به نام خدا

درس: مبانی یادگیری عمیق

تمرین سری پنجم

مدرس: استاد داودآبادی

سینا علی‌نژاد

۹۹۵۲۱۴۶۹

سوال ۱-

تعریف متغیرهای زبانی و نحوه‌ی بازه بندی آن ها:

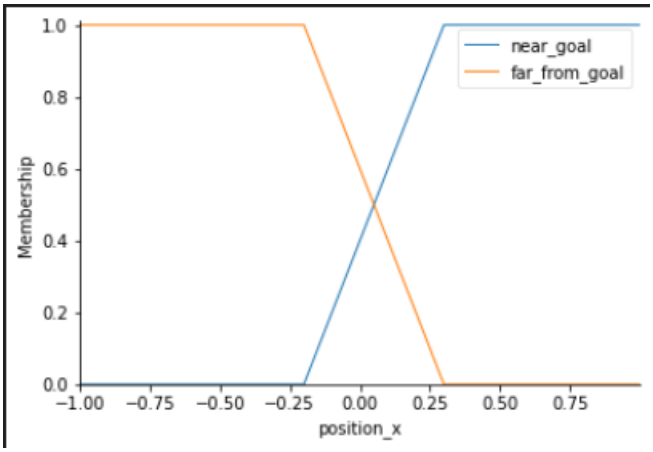
متغیرهای زبانی‌ای که تعریف کردم شامل position_x, freq_velocity, torque میباشند.

متغیر زبانی position_x را با دو مقدار near_goal و far_from_goal تعریف کردم. در واقع میخواهم بر اساس موقعیت x تعیین کنم که آیا پاندول به نقطه هدف نزدیک است یا خیر. تابع trapmf برای تعریف یک ذوزنقه به عنوان تابع عضویت بکار میرود.

```
position_x = ctrl.Antecedent(np.arange(-1, 1, 0.001), 'position_x')

position_x['near_goal'] = fuzz.trapmf(position_x.universe, [-0.2, 0.3, 1, 1])
position_x['far_from_goal'] = fuzz.trapmf(position_x.universe, [-1, -1, -0.2, 0.3])

position_x.view()
```



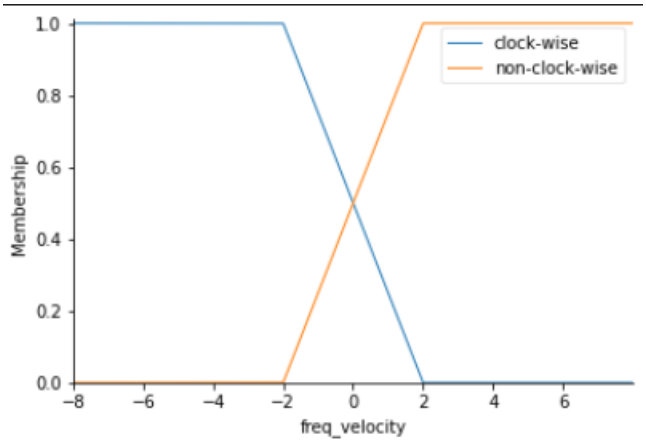
میدانیم هرچه مقدار x در این مسئله به عدد 1 نزدیکتر شود، یعنی به هدف نزدیکتریم. پس ذوزنقه مربوط به near_goal برای اکثر مقادیر منفی، صفر است ولی بین 0.2- تا 0.3 مقدار خودش را به عدد 1 میرساند و تا آخر این مقدار را خواهد داشت. برعکس این موضوع برای far_from_goal میباشد. برای اعداد منفی، مقدار یک و از 0.2- تا 0.3 مقدارش به صفر میرسد و روی این مقدار می‌ماند.

متغیر زبانی freq_velocity یا سرعت زاویه‌ای را با دو مقدار clock-wise و non-clock-wise تعریف کردم. مقدار منفی سرعت زاویه‌ای، یعنی حرکت در جهت عقربه‌های ساعت و مقدار مثبت به معنای حرکت در خلاف جهت عقربه ساعت است.

```
freq_velocity = ctrl.Antecedent(np.arange(-8, 8, 0.001), 'freq_velocity')

freq_velocity['clock-wise'] = fuzz.trapmf(freq_velocity.universe, [-8, -8, -2, 2])
freq_velocity['non-clock-wise'] = fuzz.trapmf(freq_velocity.universe, [-2, 2, 8, 8])

freq_velocity.view()
```



نکته: اینکه تابع عضویت مقادیر یک متغیر زبانی به چه اندازه همپوشانی داشته باشند را با مقادیر مختلف امتحان کردم، هرچه همپوشانی کمتر باشد، سختگیرانه تر عمل میشود.

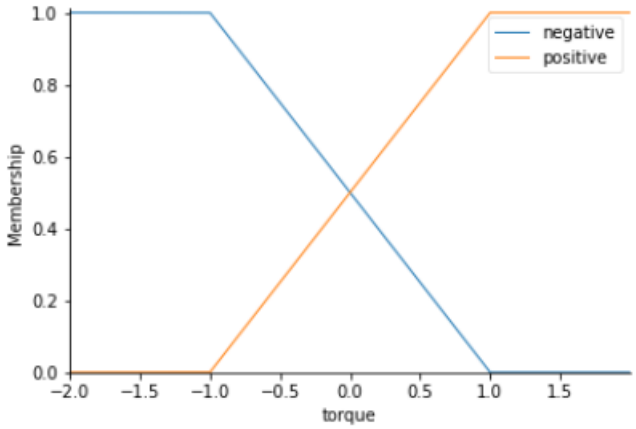
متغیرهای زبانی بالا، ورودیهای ما بودند، حال باید متغیر زبانی خروجی را تعیین کنیم.

