تمرین‌های سری پنج

درس یادگیری ماشین

فرهاد دلیرانی

۹۶۱۳۱۱۲۵

dalirani@aut.ac.ir

dalirani.1373@gmail.com

از آن‌جایی‌که در تمرین‌ها به ورژن خاصی از پایتون اشاره نشده است، تمرین‌ها را با پایتون ۳٫۶ نوشتم. همین‌طور از پکیج‌های زیر استفاده کردم:

- Pandas جهت خواندن فایل‌های ورودی
- Matplotlib جهت رسم نمودار
- Numpy جهت محاسبات بر روی آرایه‌ها

سوال ۱)

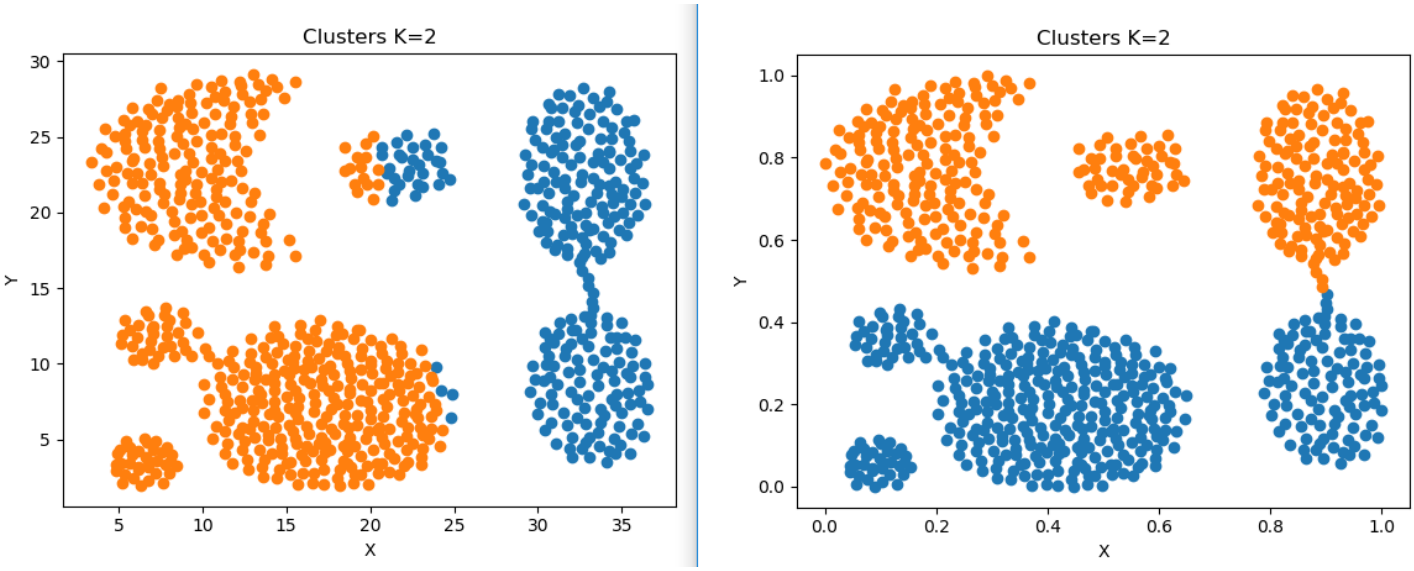
Median برابر با وسط دیتاهای مرتب شده است. میانه الزاما بخشی از داده‌ها نیست به طور مثال وقتی تعداد داده‌ها زوج است از دو مقدار وسط میانگین می‌گیریم. هنگام استفاده از میانگین کلاسترها متمایل به دایره (ابر دایره) شدن دارند. ولی هنگام استفاده از میانه کلاسترها فشرده‌تر هستند و بیشتر داده‌ها به مرکز بسیار نزدیک هستند و تعدادی کمی از داده‌ها از مرکز فاصله دارند. از آن جایی که میانه وسط داده‌های مرتب شده است تاثیر داده‌های پرت بر آن بسیار کمتر از میانگین است و در برابر داده‌های پرت Robust است. یکی از مزایای استفاده از میانگین نسبت به میانه این است که همه‌ی داده‌ها برای تعیین Centroid مورد استفاده قرار می‌گیرد.

در روش Modes از Mode هر Dimension برای ایجاد Centroid استفاده می‌شود. Mode هر Dimension برابر با مقداری است که به تعداد بیشتری رخ داده است. که برای داده‌های Categorical مناسب است. در حالی که میانگین از داده‌های حقیقی استفاده می‌کند و در صورتی که داده‌ها Categorical باشند با توجه به اینکه از نوع Nominal و یا Ordinal هستند باید آن‌ها را به عدد مناسب تبدیل کنیم. در میانگین همه‌ی داده‌ها به یک اندازه وزن داده می‌شوند در حالی که در Modes داده‌ای که بیشتر رخ داده وزن داده می‌شود. خوشه‌ها در Modes فشرده‌تر از روش میانگین هستند.

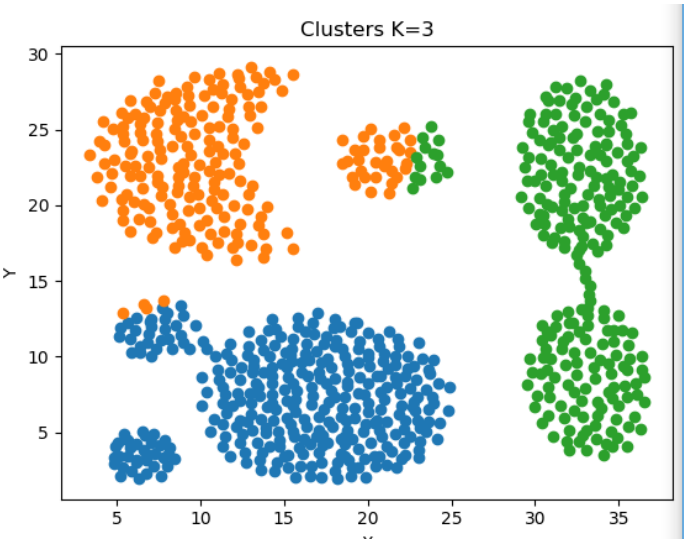
سوال ۲)

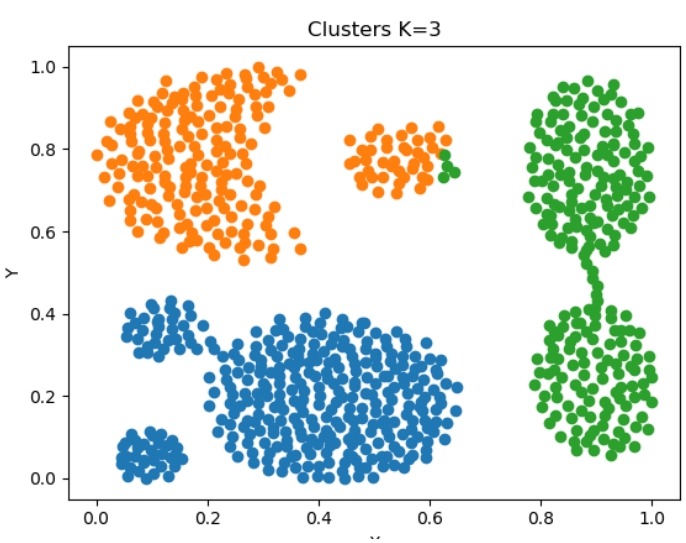
در problem2.py کدی است که در آن از تابع‌های کدی‌هایی که برای پاسخ به سوال‌های پنج و شش نوشته‌ام استفاده کرده‌ام، به همین دلیل آن تابع‌ها را در این سوال توضیح نمی‌دهم و در سوال ۵ و ۶ به آن‌ها می‌پردازم، در این سوال یک بار داده‌های فایل data2 را برای خوشه‌بندی به تابع k-means که نوشته‌ام می‌دهم و بار بعد آن داده‌ها را نرمال‌سازی می‌کنم و بعد آن‌ها را خوشه بندی می‌کنم. در زیر نتایج را مشاهده می‌کنید(برای ایجاد شرایط یکسان و مقایسه کردن در هر دو حالت داده‌های یکسانی به عنوان Centroidها انتخاب شده‌اند):

برای K=2



برای K=3





همین‌طور که مشاهده می‌شود خوشه‌ها در دو حالت متفاوت هستند. دلیل این موضوع آن است که K-means بر اساس فاصله اقلیدسی است و بر اساس این فاصله خوشه‌ها را تشکیل می‌دهد. وقتی نرمال‌سازی می‌کنیم فیچرهای مختلف با مقدارهای متفاوتی Scale می‌شوند و همه‌ی فیچرها با یک عدد یکسان Scale نمی‌شوند به همین دلیل فاصله‌ها تغییر می‌کند و خوشه‌های متفاوتی به دست می‌آید.

سوال ۳)

در الگوریتم Single Link برای سنجیدن فاصله‌ی بین دو کلاستر فاصله‌ی بین دو نقطه از دو کلاستر را در نظر می‌گیرند به طوری که فاصله‌ی بین دو کلاستر حداقل شود، این معیار سایر نقاط درون دو کلاسترها را در نظر نمی‌گیرد. اگر نقطه‌های دیتاست را در الگوریتم Single Link بر اساس ترتیبی که در الگوریتم انتخاب و با هم merge می‌شوند وصل کنیم یک درخت Minimum Spanning Tree به دست می‌آید.

در الگوریتم Complete Link برای سنجیدن فاصله‌ی بین دو کلاستر فاصله‌ی بین دو نقطه از دو کلاستر را در نظر می‌گیرند به طوری که فاصله‌ی بین دو کلاستر حداکثر شود، این معیار سایر نقاط درون دو کلاسترها را در نظر نمی‌گیرد. در این الگوریتم نسبت به Single Link کلاسترها کم‌تر تمایل به رشد در درازا دارند.

هر دو به Outliers حساس‌اند به همین دلیل برای از بین بردن حساسیت نسبت به داده‌های پرت می‌توان از Average Link استفاده کرد. Complete Link داده‌ها پرت را دیرتر با سایر خوشه‌ها ترکیب می‌کند به همین سبب نسبت به Single Link حساسیت کمتری به داده‌ی پرت دارند.

هر سه الگوریتم سقف پیچدگی زمانی $O(N^2Lg(N))$ دارند که البته می‌توان Single Link را بهبود داد و به سقف پیچیدگی زمانی $O(N^2)$ رساند.

