. در این حالت سرعت پاندول بهتر تنظیم میشد.

2- Velocity: سرعت حرکت پاندول

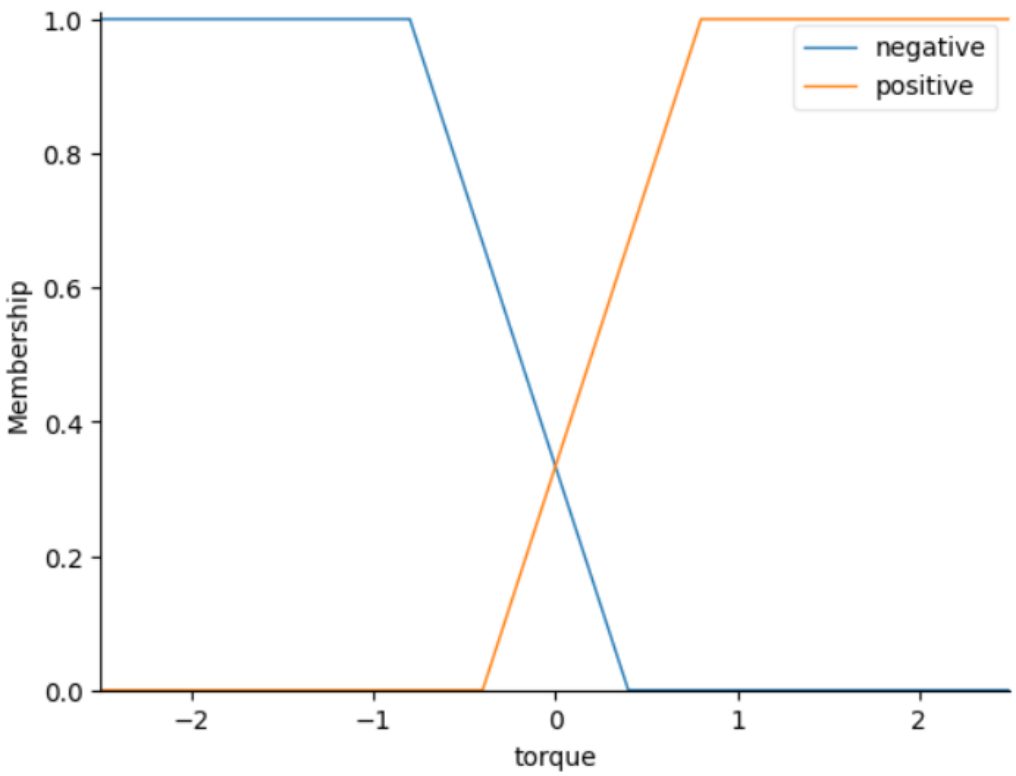


بازه بندی این متغیر را به صورت ساده و قرینه، مشابه با بازه بندی مسئله مشابه حل شده در کلاس حل تمرین قرار دادم.

البته در observation های بازی، مختصات y پاندول هم داده شده. اما همانطور که در کلاس حل تمرین اشاره شد، نیازی به این متغیر برای حل مسئله نبود. من هم برای سادگی و راحتی، این متغیر را تعریف و در محاسبات دخیل نکردم.

متغیرهای consequent:

1- Torque: نیرویی دورانی ای که به پاندول وارد میشود



بازه بندی این متغیر را هم به صورت ساده و پیشفرض انجام دادم تا در صورت نیاز بعدا تغییر دهم اما نیاز نشد.

نحوه تعریف قوانین

قوانین را به این صورت تعریف کردم:

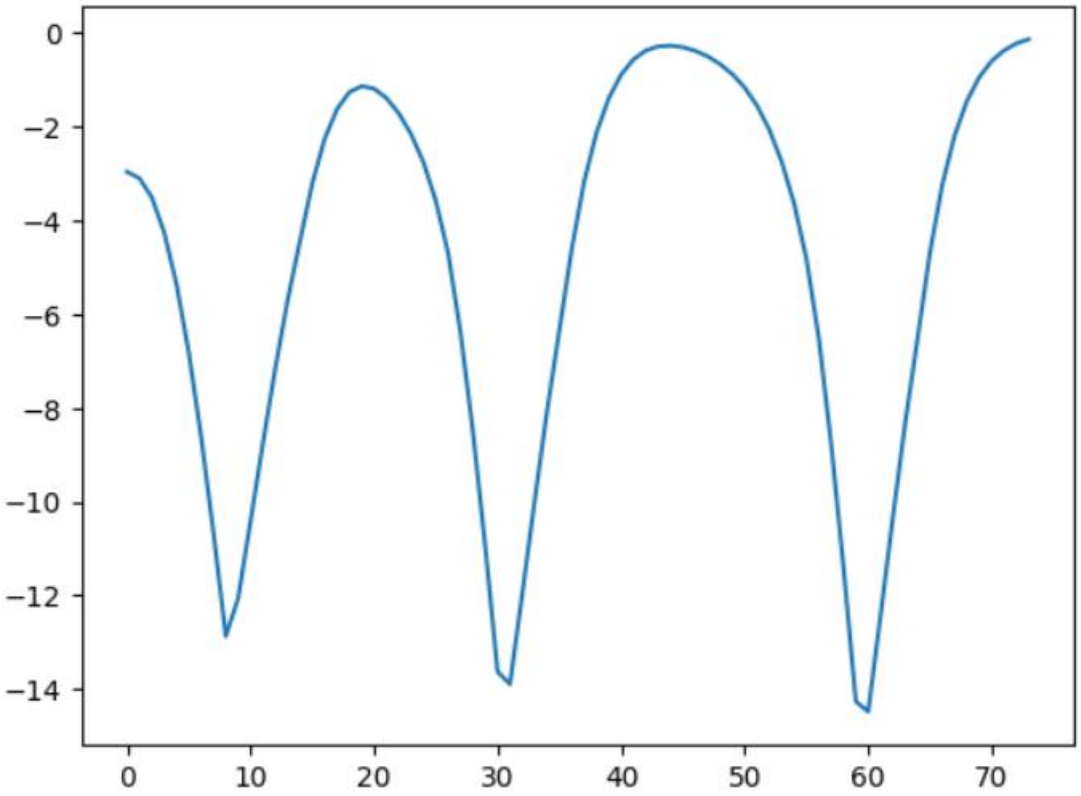
1- زمانی که مختصات x پاندول کم تر از 0 است (پاندول رو به پایین است)، باید به پاندول شتاب بدهیم، پس نیروی دورانی باید هم جهت با سرعت پاندول باشد.

2- زمانی که مختصات x پاندول بیشتر از 1 است (پاندول رو به بالا است)، باید سرعت پاندول را کم کنیم، پس نیروی دورانی خلاف جهت پاندول است.