متغیر زبانی torque یا گشتاور با دو مقدار positive, negative تعریف می‌شود. همانطور که از اسمشان پیداست، مقادیر مثبت positive و مقادیر منفی negative هستند. مقادیر منفی گشتاور، نیرو را در جهت عقربه‌های ساعت و مقادیر مثبت، نیرو را در جهت خلاف عقربه‌های ساعت وارد می‌کنند.

```
torque = ctrl.Consequent(np.arange(-2, 2, 0.001), 'torque')

torque['negative'] = fuzz.trapmf(torque.universe, [-2, -2, -1, 1])
torque['positive'] = fuzz.trapmf(torque.universe, [-1, 1, 2, 2])

torque.view()
```



برای خروجی، به جای Antecedent از Consequent استفاده می‌کنیم.

نحوه تعریف قوانین:

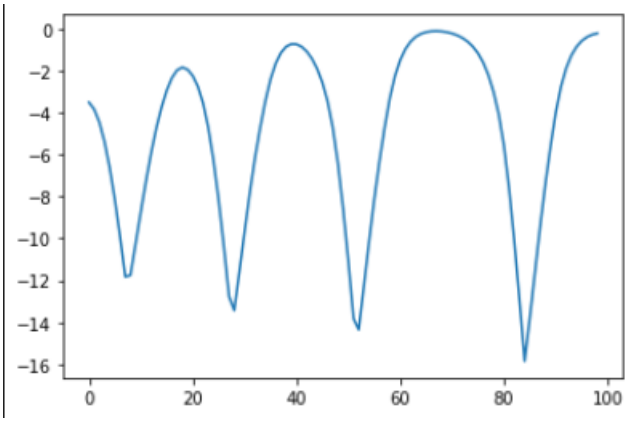
قوانین را با منطقهای زیر تعریف کردم.

منطق ۱: اگر نزدیک هدف بودیم (near_goal)، هر جهتی که سرعت بود، در خلاف آن جهت گشتاور را اعمال میکنیم. یعنی اگر non-clock-wise بود، گشتاور منفی که در جهت عقربه‌های ساعت است اعمال میکنیم و اگر clock-wise بود، گشتاور مثبت که در جهت خلاف عقربه ساعت است، اعمال می‌کنیم.

منطق ۲: اگر از هدف دور بودیم (far_from_goal)، باید خود را به هدف برسانیم، پس باید در جهت سرعتی که داریم، گشتاور اعمال کنیم تا سرعت کافی برای رسیدن به قله را پیدا کند، زیرا شتاب گرانش خود به خود باعث میشود پاندول به سمت پایین سرعت بگیرد. پس اگر سرعت در جهت عقربه ساعت بود، گشتاور منفی و اگر در خلاف جهت عقربه ساعت بود، گشتاور مثبت می‌دهیم.

```
rule1 = ctrl.Rule(position_x['near_goal'] & freq_velocity['non-clock-wise'], torque['negative'])
rule2 = ctrl.Rule(position_x['near_goal'] & freq_velocity['clock-wise'], torque['positive'])
rule3 = ctrl.Rule(position_x['far_from_goal'] & freq_velocity['clock-wise'], torque['negative'])
rule4 = ctrl.Rule(position_x['far_from_goal'] & freq_velocity['non-clock-wise'], torque['positive'])
```

نمودار reward را رسم کردم که به شکل زیر در آمد:

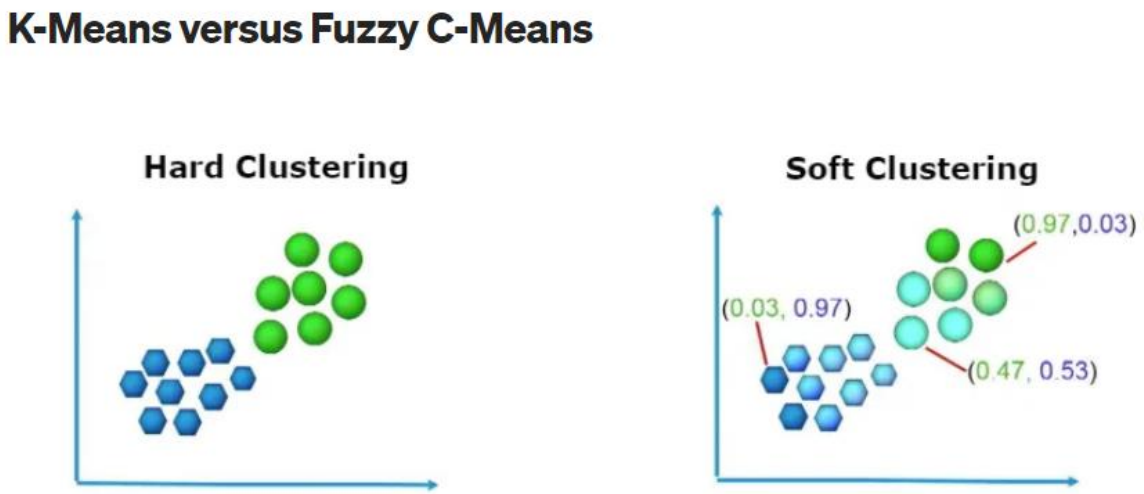


تحلیل نمودار بالا: در قوانینی که برای سیستم فازی خود تعریف کردیم، گفتیم وقتی از هدف دور است، گشتاوری هم‌جهت با سرعت بدهد تا بتواند مقدار بیشتری بالا برود. اما در اوایل بدین شکل است که زور آن به شتاب گرانش نمیرسد و باید در خلاف جهت با ارتفاعی که کسب کرده سرعت بگیرد و این بار از جهت دیگر شانس خودش را برای رسیدن به قله امتحان میکند و البته این بار میتواند ارتفاع بیشتری بگیرد و این رفت و برگشتها باعث میشود بالاخره خود را به قله برساند. پس در هربار تلاش، ارتفاع بیشتری میگیرد و این به معنای نزدیکتر شدن به قله و به معنای reward بزرگتر است. به همین دلیل در این نمودار مشاهده میکنیم که مقدار reward به یک قله میرسد و سپس کاهش پیدا میکند و وقتی از پایین ترین نقطه عبور میکند، دوباره صعودی میشود و دوباره به یک قله میرسد که از قله قبلی بلندتر است و این قله های پشت سر هم ادامه پیدا میکنند تا به قله اصلی برسیم.

سوال ۲-

الف) از اصول منطق فازی می توان برای خوشه بندی داده های چند بعدی استفاده کرد و به هر نقطه از دیتاست یک عدد عضویت در هر مرکز خوشه از ۰ تا ۱۰۰ درصد اختصاص داد. این می تواند در مقایسه با traditional K-means clustering که در آن به هر نقطه یک برچسب دقیق و hard اختصاص داده می شود بسیار قدرتمند باشد. این الگوریتم با اختصاص عضویت به هر نقطه داده مربوط به هر مرکز خوشه بر اساس فاصله بین مرکز خوشه و نقطه داده کار می کند. هر چه داده ها به مرکز خوشه نزدیک باشند، عضویت آن در مرکز خوشه ای خاص بیشتر است. واضح است که مجموع عضویت هر نقطه داده در تمام خوشه‌ها باید برابر با یک باشد.

این یک الگوریتم خوشه‌بندی بدون نظارت است که به ما امکان می‌دهد یک پارتیشن فازی از داده‌ها بسازیم. الگوریتم به پارامتر m بستگی دارد که با درجه مبهم بودن جواب مطابقت دارد. مقادیر بزرگ m کلاس ها را محو می کند و همه عناصر به همه خوشه ها تعلق دارند. راه حل های مسئله بهینه سازی به پارامتر m بستگی دارد. یعنی انتخاب های مختلف m معمولاً منجر به پارتیشن های متفاوتی می شود. تاثیر انتخاب m به صورت یک گیف در لینکی که در صورت سوال داده شد، نشان داده شده است. هرچه m کوچکتر است، الگوریتم به traditional K-means نزدیکتر میشود یعنی برای یک نقطه، به یک کلاس یا خوشه، مقدار عضویت بزرگ میدهد و به بقیه مقادیر کمتر و سختگیرانه تر عمل میکند اما با افزایش m تقریبا به همه کلاسها مقادیر مساوی میدهد.



عکس بالا تفاوت hard clustering و soft clustering را به طور واضحی نشان میدهد. نقاطی که تقریبا در تلاقی دو خوشه قرار دارند، مقادیر نرمتری دارند و تقریبا عضو هردو کلاس یا خوشه هستند.

حال باید این دو الگوریتم قدرتمند را با هم مقایسه کنیم تا متوجه شویم چه زمانی باید از fuzzy C-means استفاده کنیم.

انتساب به یک خوشه: در خوشه بندی فازی، هر نقطه احتمال تعلق به هر خوشه را دارد، نه اینکه به طور کامل فقط به یک خوشه تعلق داشته باشد، همانطور که در k-means سنتی وجود دارد. در خوشه‌بندی Fuzzy-C Means، هر نقطه دارای وزنی است که با یک خوشه خاص مرتبط است، بنابراین در اینجا قرارگیری یک نقطه در یک خوشه تعیین نمیشود بلکه قوی یا ضعیف بودن ارتباط آن نقطه با خوشه تعیین میشود.

سرعت: fuzzy C-means نسبت به traditional K-means کندتر عمل میکند، زیرا در واقع کار بیشتری انجام می دهد. هر نقطه با هر خوشه ارزیابی می شود و عملیات بیشتری در هر ارزیابی انجام می شود. K-Means فقط باید یک محاسبه فاصله انجام دهد، در حالی که میانگین c فازی باید یک وزن دهی کامل با فاصله معکوس انجام دهد.

Fuzzy C-means یک مورد خاص از K-means است وقتی که تابع احتمال مورد استفاده به 1 باشد اگر نقطه داده به یک مرکز خوشه نزدیکتر باشد و 0 باشد اگر نقطه داده به یک مرکز خوشه نزدیکترین نباشد.