حد بالای هزینه‌ی محاسباتی $O(N^2Lg(N))$ اینگونه به دست می‌آید:

$O(n^2)$ برای ساخت ماتریس فاصله‌ها، برای هر داده فاصله‌ی سایر نقطه‌ها به آن نقطه را مرتب کنیم که هزینه‌ی $O(N^2Lg(N))$ را دارد. حداکثر N به‌روزرسانی جدول فاصله‌ها را داریم که هر بار با $o(N)$ قابل انجام است که هزینه‌ی به روزرسانی $o(N^2)$ را دارد. در نتیجه حد بالا برابر با $O(N^2Lg(N))$ می‌شود.

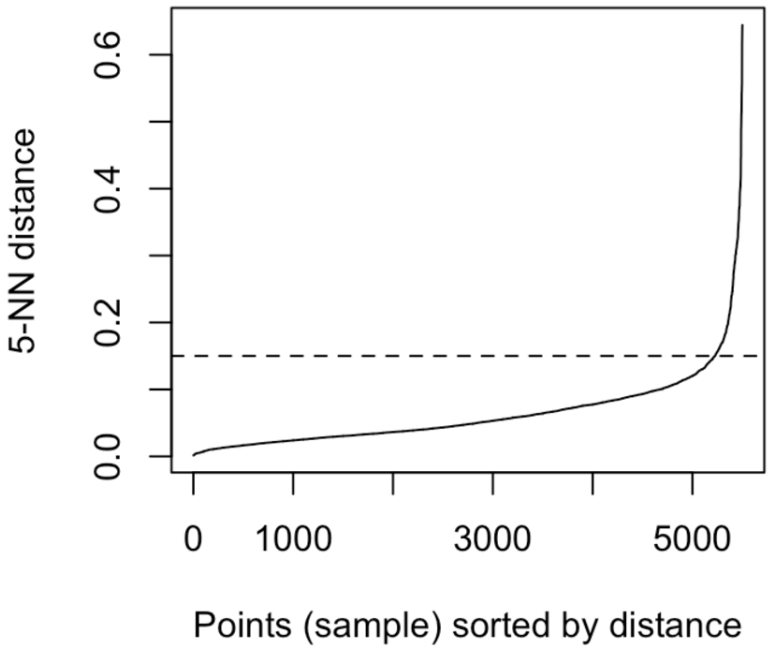
سوال ۴)

دو معیاری که نقش مهمی در تعیین خروجی دارند معیارهای زیر هستند:

- EPS: این معیار شعاع دایره‌ی همسایگی را مشخص می‌کند.
- MNP: حداقل نقاط درون دایره که مشخص می‌کند آن نقطه‌ی مرکز دایره Core است یا boundary که در هنگام تشکیل کلاستر استفاده می‌شود.

روش‌های مختلف برای تعیین این دو مقدار موجود است. در زیر روشی که در مقاله‌ی DBSCAN ذکر شده است را توضیح می‌دهم:

ابتدا یک مقدار برای MNP انتخاب می‌کنیم و بعد K Nearest Neighbors هر نقطه در دیتاست را پیدا می‌کنیم که K را برابر با MNP قرار می‌دهیم. بعد از پیدا کردن K همسایه‌ی نزدیک به هر نقطه، میانگین فاصله‌ی نقطه از K همسایه‌اش را به دست می‌آوریم. سپس فاصله‌های به دست آمده را به صورت صعودی مرتب می‌کنیم و نمودار آن را رسم می‌کنیم. تصویر زیر را به عنوان مثال در نظر بگیرید که MNP و K برابر با ۵ در نظر گرفته شده‌اند و بعد از آن میانگین فاصله‌ی نقطه‌ها تا ۵ نزدیک‌ترین همسایه‌اش به دست آمده است و بعد از مرتب سازی رسم شده اند :



برای انتخاب EPS (شعاع دایره) به نمودار نگاه می‌کنیم و محلی که میزان فاصله به مقدار زیادی نسبت به قبل‌تر تغییر کرده است را پیدا می‌کنیم و از آن نقطه یک خط عمود بر محور عمودی رسم می‌کنیم. محل برخورد آن خط با محور عمودی برابر با EPS مناسب برای کلاسترینگ است. که در تصویر بالا چیزی در حدود ۰,۱۵ است.

اکنون می‌توانیم به ازای MNP مشخص یک EPS مناسب پیدا کنیم. برای پیدا کردم MNP می‌توان شهودی عمل کرد و یا برای مقادیر مختلف MNP میزان EPS را به دست آورد و کلاسترینگ را انجام داد و بر اساس تابع ارزیابی مانند SSE ، MNP مناسب را انتخاب کرد.

سوال ۵)

کدهای این قسمت در فایل‌های problem5-a.py و problem5-b.py قرار دارد.

بخش الف)

برای این بخش تابع‌های زیر را نوشته‌ام:

```
def euclidean_distance(p, q):
    """
    Euclidean Distance Between two points
    :param p: a d-dimentional point
    :param q: a d-dimentional point
    :return: a scalar, Euclidean Distance Between point p and q
    """
```

این تابع دو نقطه‌ی d بعدی را می‌گیرد و فاصله‌ی اقلیدسی بین آن‌ها را محاسبه می‌کند.

```
def davies_bouldin_index(clusters):
    """
    Davies Bouldin Index, for measuring quality of clusters
    :param clusters: a list numpy list which demonstrates
                     points in different clusters
    :return: a scalar number
    """
```

این تابع داده‌ها را در لیست‌های جداگانه‌ای می‌گیرد، هر لیست یک کلاستر را نشان می‌دهد.
سپس بر اساس رابطه‌های زیر میزان معیار Davies-Bouldin Index را محاسبه می‌کند:

$$S_i = \left(\frac{1}{T_i} \sum_{j=1}^{T_i} |X_j - A_i|^p \right)^{1/p}$$

$$R_{i,j} = \frac{S_i + S_j}{M_{i,j}}$$

$$D_i \equiv \max_{j \neq i} R_{i,j}$$

If N is the number of clusters:

$$DB \equiv \frac{1}{N} \sum_{i=1}^{N} D_i$$

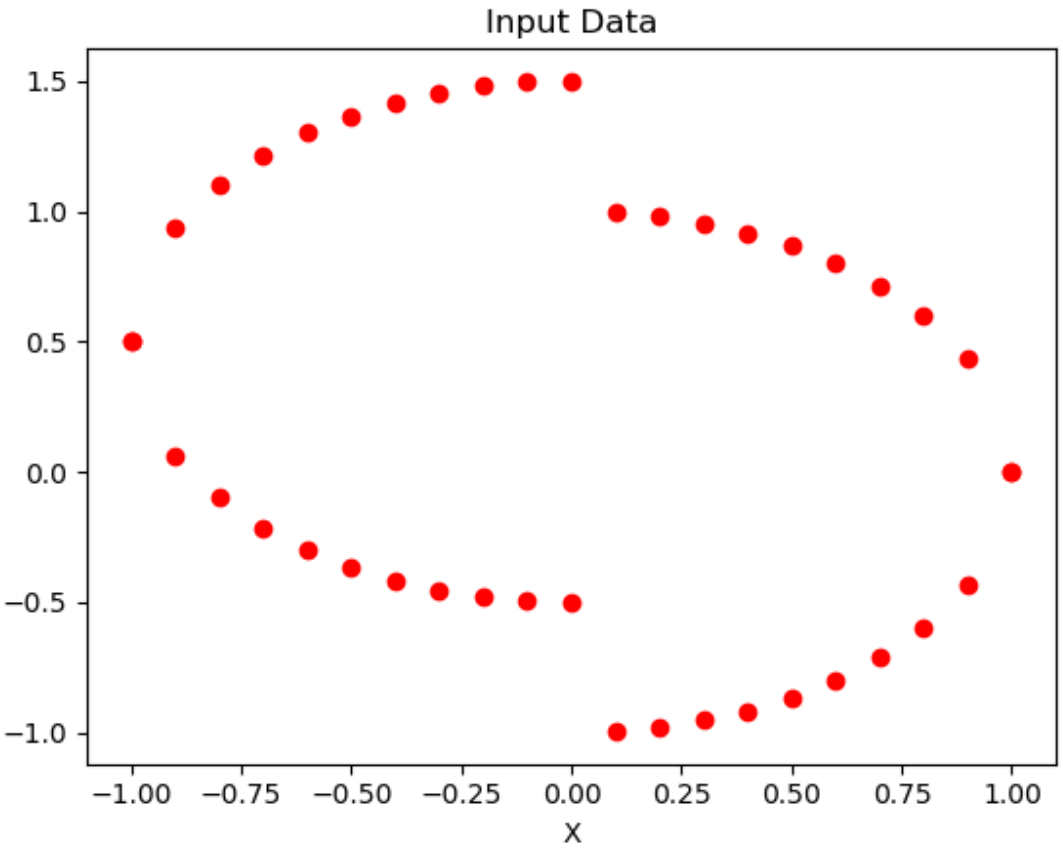
```
def kmeans(X, k, max_iter=100):
    """
    This function is a K-means clustering algorithm
    :param X: Input dataset
    :param max_iter: maximum number of iterations that clustering algorithm
                     does to coverage
    :return: Return Davies Bouldin cost for each iteration, clusters, Centroids for each iteration
             if a cluster become empty, exit function by returning None, None, None
    """
```

این تابع الگوریتم K-mean است، X داده‌های ورودی است ، K تعداد کلاسترهایی است که می‌خواهیم الگوریتم K-means به آن تعداد خوشه ایجاد کند. خروجی این تابع مقدارهای زیر است:

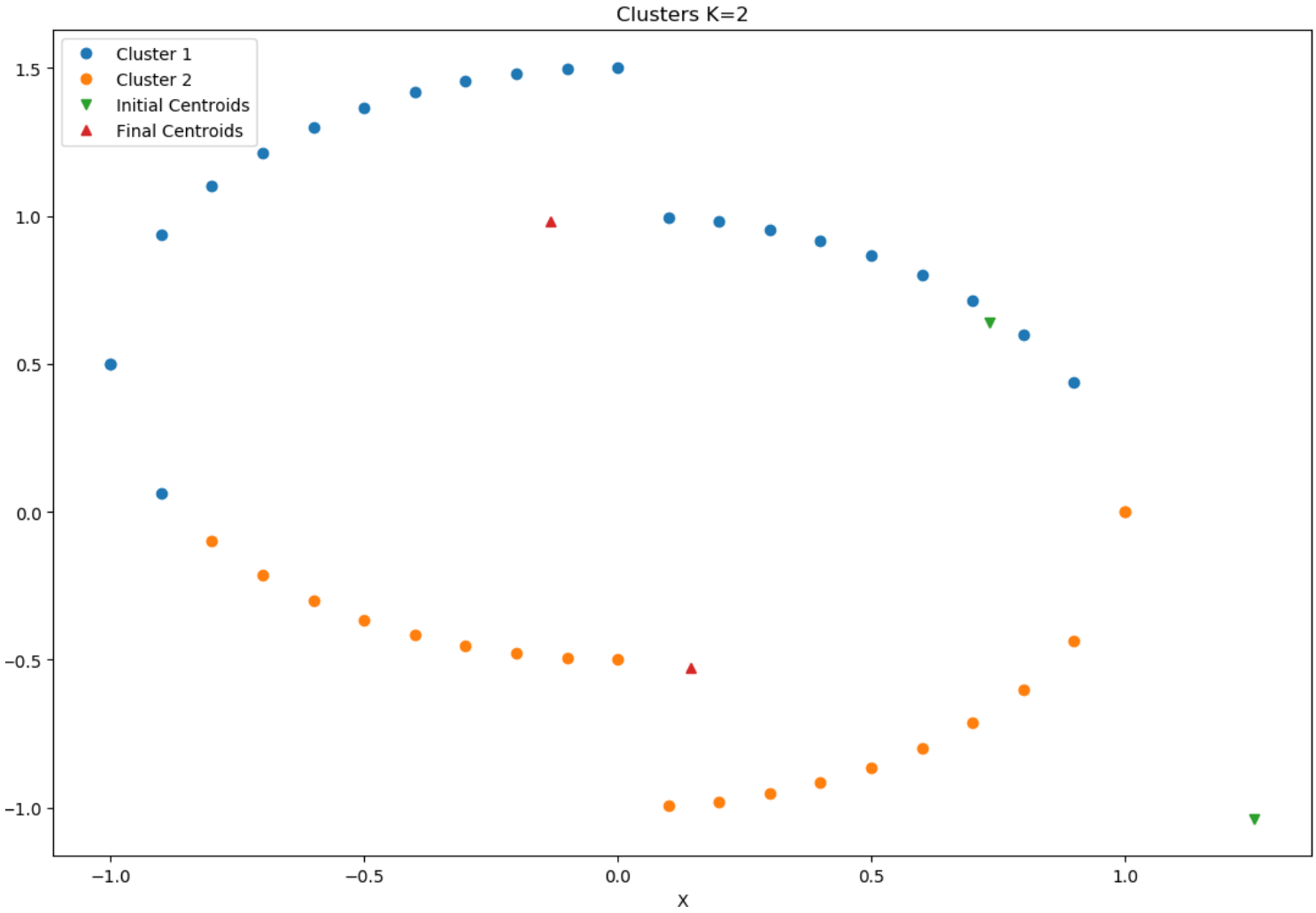
- مقدار Davies-Bouldin Index در هر Iteration
- K کلاستر به دست آمده‌ی نهایی
- مقدار Centroid ها در هر Iteration

در قسمت نهایی کد تابع های مختلف بالا را فراخوانده ام و نتایج زیر را به دست آورده ام:

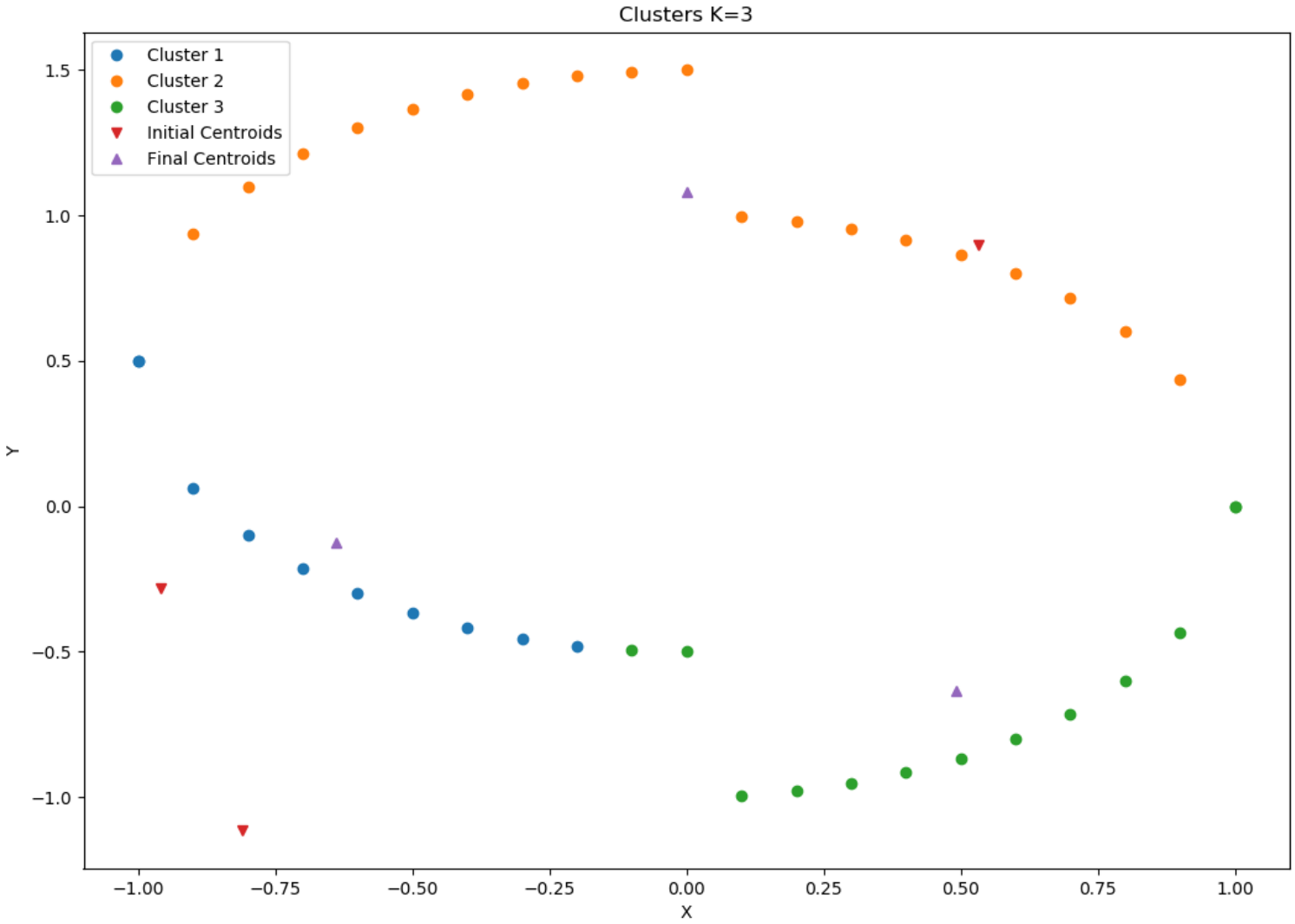
داده‌های ورودی:



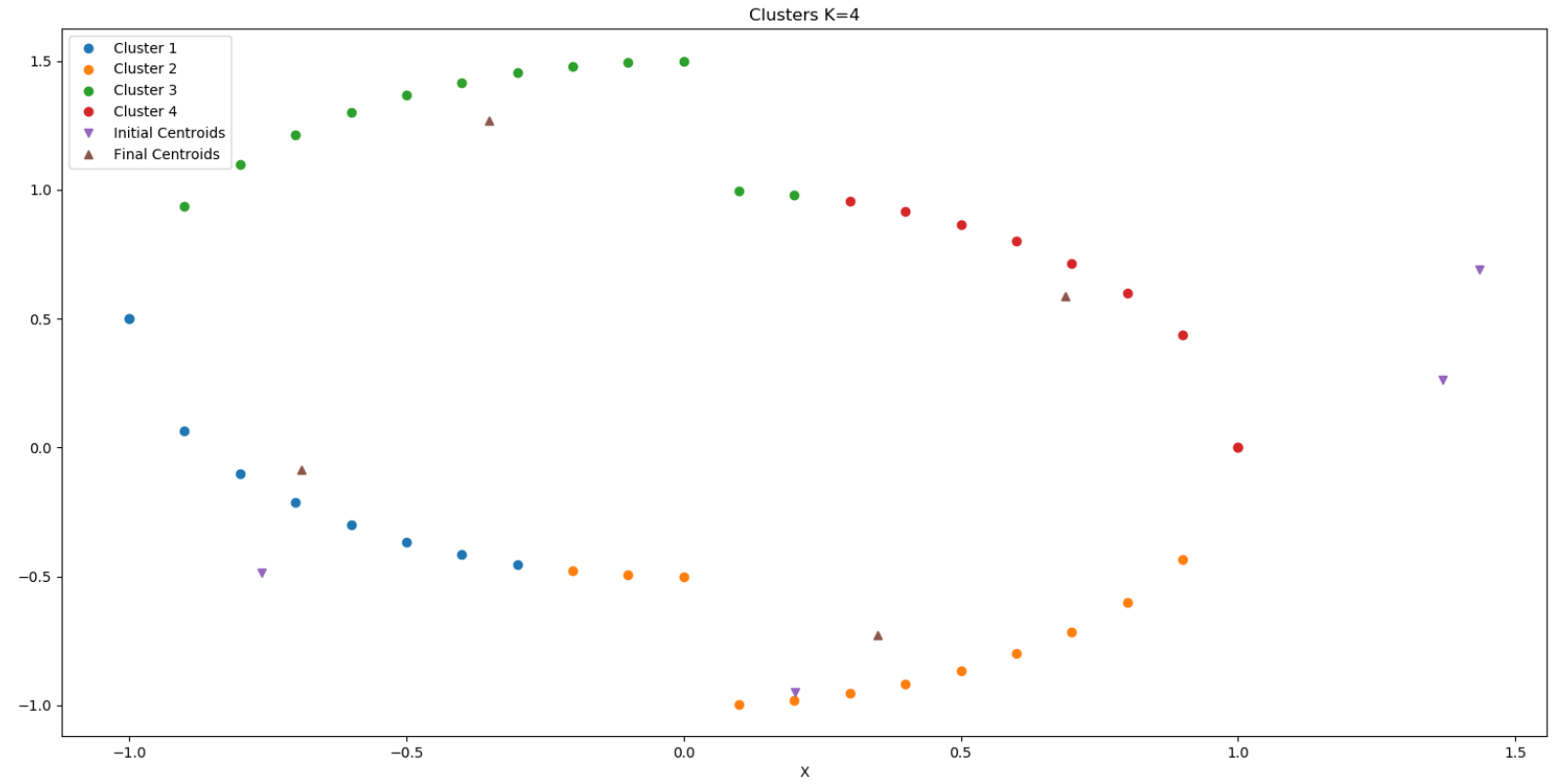
کلاسترهای به دست آمده به ازای K=2



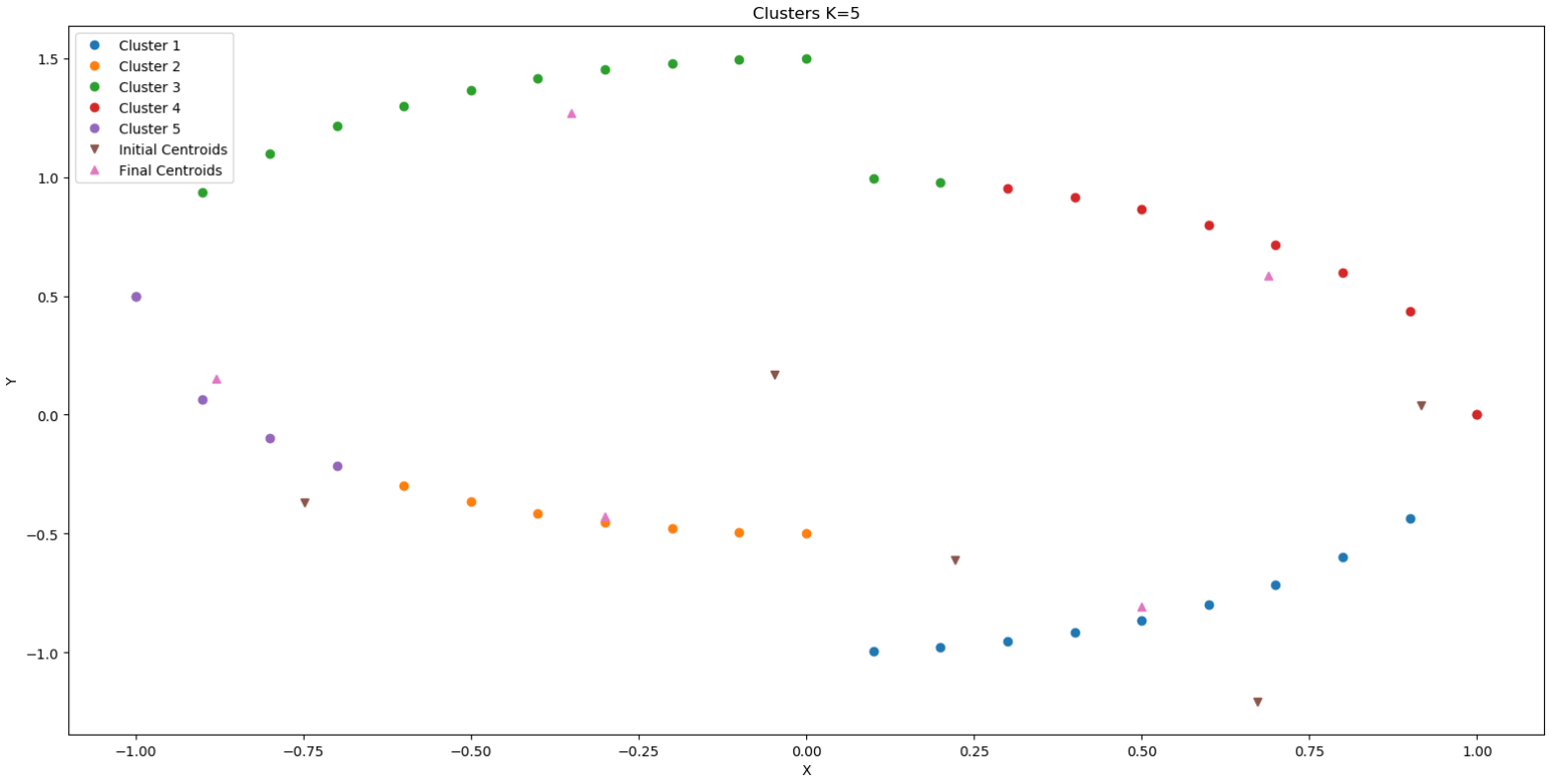
کلاسترهای به دست آمده به ازای K=3



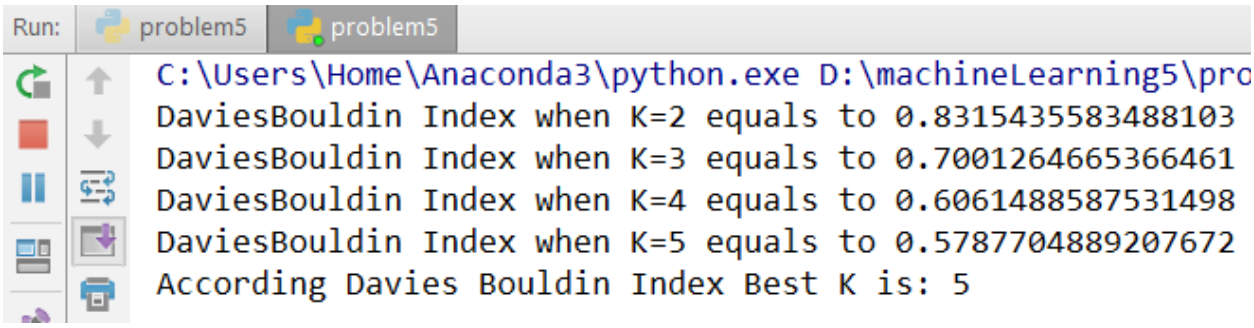
کلاسترهای به دست آمده به ازای K=4



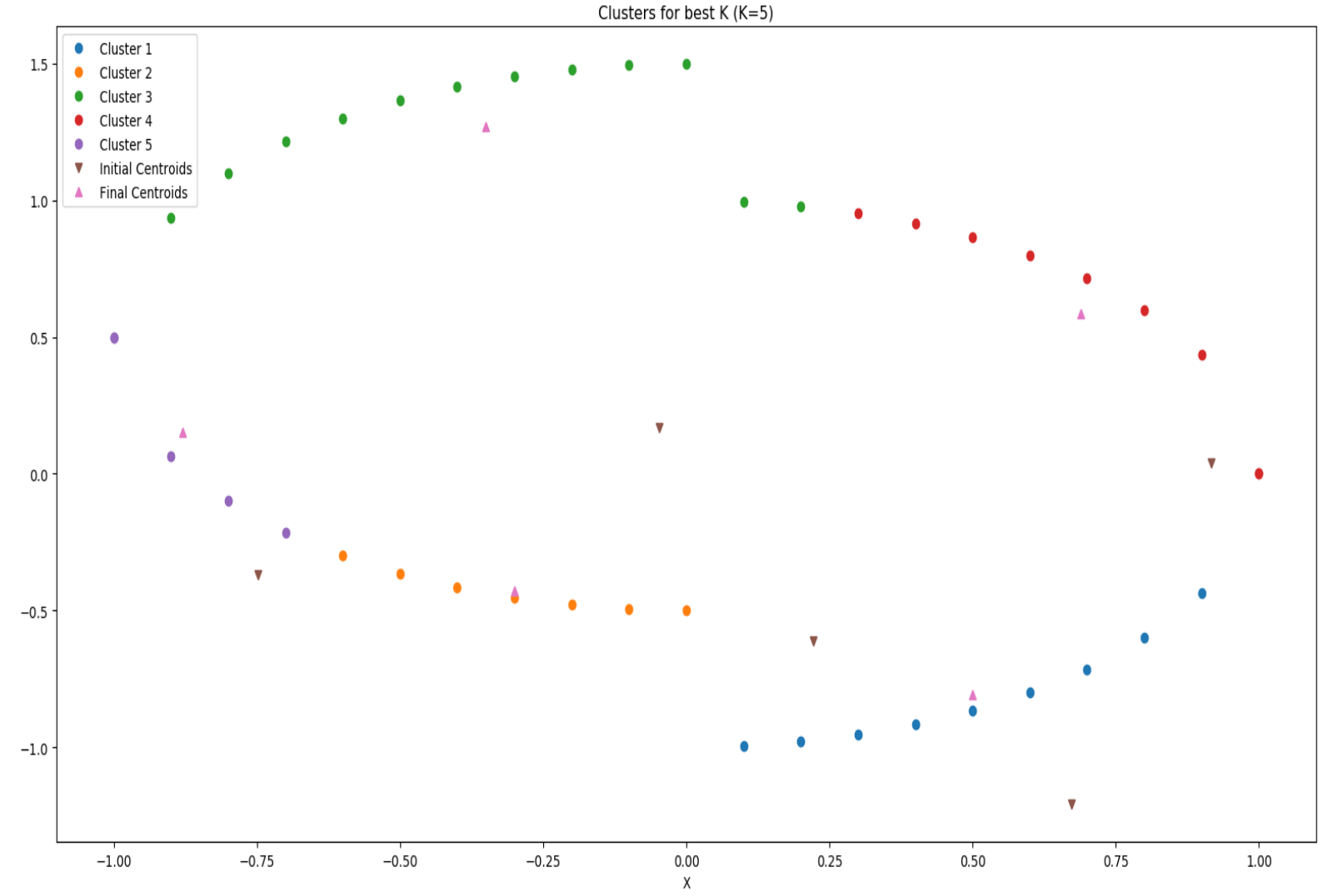
کلاسترهای به دست آمده به ازای K=5

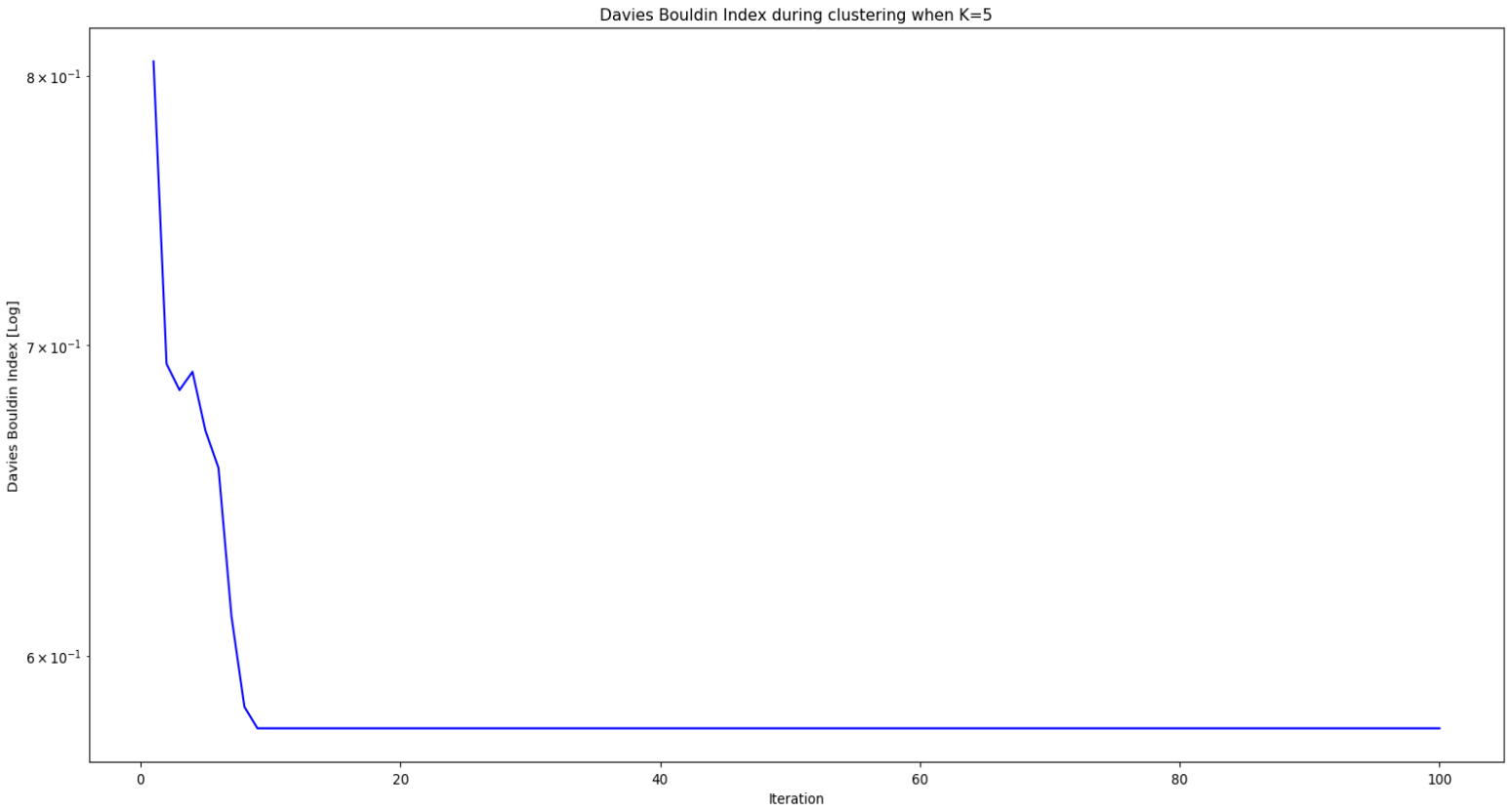


در زیر میزان معیار Davies-Bouldin Index را برای K های مختلف مشاهده می‌کنید:



که از آنجایی که K=5 کمترین میزان Davies-Bouldin را دارد آن را انتخاب می‌کنیم که در زیر کلاسترهای نهایی زمانی که K=5 است و میزان DB در هر Iteration را مشاهده می‌کنید.


Clusters for best K (K=5)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Initial Centroids
Final Centroids
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X



بخش ب)

کدهای این قسمت در problem5-b.py قرار دارد. برای انجام این قسمت تابع‌های زیر را نوشته‌ام:

```
def euclidean_distance(p, q):
    """
    Euclidean Distance Between two points
    :param p: a d-dimentional point
    :param q: a d-dimentional point
    :return: a scalar, Euclidean Distance Between point p and q
    """
```

این تابع دو نقطه‌ی d بعدی را می‌گیرد و فاصله‌ی اقلیدسی بین آن‌ها را محاسبه می‌کند.

```
def plot_k_nearest_neighbor_distances(X, k):
    """
    This Function finds K-Nearest-Neighbour for all points in dataset
    and calculates average distance of all points to their KNN.
    After that it plots sorted distances, form plot we can find out
    proper EPS for given MNP( MNP==K)
    :param X: Dataset, a numpy ndarray
    :param k: first k nearest neighbour
    :return: None
    """
```

این تابع طبق روشی که در سوال ۴ شرح داده شده است، K نزدیک ترین همسایه‌ی همه‌ی نقاط در دیتاست را به دست می‌آورد و سپس میانگین فاصله‌ی هر نقطه تا K نزدیک ترین همسایه‌اش را به دست می‌آورد. بعد از آن فاصله‌های به دست آمده را صعودی مرتب می‌کند و بعد نمودار آن را رسم می‌کند. نمودار امکان انتخاب EPS مناسب را می‌دهد.