این قوانین به زبان متغیرها به این چهار شرط تبدیل شدند:

```
rule1 = ctrl.Rule(px['negative'] & velocity['negative'], torque['negative'])
rule2 = ctrl.Rule(px['negative'] & velocity['positive'], torque['positive'])
rule3 = ctrl.Rule(px['positive'] & velocity['positive'], torque['negative'])
rule4 = ctrl.Rule(px['positive'] & velocity['negative'], torque['positive'])
```

نمودار پاداش‌های دریافتی



تحلیل نمودار پاداش

تعریف پاداش دریافتی به این صورت بود:

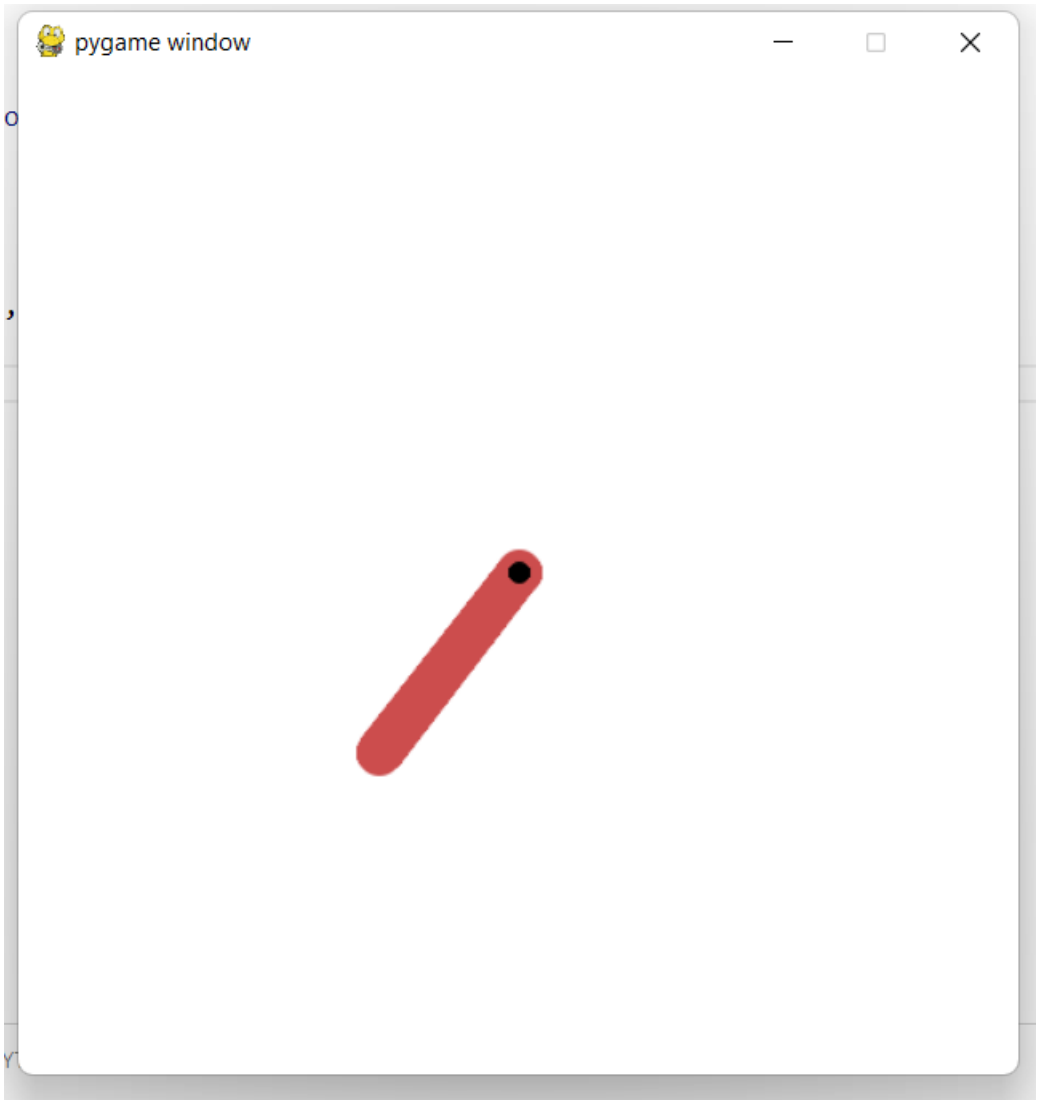
$$r = -(theta^2 + 0.1 * theta\_dt^2 + 0.001 * torque^2)$$

میبینیم که پاداش همیشه منفی است و بیشتری حد آن 0 است. پاداش زمانی به بیشتر حد میرسد که پاندول رو به بالا باشد و سرعت آن هم صفر باشد.

این نمودار نوسان پاندول و بیشتر شدن پاداش را در بالای هر نوسان نشان میدهد. نقاط پایین نمودار زمانی هستند که پاندول با سرعت زیاد از نقطه پایین رد میشود و نقاط بالا زمانی هستند که پاندول سعی میکند که تعادل خود را رو به بالا حفظ کند. در نهایت میبینیم که در مرحله 70، شرط پایان بازی (x بیشتر از 0.99 و سرعت کم تر از 1.5) براورده شده و بازی متوقف شده.

شرح کد

مرحله اول، تنها اجرای بازی به کمک کتابخانه gym بود:




```python
import gymnasium as gym
from time import sleep
env = gym.make('Pendulum-v1', g=9.81, render_mode="human")
observation, info = env.reset(seed=42)

for _ in range(500):
    action = env.action_space.sample()  # this is where you would insert your policy
    observation, reward, terminated, truncated, info = env.step(action)
    print(observation)
    # sleep(1)
    if terminated:
        observation, info = env.reset()

env.close()
```

در این کد که از سایت gym گرفته شده، بازی با اکشن های رندوم اجرا میشود و هیچ انتخابی برای اکشن ها صورت نمیگیرد.

مرحله دوم طراحی سیستم فازی با استفاده از skfuzz بود.

ابتدا ایمپورت ها را انجام دادم:

```
import numpy as np
import skfuzzy as fuzz
import gymnasium as gym
import matplotlib.pyplot as plt
from skfuzzy import control as ctrl
```

سپس متغیرهای توضیح داده شده را تعریف کردم و ترم های آنها را مشخص کردم:

```
px = ctrl.Antecedent(np.arange(-1.2, 1.2, 0.001), 'px')

px['negative'] = fuzz.trapmf(px.universe, [-1.2, -1.2, -0.4, 0.2])
px['positive'] = fuzz.trapmf(px.universe, [-0.2, 0, 1.2, 1.2])

px.view()
```

```
velocity = ctrl.Antecedent(np.arange(-10, 10, 0.001), 'velocity')

velocity['negative'] = fuzz.trapmf(velocity.universe, [-10, -10, -3.2, 1.6])
velocity['positive'] = fuzz.trapmf(velocity.universe, [-1.6, 3.2, 10, 10])

velocity.view()
```

```
torque = ctrl.Consequent(np.arange(-2.5, 2.5, 0.001), 'torque')

torque['negative'] = fuzz.trapmf(torque.universe, [-2.5, -2.5, -0.8, 0.4])
torque['positive'] = fuzz.trapmf(torque.universe, [-0.4, 0.8, 2.5, 2.5])

torque.view()
```

نمودار ترم های این متغیرها بالاتر آورده شده.