Steps in Fuzzy C-Means

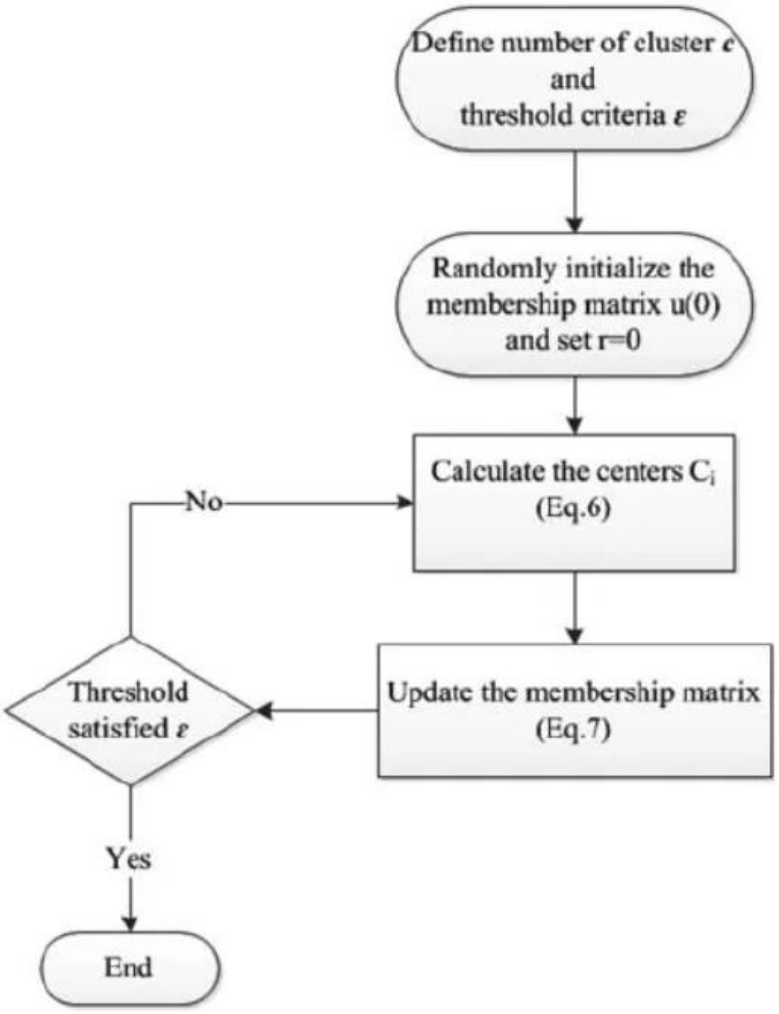


Image Credits: https://www.researchgate.net

همانطور که در دیاگرام بالا مشخص است، ابتدا باید تعداد خوشه و یک threshold برای تعیین اتمام کار الگوریتم مشخص کنیم. سپس ماتریس عضویت را به صورت رندوم مقداردهی میکنیم. برای مثال اگر تعداد خوشه برابر با 5 باشد و تعداد نقاط برابر با 1000 باشد، ابعاد این ماتریس برابراست با: 1000*5 که برای هر نقطه در هر خوشه یک مقدار عضویت اولیه به صورت رندوم تعیین میشود. مراکز هر خوشه محاسبه میشوند تا به کمک این مراکز جدید، یک مرحله دیگر، ماتریس عضویت را بروزرسانی کنیم. پس از بروزرسانی ماتریس عضویت به کمک فرمول زیر میتوان مراکز خوشه جدید را به صورت مرکز وزندار از مقادیر عضویت هر نقطه بدست آورد.

$$\mu_k(n+1) = \frac{\sum_{x_i \epsilon k} x_i * P(\mu_k|x_i)^b}{\sum_{x_i \epsilon k} P(\mu_k|x_i)^b}$$

اتمام الگوریتم نیز بدین صورت است که یا به تعداد iteration مشخص که توسط کاربر گفته شده، ادامه میدهیم، یا تا همگرایی ادامه میدهیم.

حال به پیاده سازی این الگوریتم میپردازیم. نیازی به پیاده سازی ندارد زیرا تابع آماده برای آن وجود دارد.

به معرفی این تابع می‌پردازیم:

```
cntr, u, u0, d, jm, p, fpc = skfuzzy.cluster.cmeans(final_inp, num_clusters, 2, error=0.005, maxiter=1000, init=None)
```

معرفی پارامترهای ورودی:

پارامتر اول: data

data : 2d array, size (S, N)

> Data to be clustered. N is the number of data sets; S is the number of features within each sample vector.

ورودی اول دیتایی است که میخواهیم روی آن خوشه‌بندی را انجام دهیم. ابعاد آن به شکل (S,N) است، بعد اول تعداد ویژگیها و بعد دوم تعداد نقاط یا رکورد دیتاست است.

پارامتر دوم: c

این پارامتر تعداد خوشه ها را مشخص میکند. این مقدار با آزمون و خطا بدست می‌آید.

c : int

> Desired number of clusters or classes.

پارامتر سوم: m

درباره مقدار m در بخش الف صحبت کردیم. هرچه m بزرگتر شود، نقاط در همه خوشه‌ها به یک میزان عضویت خواهند داشت و این مقادیر به هم نزدیک خواهند شد.

m : float

> Array exponentiation applied to the membership function u_old at each iteration, where U_new = u_old ** m.

پارامتر چهارم: error

error : float

> Stopping criterion; stop early if the norm of (u[p] - u[p-1]) < error.

این پارامتر را تعیین میکنیم تا اگر مقدار ارور از حدی کمتر شد، الگوریتم متوقف شود. به نوعی به عنوان stopping criterion است.

پارامتر پنجم: maxiter

maxiter : int

> Maximum number of iterations allowed.

این پارامتر تعیین میکند که الگوریتم تا یک تعداد iteration مشخص ادامه پیدا کند. (هر iteration در دیاگرام بالاتر نشان داده شده است)

مقادیری که توسط این تابع به ما برگردانده میشوند:

اول: cntr

cntr : 2d array, size (S, c)

> Cluster centers. Data for each center along each feature provided for every cluster (of the c requested clusters).

مراکز نهایی هر خوشه را برمیگرداند. ابعاد آن (S, c) است که S تعداد ویژگیها و c تعداد خوشه ها است.