```
def neighbours_of_points(X, eps):
    """
    This function find neighbours for all points. Neighbours of a point
    are points that their distance to the point is less equal to EPS
    :param X: a numpy array which shape m*d. m point each d dimensions.
    :param eps: is radios of a circle centered at a point which determines
                neighbourhood of the point.
    :return: a list of lists, each internal list determines indexes of
           neighbours of a point
    """
```

این تابع همسایه‌های تمام نقاط موجود در دیتاست را پیدا می‌کند. همسایه‌های یک نقطه، نقاطی هستند که فاصله‌ی آن‌ها کمتر مساوی EPS است. خروجی یک list از list ها است که لیست i ام حاوی ایندکس همسایه‌های نقطه‌ی i ام در دیتاست است.

```
def is_core(neighbours, MNP):
    """
    This function determines points of datasets are core or boundary
    :param neighbours:a list of lists, each internal list determines indexes of
           neighbours of a point
    :param MNP: Minimum number of neighbours that makes a point a core
    :return:
    """
```

این تابع خروجی تابع قبل که همسایگی‌های هر نقطه از دیتاست است را می‌گیرد و با توجه به MNP مشخص می‌کند آن نقطه و دایره‌ی آن از نوع Core یا Boundary است.

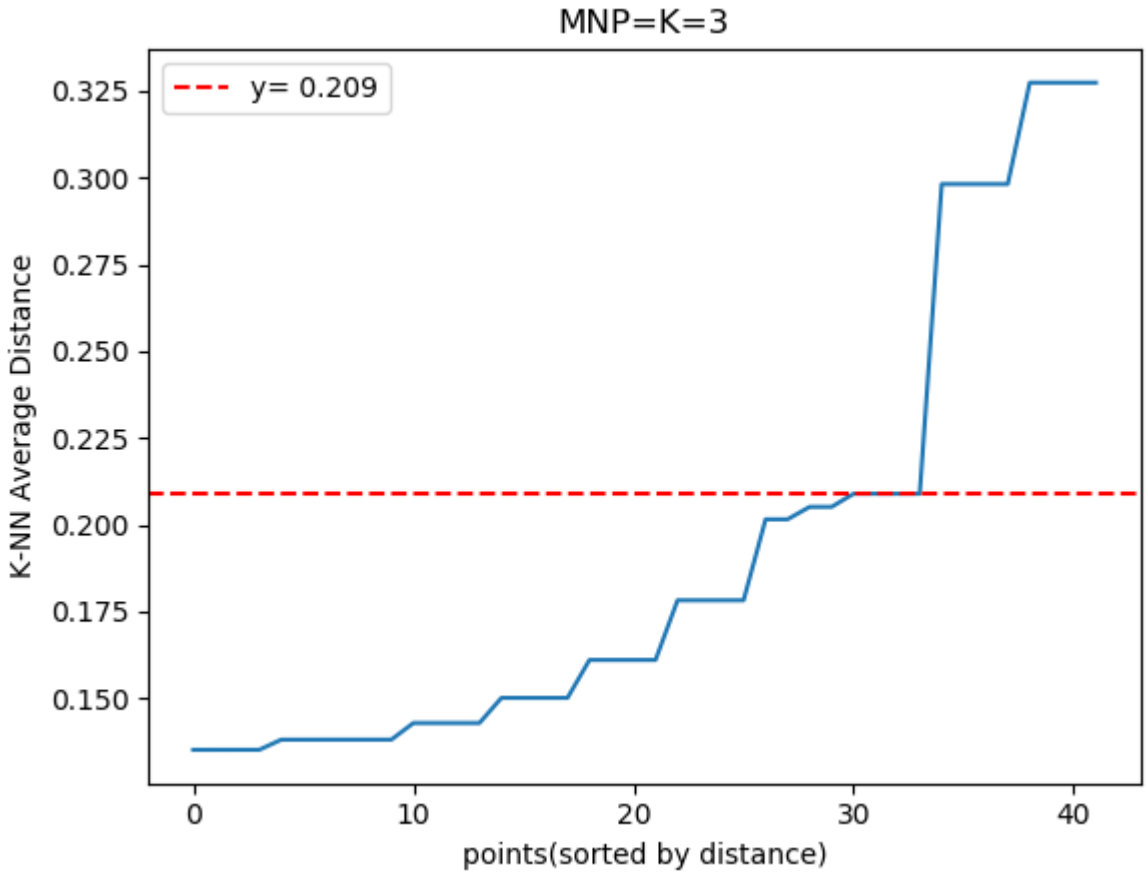
```
def DBScan(X, eps, MNP):
    """
    DBSCAN Clustering
    :param X: a numpy array which shape m*d. m point each d dimensions.
    :param eps:  is radios of a circle centered at a point which determine
                 neighbourhood of the point.
    :param MNP: Minimum number of neighbours that makes a point a core
    :return: It returns a pair: (clusters, Outliers/Noise)
    """
```

این تابع دو معیار EPS و MNP را همراه با دیتاست می‌گیرد و با استفاده از الگوریتم DBScan کلاسترینگ را بر روی داده‌ها انجام می‌دهد.