سپس قوانین را مطابق توضیحات تعریف کردم:

```
rule1 = ctrl.Rule(px['negative'] & velocity['negative'], torque['negative'])
rule2 = ctrl.Rule(px['negative'] & velocity['positive'], torque['positive'])
rule3 = ctrl.Rule(px['positive'] & velocity['positive'], torque['negative'])
rule4 = ctrl.Rule(px['positive'] & velocity['negative'], torque['positive'])
```

در نهایت کنترلر و شبیه ساز را ایجاد کردم:

```
controller = ctrl.ControlSystem([rule1, rule2, rule3, rule4])
simulator = ctrl.ControlSystemSimulation(controller)
```

در این قسمت، در لوپ اصلی بازی، به جای این که اکشن ها را رندوم انتخاب کنم، ابتدا observation های مورد نیاز برای سیستم فازی را استخراج کردم و به عنوان ورودی به کنترلر دادم، سپس خروجی را محاسبه کردم و آن را به عنوان Action بازی اعمال کردم:

```
px = observation[0]
velocity = observation[2]

simulator.input['px'] = px
simulator.input['velocity'] = velocity
simulator.compute()
decision = simulator.output['torque']
observation, reward, terminated, truncated, info = env.step([decision])
rewards.append(reward)
```

همچنین پاداش هر مرحله را در یک لیست ذخیره کردم.

شرط اتمام بازی را هم مطابق سوال تنظیم کردم:

```
if px > 0.99 and abs(velocity) < 1.5:
    terminated = True
```

نتیجه این شد که در 73 مرحله بازی به اتمام رسید:

```
You Win in 73 iteration!
```

(Q2

الف)

الگوریتم Fuzzy C-Means (FCM) یک الگوریتم خوشه‌بندی است که در مقایسه با الگوریتمK-Means ، هر نقطه به تمام خوشه‌ها با یک درصد وزن اختصاص می‌یابد. این الگوریتم به عنوان یک الگوریتم خوشه‌بندی فازی شناخته می‌شود، زیرا برخلاف K-Means که هر نقطه را به یک خوشه اختصاص می‌دهد، FCMهر نقطه را با درصدهای وزنی مختلف به همه خوشه‌ها اختصاص می‌دهد.

فرایند اجرای Fuzzy C-Means به شرح زیر است:

مقداردهی اولیه:

مشخص کردن تعداد خوشه‌ها.(c)

مشخص کردن پارامتر فازی (m) که یک عدد حقیقی بزرگتر از 1 است و در واقع نشان‌دهنده فازی بودن خوشه‌بندی است.

مقداردهی اولیه به اعضای ماتریس عضویت (U) که نشان‌دهنده درصد وابستگی هر نقطه به هر خوشه است.

تولید مرکزهای خوشه جدید:

محاسبه مرکزهای خوشه با استفاده از ماتریس عضویت و ویژگی‌های نقاط.

بروزرسانی ماتریس عضویت:

محاسبه ماتریس عضویت جدید با استفاده از فاصله نقطه تا مرکزهای خوشه جدید و پارامتر فازی.

ارزیابی همگرایی:

اگر ماتریس عضویت تغییرات چندانی نداشته باشد یا تعداد تکرارها به حد مشخصی برسد، الگوریتم همگرا شده است.

خاتمه:

نقاط به خوشه‌هایی تخصیص داده می‌شوند که در ماتریس عضویت مقدار بیشتری دارند.

مزایای Fuzzy C-Means شامل این است که این الگوریتم به ارائه اطلاعات بیشتر در مورد نحوه توزیع نقاط در خوشه‌ها کمک می‌کند. این الگوریتم در مواردی کاربرد دارد که نقاط به طور همزمان می‌توانند به چندین خوشه تعلق داشته باشند. همچنین، مشکلاتی نظیر حساسیت به مقدار اولیه و مشکل در تعیین تعداد نهایی خوشه‌ها نیز وجود دارد.

ماتریس انتمال (Membership Matrix) در الگوریتم Fuzzy C-Means (FCM) نشان‌دهنده میزان وابستگی هر نقطه به هر خوشه است. در FCM، ماتریس انتمال ابتدا مقادیر تصادفی دارد و سپس در هر مرحله به‌روزرسانی می‌شود.

تفاوت‌های اصلی بین الگوریتم‌های Fuzzy C-Means (FCM) و K-Means به شرح زیر هستند:

وابستگی نقاط به خوشه‌ها:

درK-Means ، هر نقطه به دقت به یک خوشه اختصاص می‌یابد. به این معنا که یک نقطه به‌طور کامل به یک خوشه تعلق دارد.

درFCM ، هر نقطه با یک درصد وزن به همه خوشه‌ها اختصاص می‌یابد. این به این معناست که هر نقطه با میزان وزن مختلف به همه خوشه‌ها تعلق دارد. این باعث می‌شود که اطلاعات بیشتری در مورد انتمال نقاط به خوشه‌ها در اختیار قرار گیرد.

نوع انتمال:

درK-Means ، نقاط به صورت دقیق به یک خوشه تعلق دارند و این ارتباط به صورت دودویی است (نقطه یا به خوشه تعلق دارد یا ندارد).

درFCM ، نقاط با انتمالات فازی به خوشه‌ها تعلق دارند که مقادیر انتمال در بازه [0, 1] قرار دارند.

حساسیت به نقاط خارجی:

K-Meansحساس به نقاط خارجی است و ممکن است به نقاطی که دورتر از مراکز خوشه‌ها قرار دارند حساس باشد.

FCMبه دلیل اختصاص انتمالات به همه خوشه‌ها، در مقابل نویز و نقاط خارجی حساسیت کمتری دارد.

مفهوم خوشه‌ها:

درK-Means ، خوشه‌ها به صورت منطقی و واضح تر تعریف می‌شوند و هر خوشه میانگین نقاطی است که به آن تعلق دارند.

درFCM ، خوشه‌ها به دلیل وجود انتمالات فازی، ترکیبی از نقاط هستند و هر خوشه میانگین نقاطی است که به آن با احتمالات مختلف تعلق دارند.

انعطاف در خوشه‌بندی:

FCMبه دلیل وابستگی فازی نقاط به خوشه‌ها، برای مواردی که نقاط به چندین خوشه تعلق دارند مناسب است.

K-Means معمولاً برای مواردی که خوشه‌ها به صورت واضح تعریف شده‌اند و نقاط به دقت به یک خوشه تعلق دارند، مؤثرتر است.