دوم: u

u : 2d array, (S, N)

> Final fuzzy c-partitioned matrix.

مقادیر نهایی ماتریس عضویت میباشد که میتوان با استفاده از تابع np.argmax بر روی آن یک خوشه نهایی برای هر نقطه تعیین کرد.

سوم تا ششم:

u0 : 2d array, (S, N)

> Initial guess at fuzzy c-partitioned matrix (either provided init or random guess used if init was not provided).

d : 2d array, (S, N)

> Final Euclidian distance matrix.

jm : 1d array, length P

> Objective function history.

p : int

> Number of iterations run.

هفتم: fpc

این مقدار مشخص میکند که خوشه بندی انجام شده تا چه حد خوب صورت گرفته است. هرچه این مقدار بزرگتر باشد، یعنی خوشه بندی بهتری صورت گرفته است. تا جایی که من میدانم، بر اساس واریانس هر خوشه کار میکند. مجموع واریانس خوشه ها هرچه کمتر باشد بهتر است.

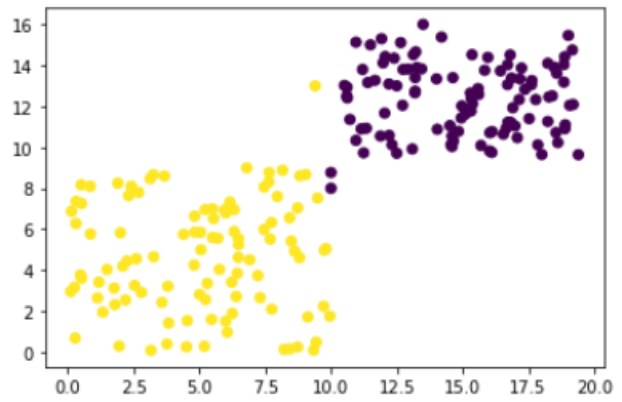
fpc : float

> Final fuzzy partition coefficient.

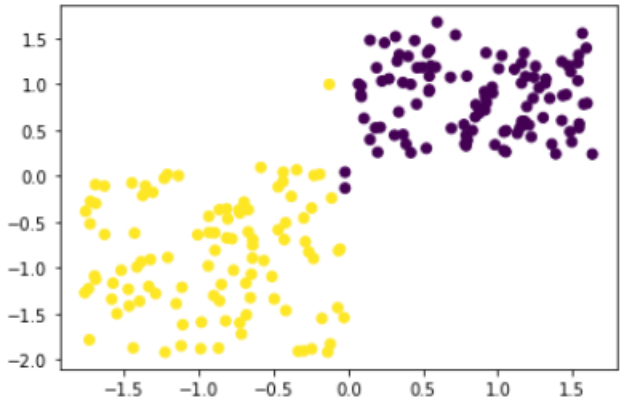
تحلیل نتایج:

Data1.csv: کد خوشه بندی برای این دیتاست در فایل q2_data1 قرار دارد.

توزیع اولیه داده ها:



توزیع داده ها پس از normalization به کمک StandardScaler:



پس از این نوع نرمالسازی، میانگین داده ها صفر و واریانس برابر با ۱ میشود.

کد نرمالسازی:

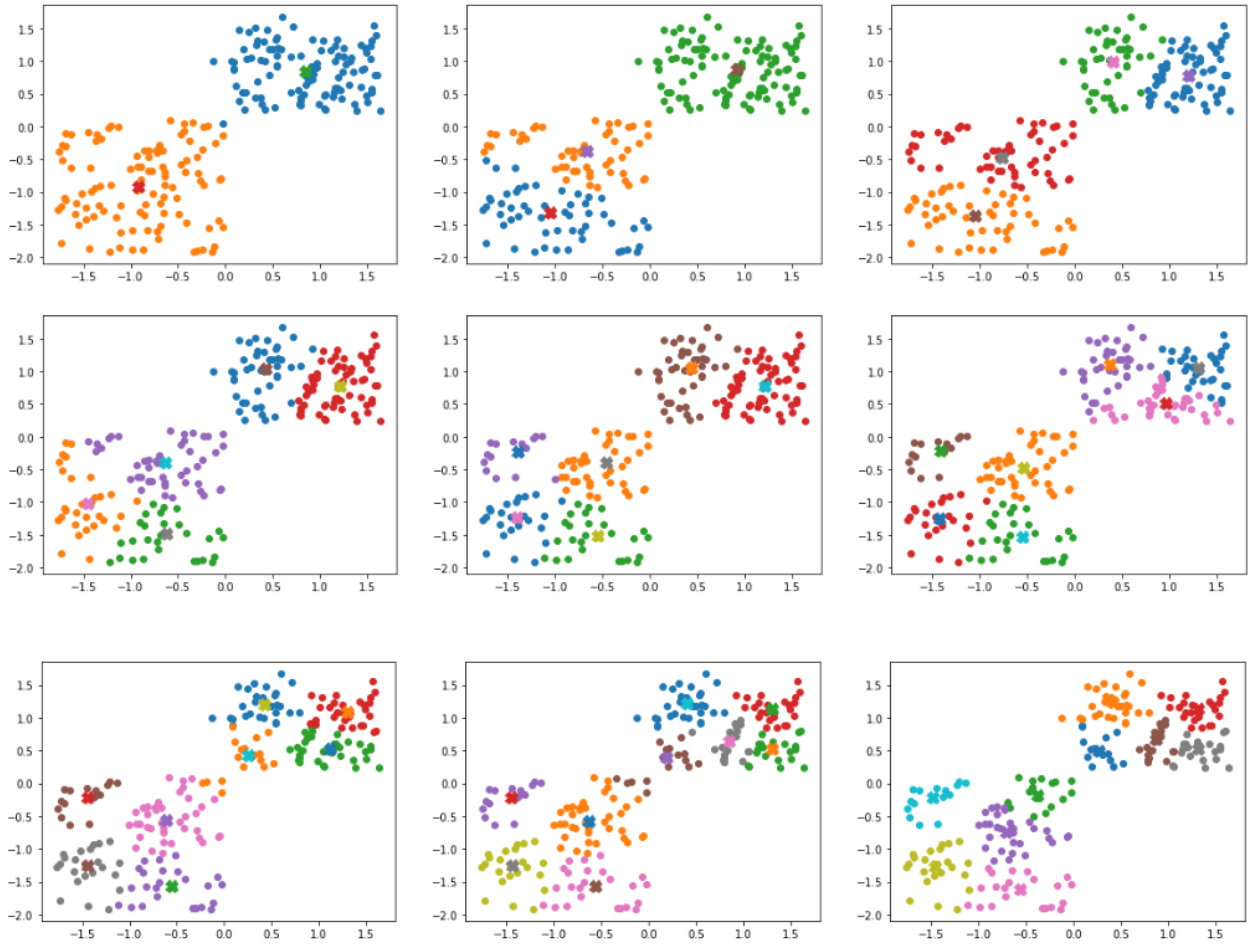
```
# Reshape the array to a 2D array
x = x.reshape(-1, 1)
y = y.reshape(-1, 1)

# Create a StandardScaler object
scaler = StandardScaler()

# Fit the scaler to the data
scaler.fit(x)
x_normal = scaler.transform(x)
scaler.fit(y)
y_normal = scaler.transform(y)

x_normal = np.ravel(x_normal)
y_normal = np.ravel(y_normal)
```