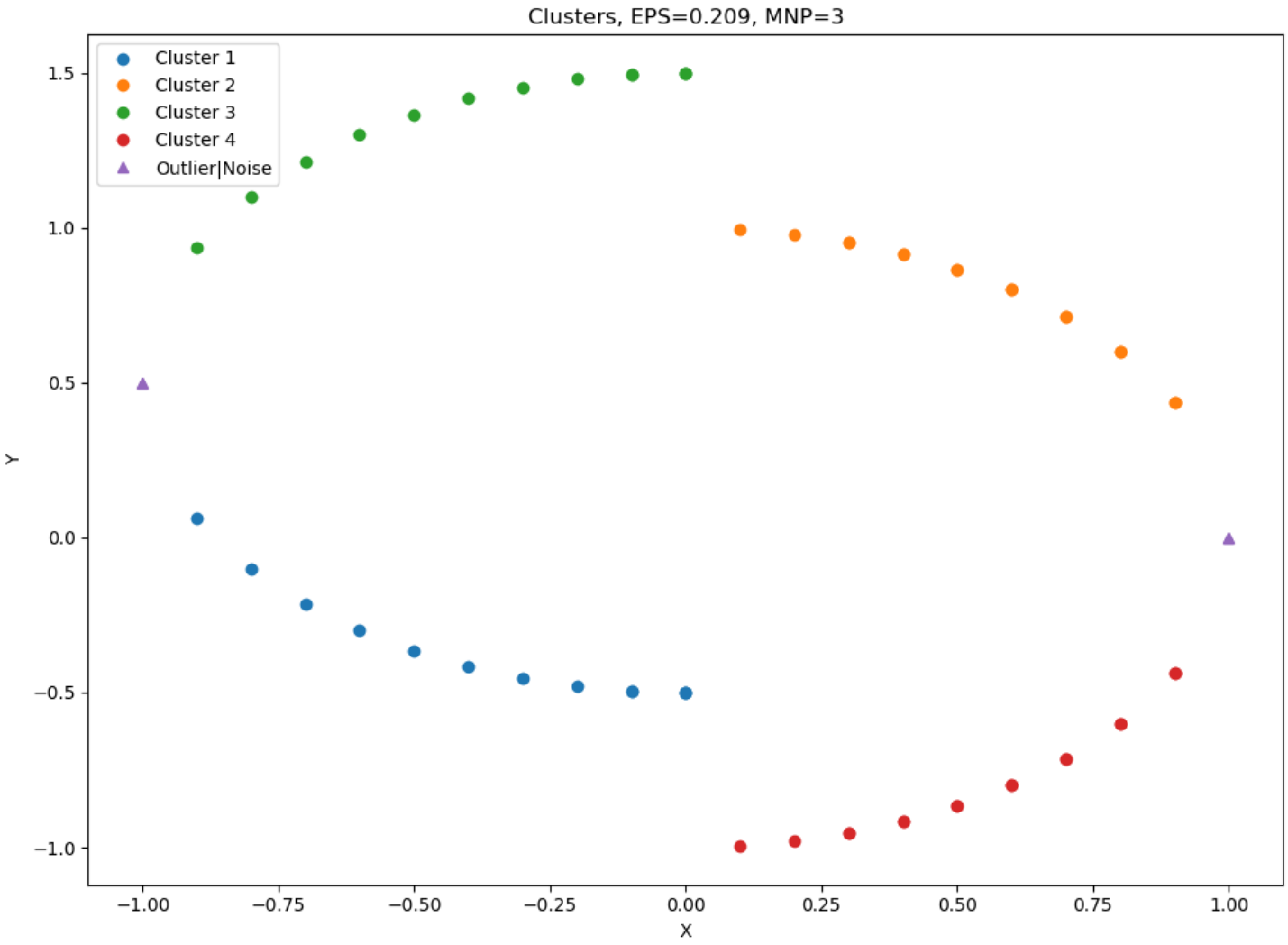
خروجی این تابع دو list است، لیست اول لیستی از لیست‌ها است که هر لیست یک کلاستر است، لیست دوم داده‌هایی هستند که به عنوان outlier و یا noise تشخیص داده شده اند و جز هیچ‌کدام از کلاسترها نیستند.

در ادامه کد هم داده‌ها از فایل خوانده شده‌اند و تابع‌های بالا بر روی آن‌ها فراخوانی شده است و در آخر کلاسترها با رنگ‌های مختلفی نمایش داده شده‌اند. در زیر خروجی‌های برنامه را مشاهده می‌کنید:

طبق روش توضیح داده شده در بالا و در سوال ۴ برای انتخاب پارامترها، در تصویر زیر میانگین فاصله‌ی نقطه‌های درون دیتاست تا K نزدیک ترین همسایه‌هایشان را مشاهده می‌کنید که بر اساس مقدار به صورت صعودتی مرتب شده اند و بعد نمودار آن‌ها رسم شده است. همین‌طور که مشاهده می‌شود هنگامی که مقدار محور عمودی برابر با ۰,۲۰۹ است جهش بزرگی در فاصله‌ی میانگین رخ می‌دهد به همین **دلیل ۰,۲۰۹ را عنوان EPS** زمانی که **MNP برابر با ۳** است انتخاب می‌کنیم.



بعد از فراخوانی DBSCAN با پارامترهای EPS=0.209 و MNP=3 داده‌ها به ۴ کلاستر تقسیم می‌شوند که با دایره و با رنگ‌های متفاوت نمایش داده شده اند و همین طور دو عدد از داده‌ها به عنوان Noise و یا Outlier تشخیص داده می‌شوند که با مثلث مشخص شده اند.



بخش ج)

در قمست الف از k-means استفاده کردیم که باید تعداد کلاسترها را خودمان پیدا می‌کردیم همین‌طور انتخاب تصادفی centroids های اولیه نقش بسیار مهمی در تعیین کلاسترها داشت و داده‌ها به درستی خوشه‌بندی نشدند. در این روش ۵ کلاستر با توجه به معیار از سایر Kها بهتر بود.

ولی در DBScan با یک روش ابتکاری توانستیم پارامترهای مناسب MNP و EPS را پیدا کنیم و همین طور نیازی به مشخص کردن تعداد کلاسترها نیست. همین طور این الگرویتم داده‌های

Outlier/Noise را تشخیص داد که منجر به دست آمدن کلاسترهای مناسبی شد. در این روش ۴ کلاستر به دست آمد و همین طور ۲ داده‌ی نویز/پرت، که از نتیجه‌ی به دست آمده در قسمت الف بسیار بهتر است.

سوال ۶)

بخش الف)

کدهای این بخش در فایل problem6-a.py قرار دارد. برای انجام این بخش تابع‌های زیر را نوشته‌ام:

```
def euclidean_distance(p, q):
    """
    Euclidean Distance Between two points
    :param p: a d-dimentional point
    :param q: a d-dimentional point
    :return: a scalar, Euclidean Distance Between point p and q
    """
    import numpy as np

    dis = np.sqrt(((p-q)**2).sum())
    return dis
```

این تابع فاصله‌ی اقلیدسی بین دو نقطه‌ی d بعدی را محاسبه می‌کند.

```
def sum_of_squared_error(clusters):
    """
    This function calculates SSE (Sum Of Squared Error), for measuring quality of clusters
    :param clusters: a list numpy list which demonstrates
                    points in different clusters
    :return: a scalar number
    """
```

این تابع مجموع مربع خطاها را برای یک یا چند کلاستر محاسبه می‌کند که از این معیار برای انتخاب یک کلاستر از میان دو کلاستر برای تقسیم کردن استفاده می‌کنیم.

```
def kmeans(X, k, max_iter=100):
    """
    This function is a K-means clustering algorithm
    :param X: Input dataset
    :param max_iter: maximum number of iterations that clustering algorithm
                     does to coverage
    :return: Return SSE cost for each iteration, clusters, Centroids for each iteration
            if a cluster become empty, exit function by returning None, None, None
    """
```

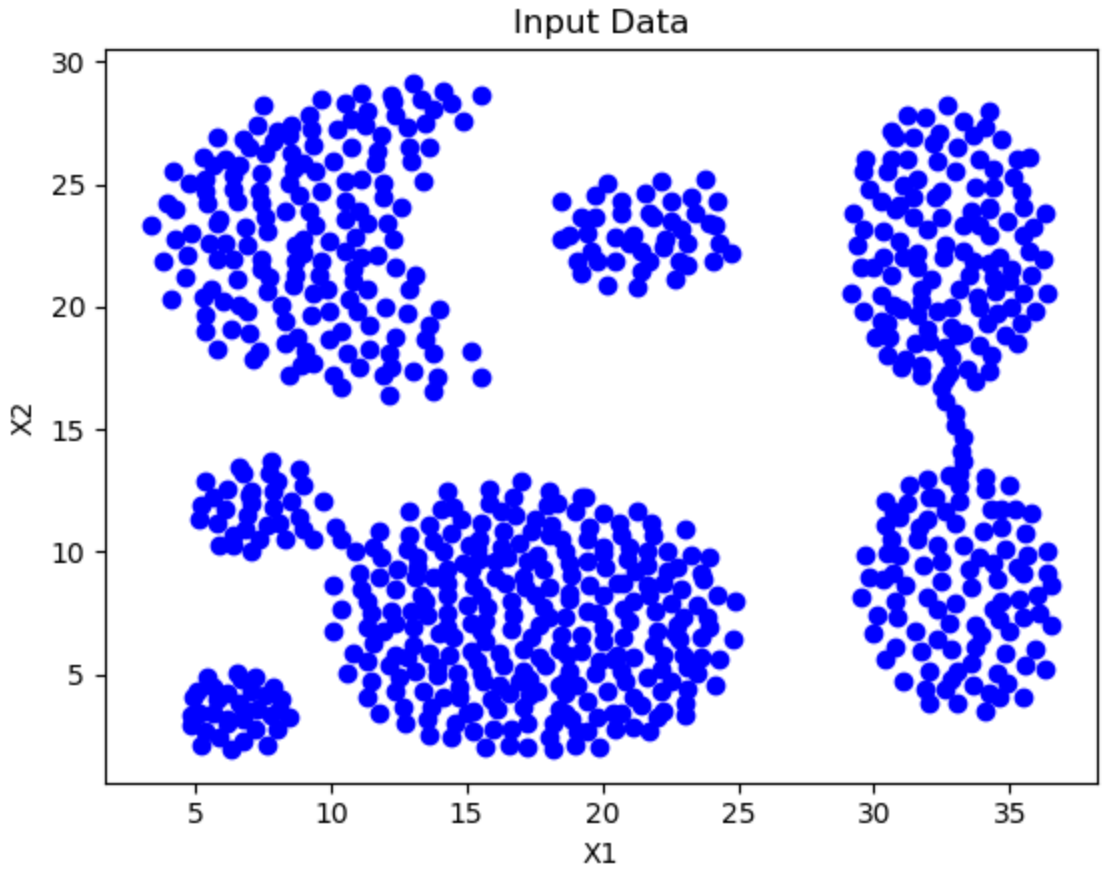
این تابع، الگوریتم خوشه‌بندی K-means است که در سوال قبل از آن استفاده کردیم و آن را توضیح دادیم.

```
def top_down(X, n):
    """
    This function uses a top-down method, in each step it divides data to two cluster
    with k-means (k=2), then it selects a cluster with higher SSE, and it do the same
    on it, it continue to breaking until it reach a cluster with one point or it reach
    the limit of dividing steps(input argument 'n')
    :param X: Input dataset
    :param n: limit of number of dividing steps that is done on dataset
    :return: clusters
    """
```