ب)

ابتدا سوالات تئوری را پاسخ میدهم سپس کد را تحلیل میکنم.

مفهوم FPC:

معیار Fuzzy Partition Coefficient (FPC) یکی از اندازه‌گیرهای ارزیابی کیفیت خوشه‌بندی در الگوریتم‌های خوشه‌بندی فازی مانند Fuzzy C-Meansاست. این معیار از ماتریس عضویت به دست می‌آید و نشان‌دهنده وابستگی نقاط به خوشه‌ها است.

برای درک بهتر معیارFPC ، ابتدا به مفهوم ماتریس عضویت (Membership Matrix) در خوشه‌بندی فازی اشاره کوتاهی میکنم. در الگوریتم‌های خوشه‌بندی فازی، هر نقطه به هر خوشه با درصدی از 0 تا 1 اختصاص می‌یابد. این عضویت‌ها نشان‌دهنده وابستگی هر نقطه به هر خوشه است.

حالا به معیار FPC می‌پردازیم. FPC بر اساس میزان فازی بودن عضویت‌ها محاسبه می‌شود. این معیار بر مبنای میزان تفکیک و رخداد فازی خوشه‌ها است. فرمول معیار FPC به صورت زیر است:

$$FPC = \frac{\sum_{i=1}^{n} \sum_{j=1}^{c} u_{ij}^2}{n}$$

FPC از جمع مربعات انتمالات نقاط به هر خوشه برای تمام نقاط، به ازای تمام خوشه‌ها محاسبه می‌شود و سپس بر تعداد کل نقاط تقسیم می‌شود. این معیار به این دلیل مفید است که مقادیر بالاتر نشان‌دهنده تفکیک بهتر و رخداد فازی کمتر در خوشه‌ها هستند.

حالا کد را شرح میدهم.

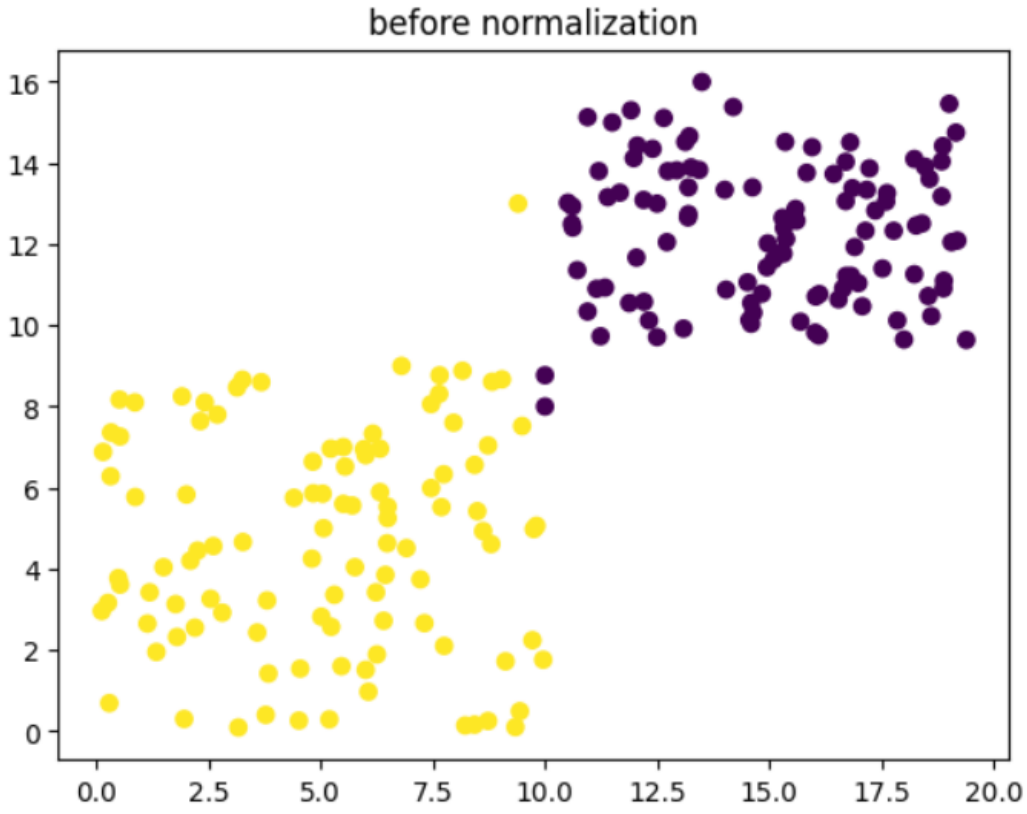
ابتدا کتابخانه ها را ایمپورت میکنیم:

```
:
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import matplotlib.pyplot as plt
import skfuzzy
import numpy as np
```

سپس دیتا ها را میخوانیم. دیتا قبل از نرمال سازی به این صورت است:

```
7]:
df = pd.read_csv('data1.csv')
```

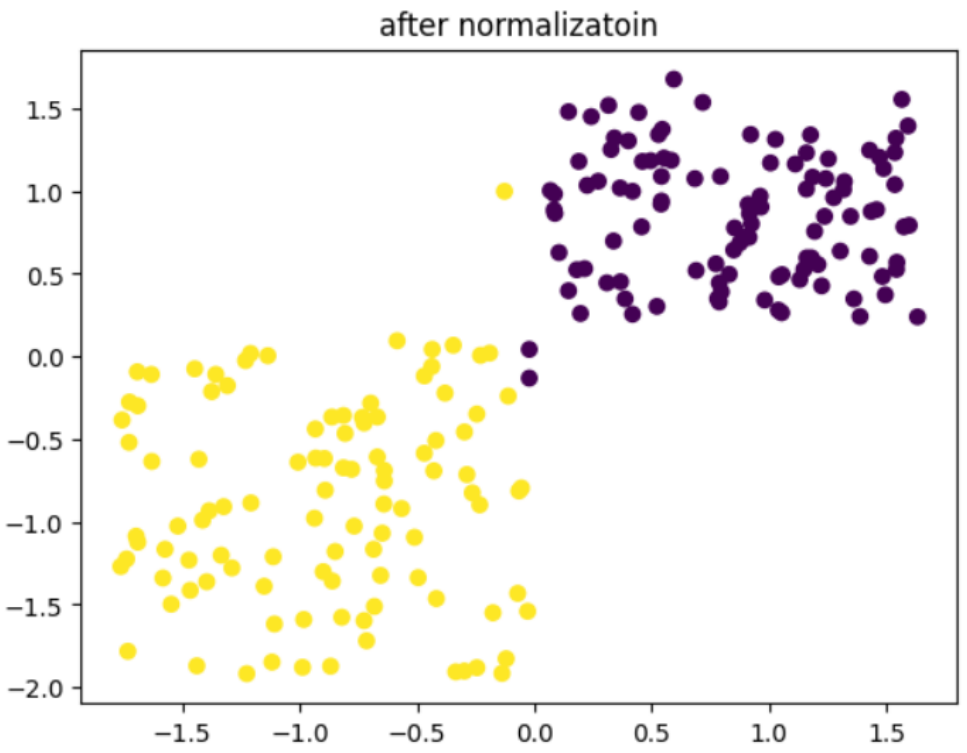
```
8]:
plt.scatter(df['X'], df['Y'], c=df['Class'])
plt.title('before normalization')
plt.show()
```