آزمایش با تعداد خوشه های ۲ تا ۱۰ انجام شد و نتیجه هر کدام به شکل زیر بود:



به صورت چشمی انتظار داریم، تعداد مناسب خوشه ها برابر با ۲ باشد. باید مقادیر fpc را مقایسه کنیم. هر کدام که مقدار بزرگتری از fpc داشت، به عنوان بهترین تعداد خوشه انتخاب میشود.

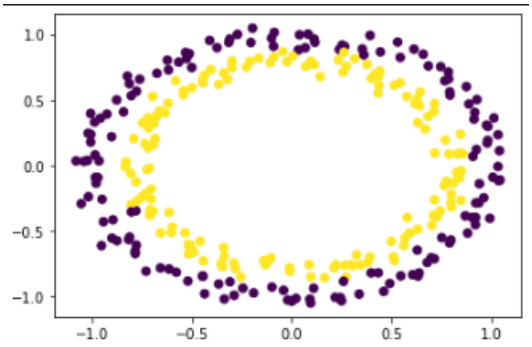
```
max_ind = np.argmax(np.array(fpc_s))
best_num_cluster = 2 + max_ind
best_fpc = fpc_s[max_ind]
print(f"best fpc is {best_fpc} which belongs to number of clusters equal to {best_num_cluster}")
```

```
best fpc is 0.8687990750112378 which belongs to number of clusters equal to 2
```

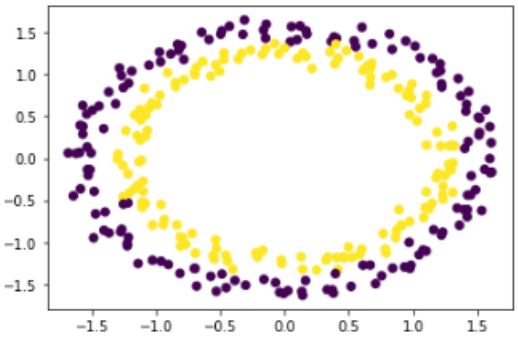
FPC در محدوده ۰ تا ۱ تعریف شده است که ۱ بهترین است. این معیاری است که به ما می گوید داده های ما با یک مدل خاص چقدر تمیز توصیف می شوند.

مراحل بالا را برای data2.csv در فایل q2_data2 انجام دادم و به نتایج زیر رسیدم:

توزیع اولیه:



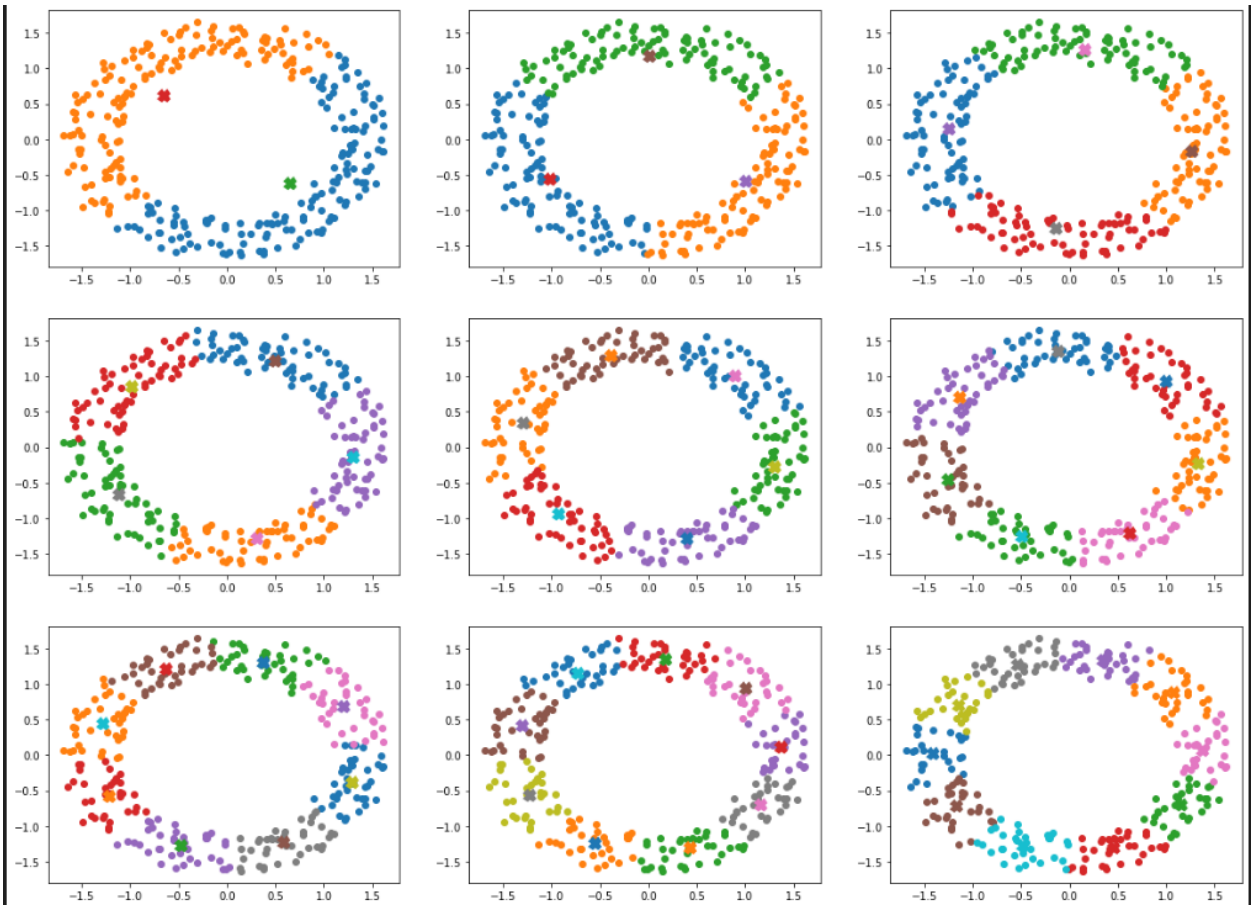
توزیع پس از نرمالسازی:



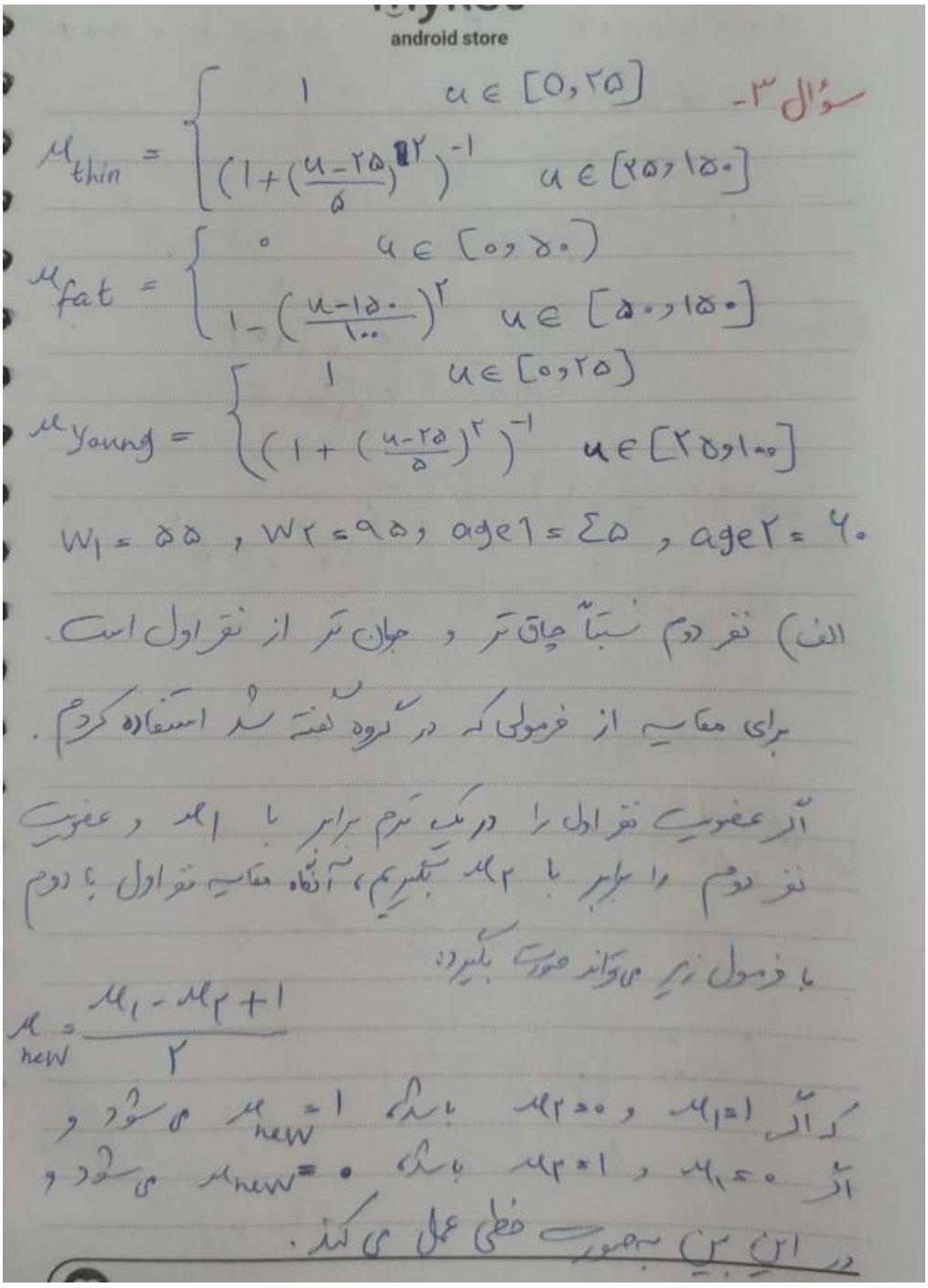
```
    max_ind = np.argmax(np.array(fpc_s))
    best_num_cluster = 2 + max_ind
    best_fpc = fpc_s[max_ind]
    print(f"best fpc is {best_fpc} which belongs to number of clusters equal to {best_num_cluster}")
```

```
best fpc is 0.7064717222211385 which belongs to number of clusters equal to 2
```

خوشه‌بندی نهایی برای تعداد خوشه ۲ تا ۱۰:



سوال ۳-



سؤال ۳-

$$\mu_{thin} = \begin{cases} 1 & u \in [0,25] \\ \left(1+\left(\frac{u-25}{5}\right)^{2}\right)^{-1} & u \in [25,150] \end{cases}$$

$$\mu_{fat} = \begin{cases} 0 & u \in [0,80) \\ 1-\left(\frac{u-150}{100}\right)^{2} & u \in [80,150] \end{cases}$$

$$\mu_{young} = \begin{cases} 1 & u \in [0,25] \\ \left(1+\left(\frac{u-25}{5}\right)^{2}\right)^{-1} & u \in [25,100] \end{cases}$$

$W_1 = 55$, $W_2 = 95$, $age1 = 45$, $age2 = 40$

الف) نفر دوم نسبتاً چاق تر و جوان تر از نفر اول است.

برای مقایسه از فرمولی که در گروه گفته شد استفاده کردم.

اگر عضویت نفر اول را در یک ترم برابر با μ_1 و عضویت نفر دوم را برابر با μ_2 بگیریم، آنگاه مقایسه نفر اول با دوم با فرمول زیر می‌تواند صورت بگیرد:

$$\mu_{new} = \frac{\mu_1 - \mu_2 + 1}{2}$$

که اگر $\mu_1 = 1$ و $\mu_2 = 0$ باشد $\mu_{new} = 1$ می‌شود و اگر $\mu_1 = 0$ و $\mu_2 = 1$ باشد $\mu_{new} = 0$ می‌شود، در این بین بصورت خطی عمل می‌کند.

- نفر اول چاق است :

$\mu_{fat}(1) = 0.97$

- نفر دوم چاق است : $\mu_{fat}(2) = 0.7$

- نفر دوم چاق‌تر از نفر اول است :

$\mu_{\text{چاق‌تر}} = \frac{\mu_{fat}(2) - \mu_{fat}(1) + 1}{2} = 0.8$

- نفر دوم نسبتاً چاق‌تر از نفر اول است :

* برای modifier «نسبتاً» از رادیکال‌گیری استفاده می‌کنیم.

$\mu_{\text{نسبتاً چاق‌تر}} = \sqrt{0.8} = 0.89$

- نفر اول جوان است : $\mu_{young}(1) = 0.05$

- نفر دوم جوان است : $\mu_{young}(2) = 0.02$

- نفر دوم جوانتر از نفر اول است:

$\mu_2 = 0.3$, $\mu_1 = 0.05$

$\mu_{\text{جوانتر}} = \frac{\mu_2 - \mu_1 + 1}{2} = \frac{0.2 - 0.05 + 1}{2} = 0.475$

برای And کردن از فرمول min استفاده می‌کنم.

- نفر دوم نسبتاً جاق‌تر و جوانتر از نفر اول است:

$\mu_{\text{نسبتاً جاق‌تر و جوانتر}} = \min(\mu_{\text{نسبتاً جاق‌تر}}, \mu_{\text{جوانتر}})$

$= \min(0.89, 0.475) = 0.475$

پس مورد (الف) حل شد.

ب) اگر نفر اول خیلی لاغر باشد، آنگاه نفر دوم نسبتاً جوان است.

فرض: برای implication از فرمول $V(A \Rightarrow B) = \max(1 - V(a), V(B))$ استفاده می‌کنم.

- نفر اول خیلی لاغر است:

$\mu_{thin}(1) = 0.27$

* برای modifier «خیلی» از توان دوم استفاده می‌کنم.

$\mu_{very\ thin}(1) = (0.027)^2 = 7 \times 10^{-4}$

- نفر دوم [redacted] است.
نسبتاً جوان

$\mu_{young}(2) = 0.02$

* برای modifier «نسبتاً» از رادیکال گیری استفاده می‌کنم.

$\mu_{\text{نسبتاً } young}(2) = \sqrt{0.02} = 0.14$

- اگر نفر اول خیلی لاغر باشد، آنگاه نفر دوم نسبتاً جوان است.

$\mu_{final} = \max\left(1 - \mu_{very\ thin}(1),\ \mu_{\text{نسبتاً } young}(2)\right) =$

$\max(1 - 0.0007,\ 0.14) = 0.14$

پس مورد (ب) نیز حل شد.