از نرمال سازی استاندارد استفاده میکنیم و داده ها را به این صورت در میاوریم:

[117]:
```
features = df[['X', 'Y']]
scaler = StandardScaler()
df[['X', 'Y']] = scaler.fit_transform(features)
```

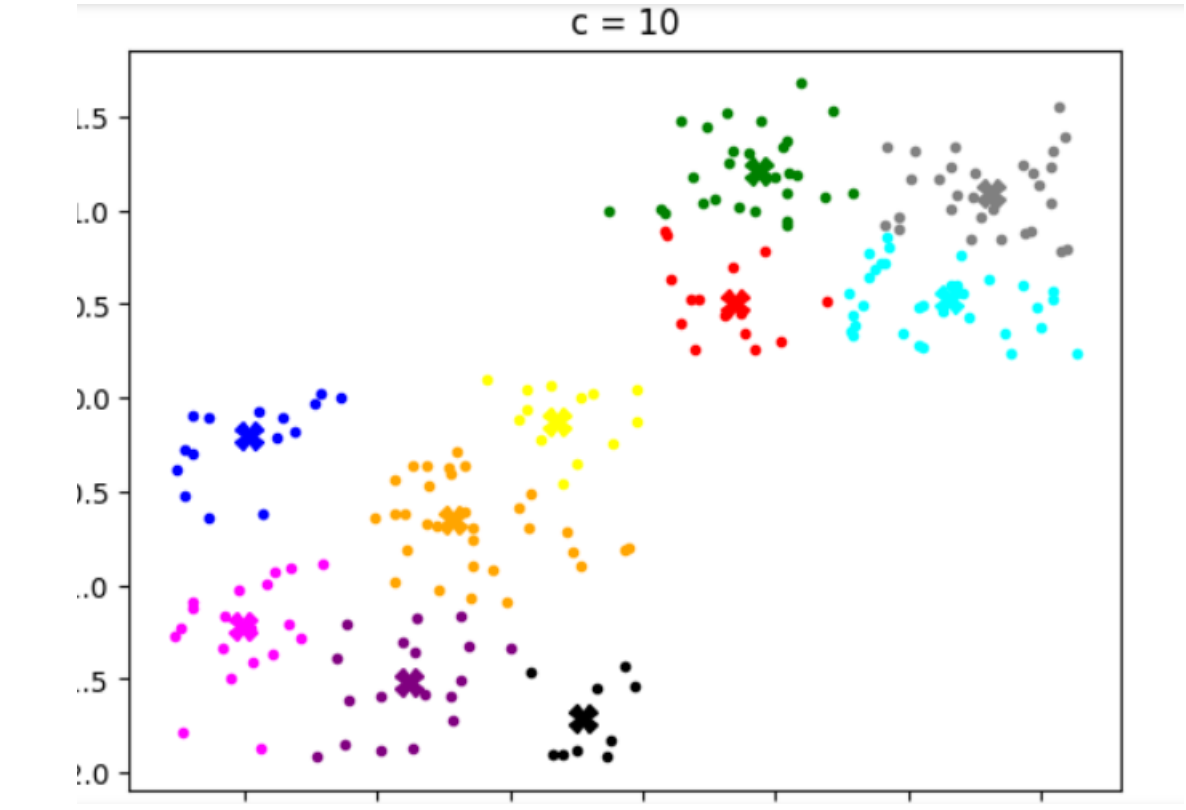
[118]:
```
plt.scatter(df['X'], df['Y'], c=df['Class'])
plt.title('after normalizatoin')
plt.show()
```

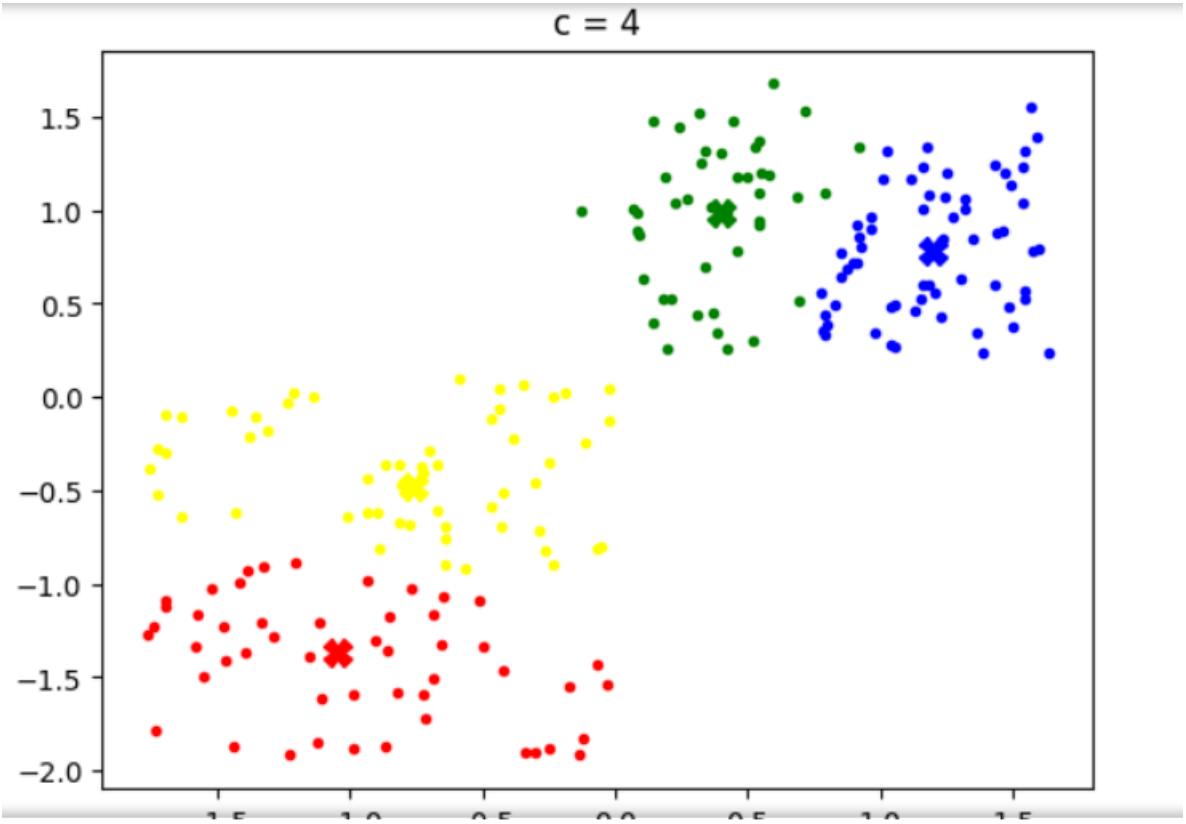


سپس عملیات کلاسترینگ را با C=2 تا 10 انجام میدهیم. کد این قسمت از نمونه کلاس حل تمرین گرفته شده و ویرایش شده. همزمان با محاسبه کلاسترها، مقادیر fpc ها را هم در یک لیست ذخیره میکنیم:

```
fpcs = []
for num_clusters in range(2, 11):
    cntr, u, u0, d, jm, p, fpc = skfuzzy.cluster.cmeans(df[['X', 'Y']].T, num_clusters, 2, error=0.005, maxiter=1000, init=None)
    cluster_membership = np.argmax(u, axis=0)
    fpcs.append(fpc)
```

برای هر c، نتایج خوشه بندی را با رنگ های مختلف چاپ میکنیم. در اینجا دوتا از نمونه ها را نشان میدهم. تمام حالات در فایل نوتبوک چاپ شده.


c = 10

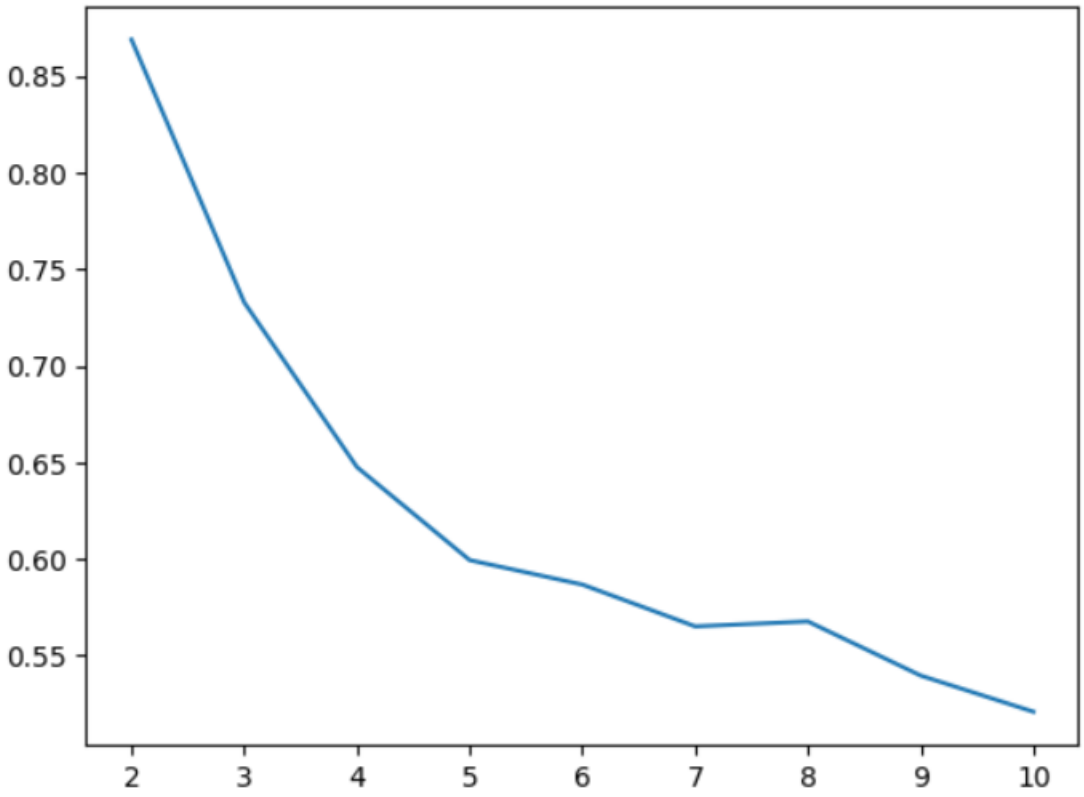

c = 4
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0

سپس نتایج fpc ها را چاپ میکنیم:

Best clustring

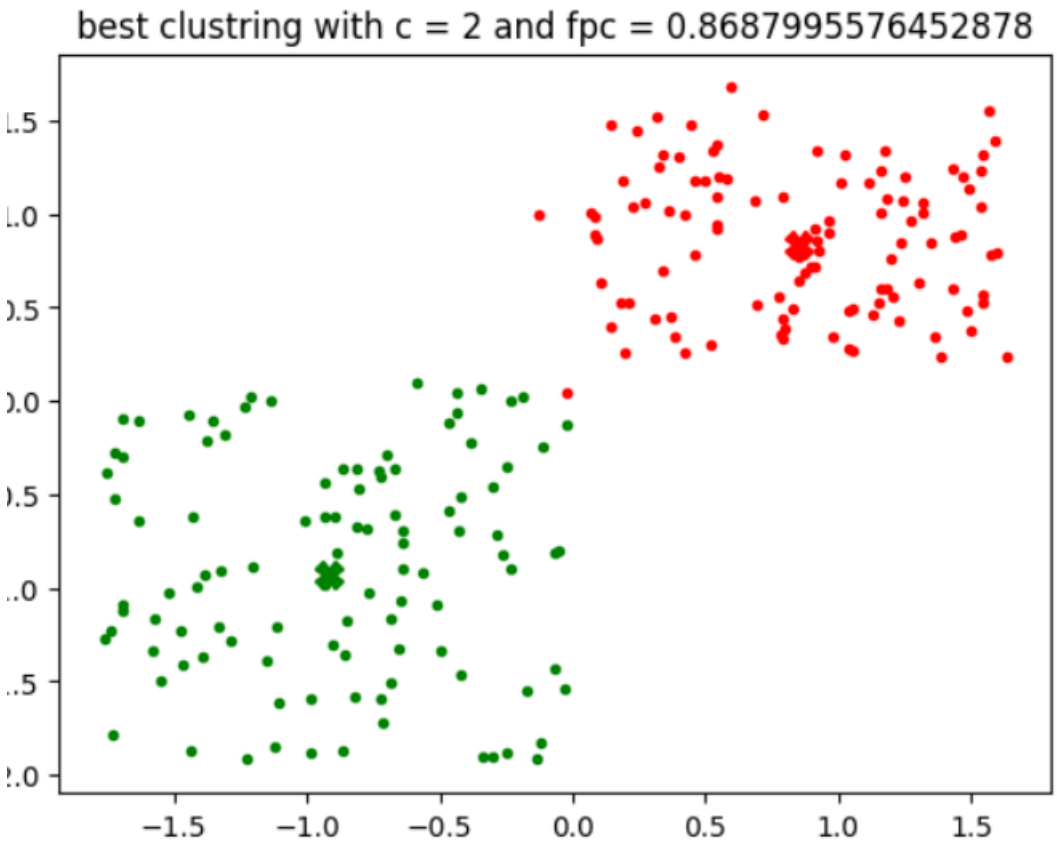
```
plt.plot(range(2, 11), fpcs)
```

```
[<matplotlib.lines.Line2D at 0x26641feea40>]
```



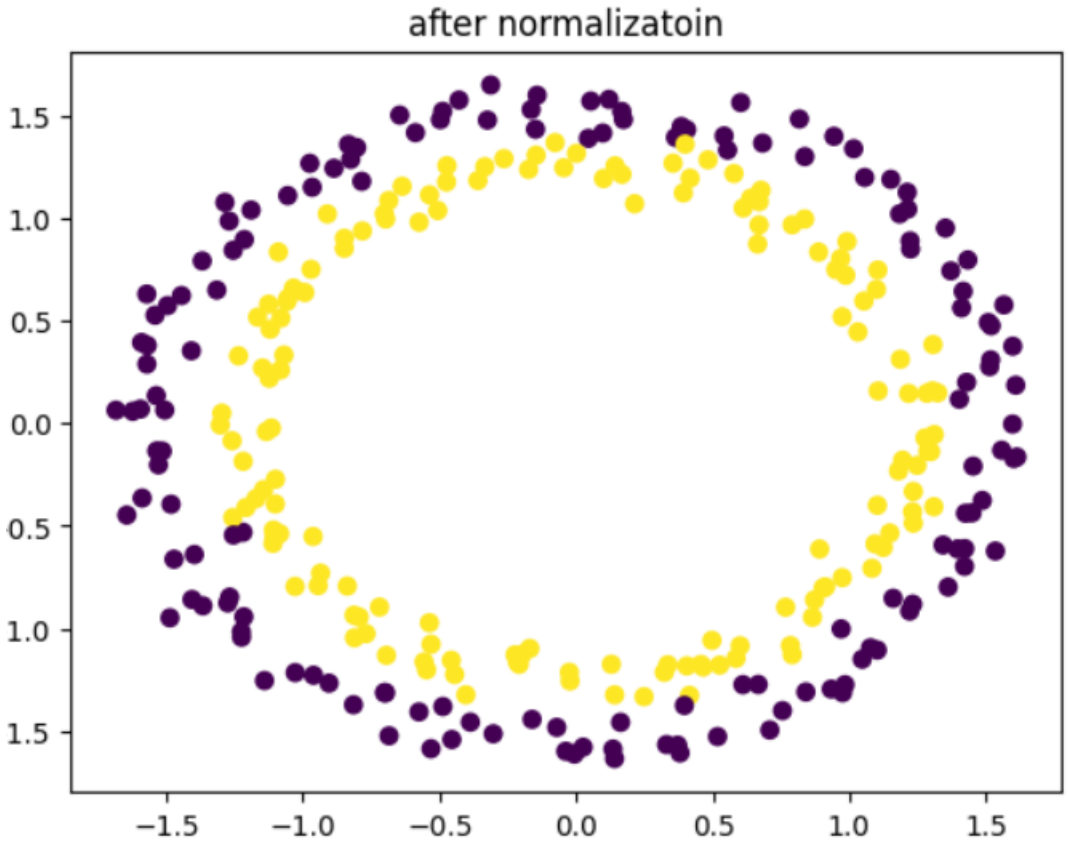
میبینیم که بهترین حالت در c=2 رخ داده.

همانطور که خواسته شده، مجددا نقشه کلاسترینگ را با c=2 رسم میکنیم:



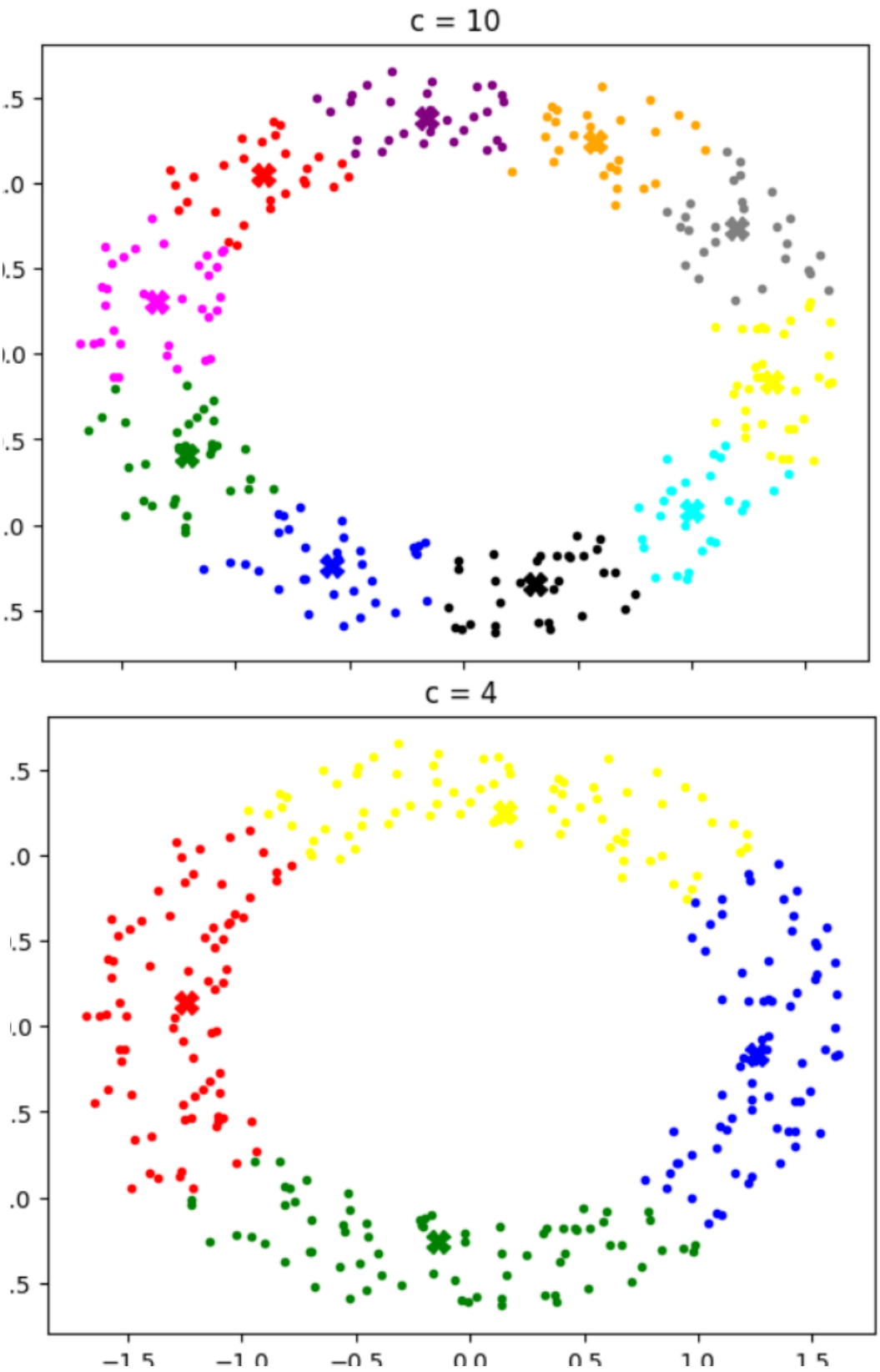
میبینیم که در این مورد، کلاسترینگ به خوبی انجام شده و لیبل‌ها تقریبا مشابه نمونه های دیتاست شده.

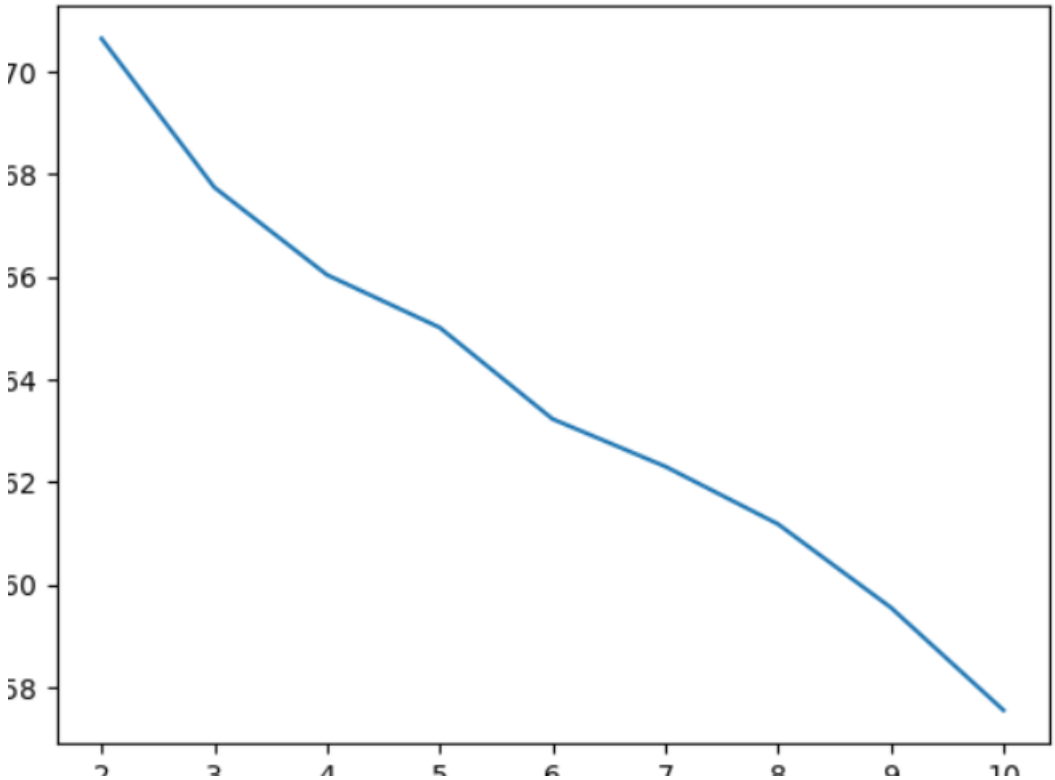
در نهایت همه این کارها را مجددا برای data2 انجام میدهیم:

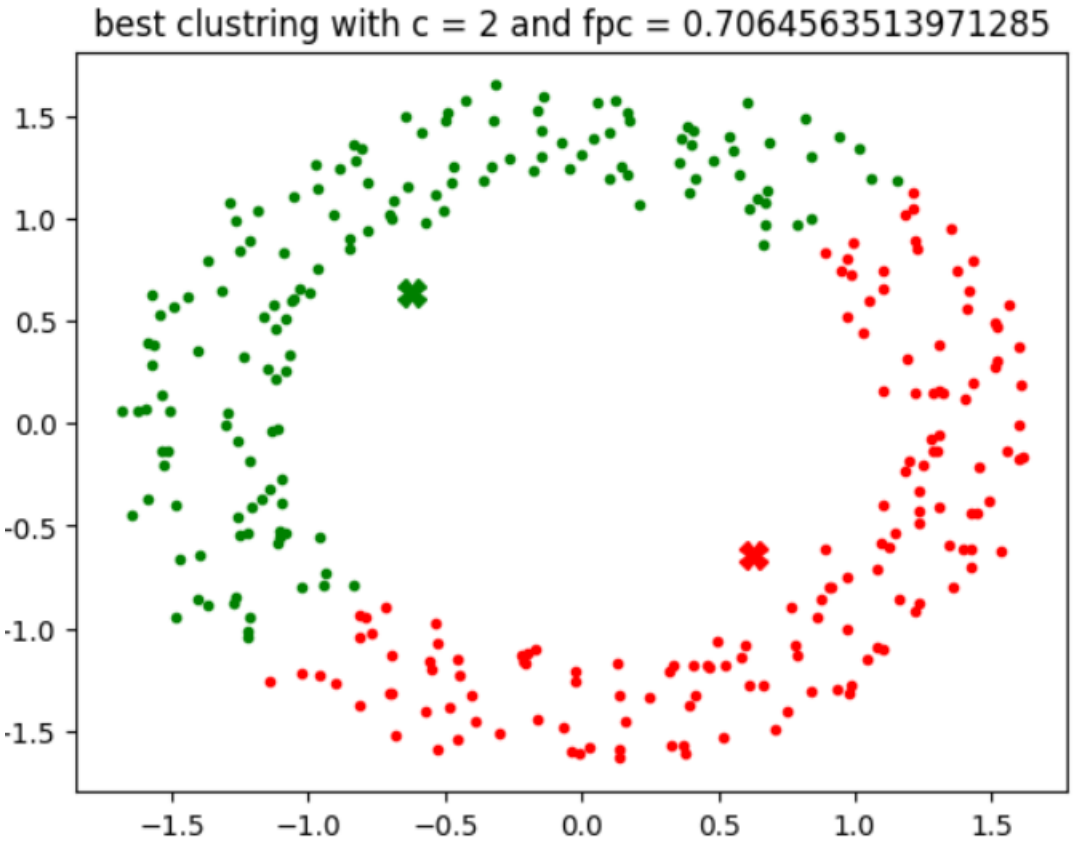


داده های دیتا دوم به خاطر این حالت از کلاس بندی و توی هم قرار گرفتن، نمیتوانند به کمک این الگوریتم به خوبی دسته بندی شوند.

نتایج به این صورت بود:


c = 10
c = 4


70
68
66
64
62
60
58
2
3
4
5
6
7
8
9
10


best clustring with c = 2 and fpc = 0.7064563513971285
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

Q3)

پاسخ سوال 3 در فایل پی دی اف q3.pdf موجود است.