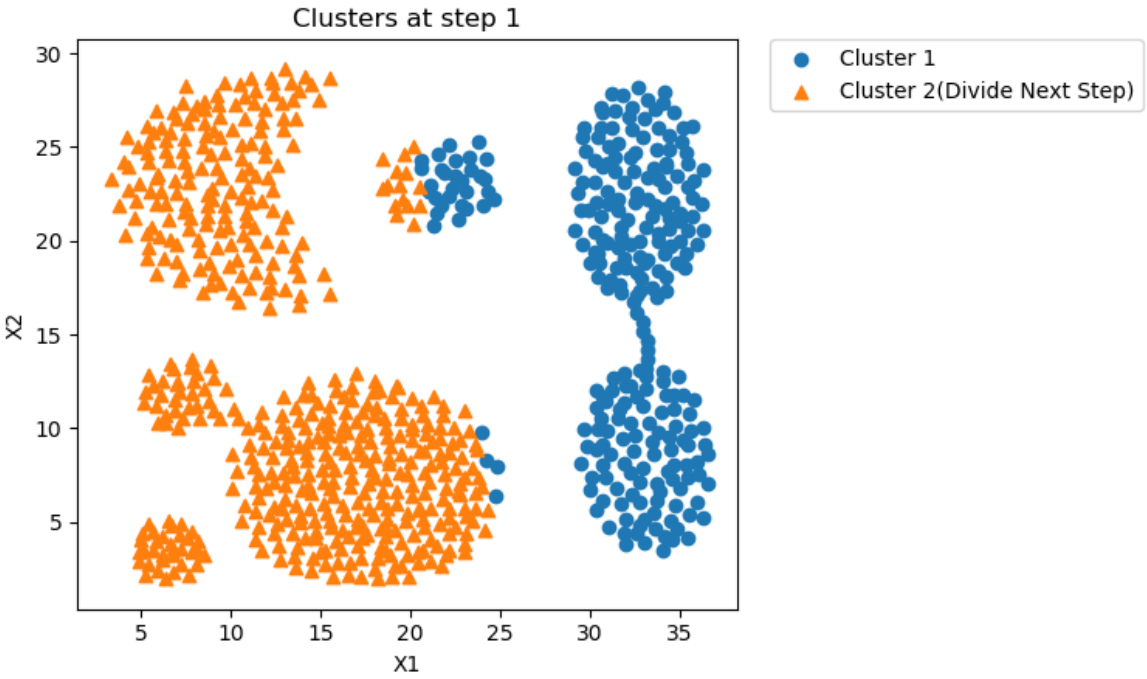
در این تابع روش کلاسترینگ divisive(Top Down) توضیح داده شده در سوال، پیاده‌سازی شده است. این تابع یک دیتاست به همراه تعداد خوشه‌های خروجی را می‌گیرد و با استفاده از K-means آن را به دو قسمت تقسیم می‌کند و بعد با استفاده از معیار Some Of Square Error خوشه‌ای را انتخاب می‌کند که SSE بزرگ تری دارد، سپس همان کار را بر روی خوشه‌ی منتخب انجام می‌دهد تا زمانی که به تعداد خوشه های خواسته شده برسد. همین طور خوشه‌ها در طی کلاسترینگ رسم می‌شوند.

در ادامه‌ی کد فایل داده‌ها خوانده شده اند، سپس تابع‌های معرفی شده در بالا فراخوانده شده‌اند.

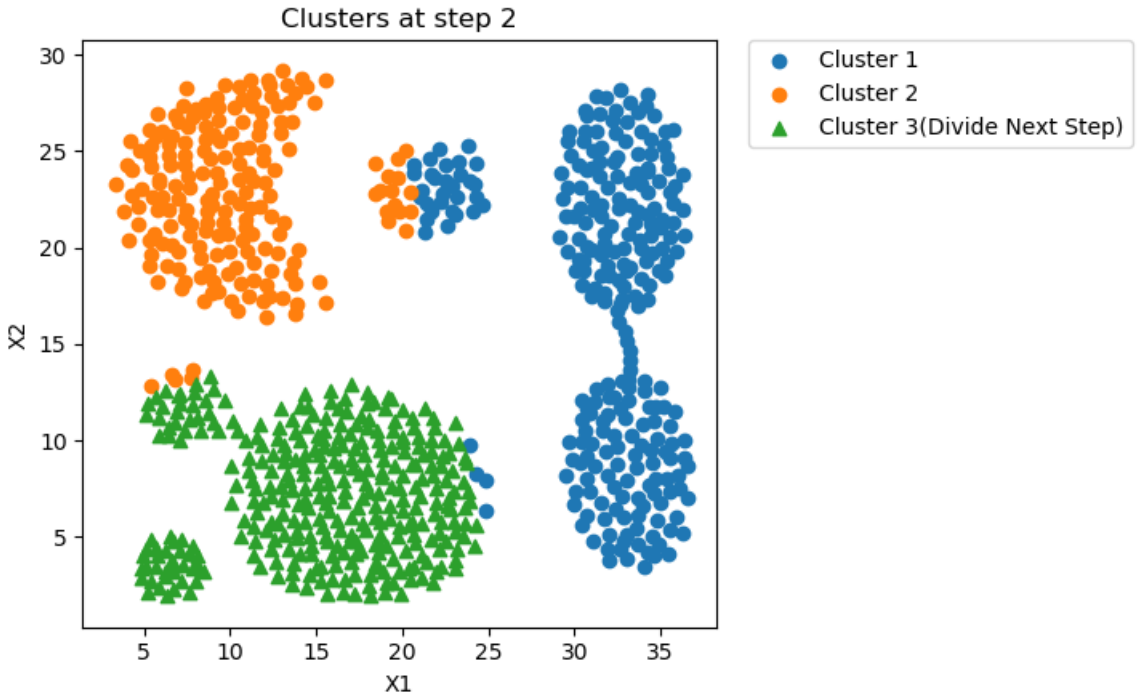
در زیر خروجی‌های برنامه را مشاهده می‌کنید:



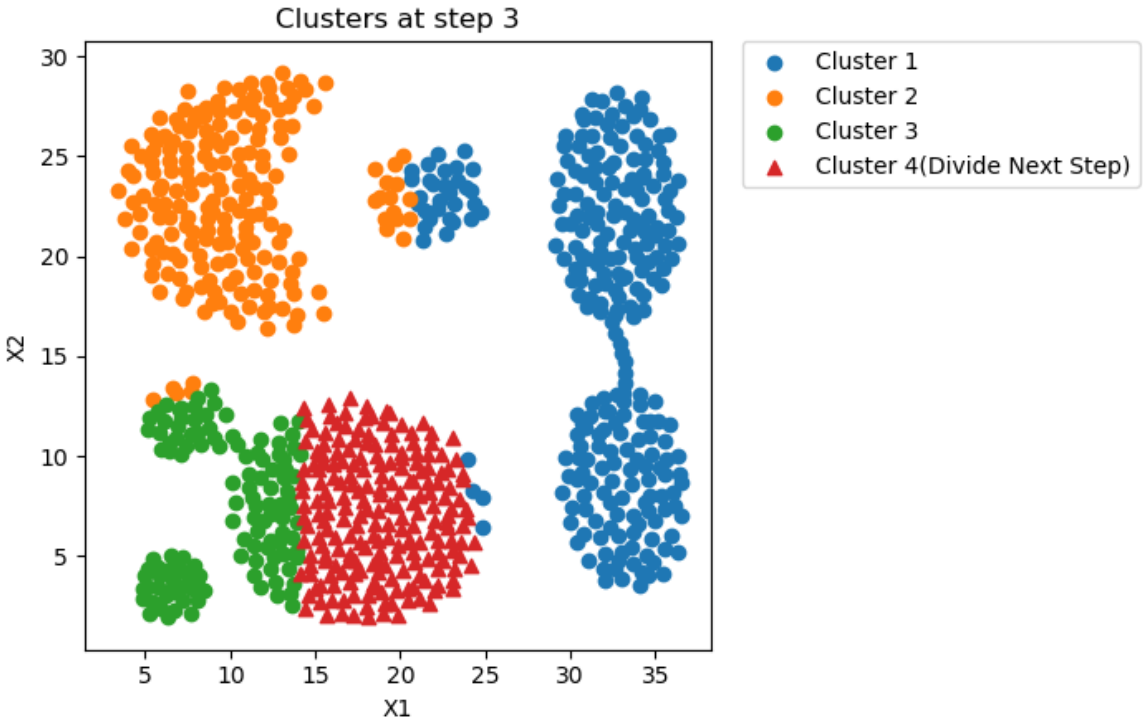
داده‌های ورودی



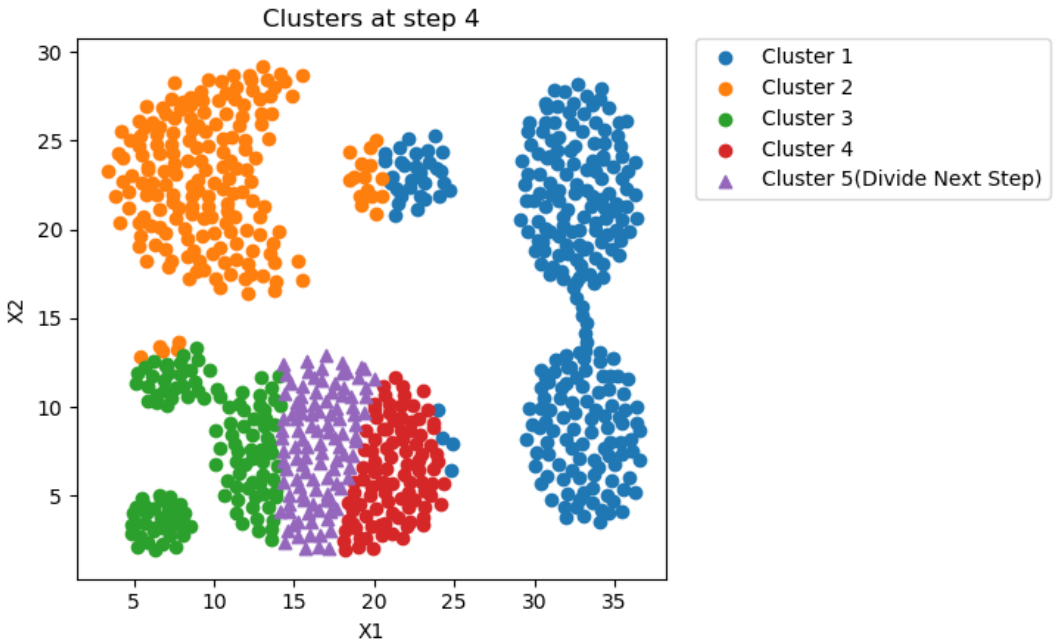
کلاسترها در مرحله‌ی اول تقسیم دیتاست



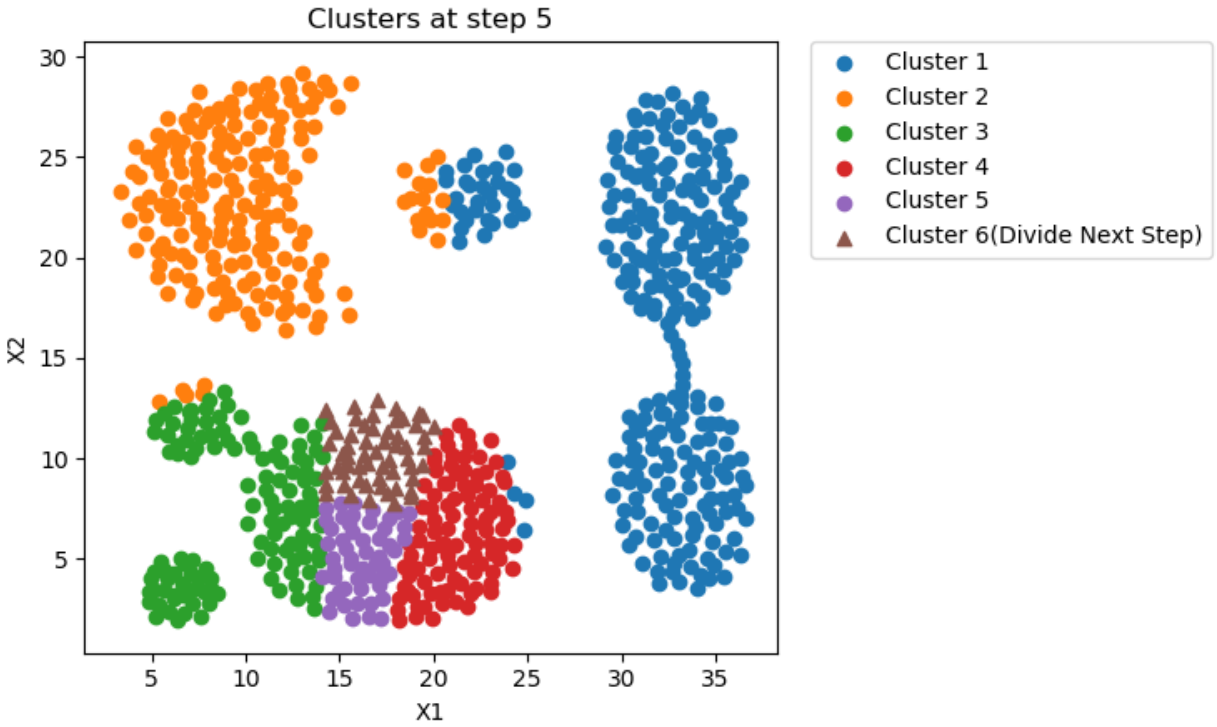
کلاسترها در مرحله‌ی دوم تقسیم دیتاست



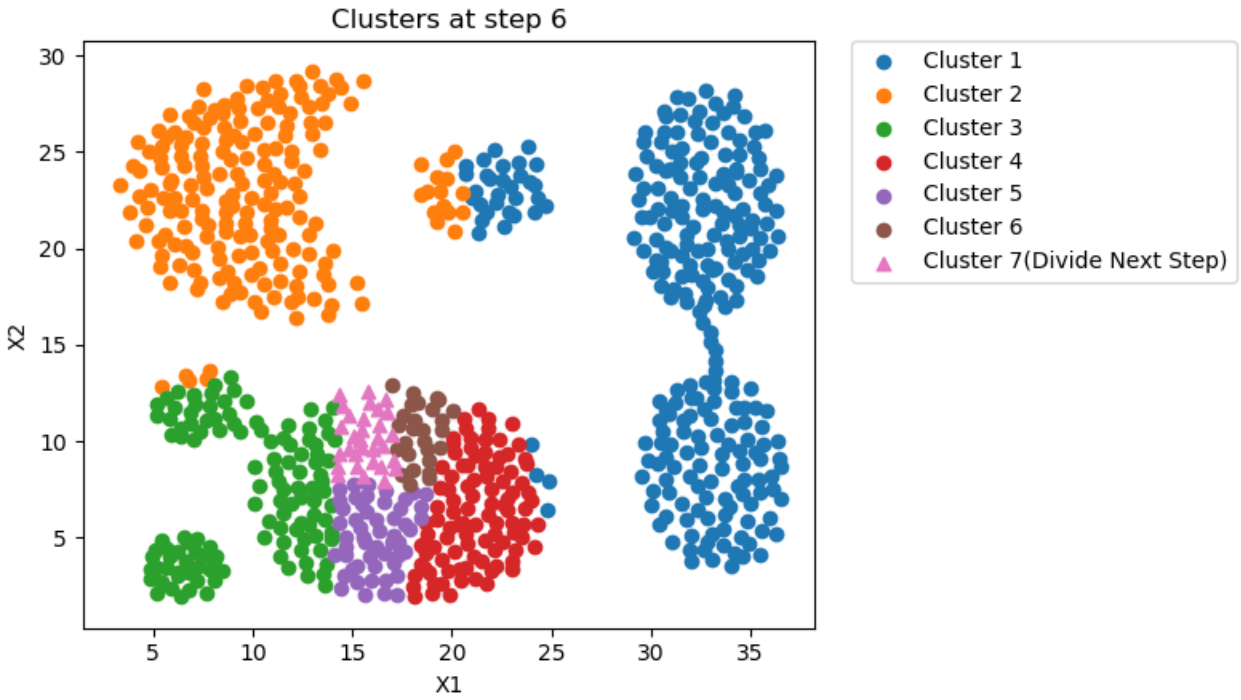
کلاسترها در مرحله‌ی سوم تقسیم دیتاست



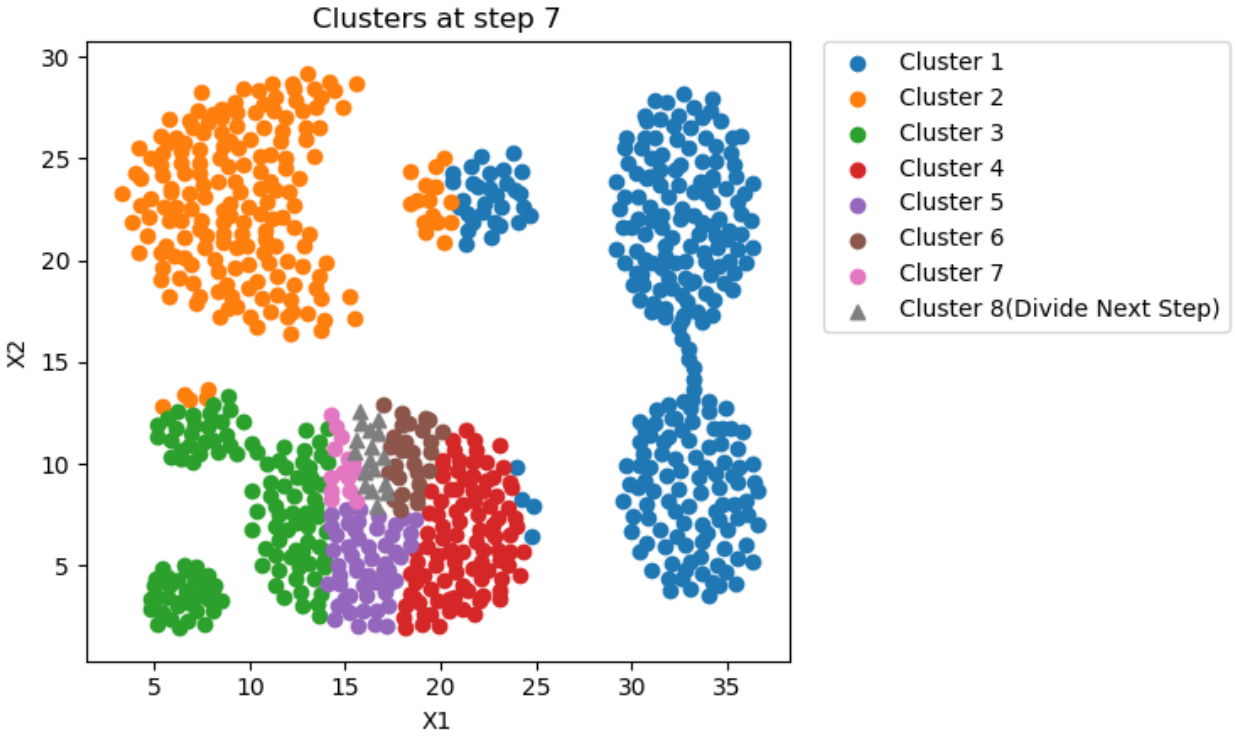
کلاسترها در مرحله‌ی چهارم تقسیم دیتاست



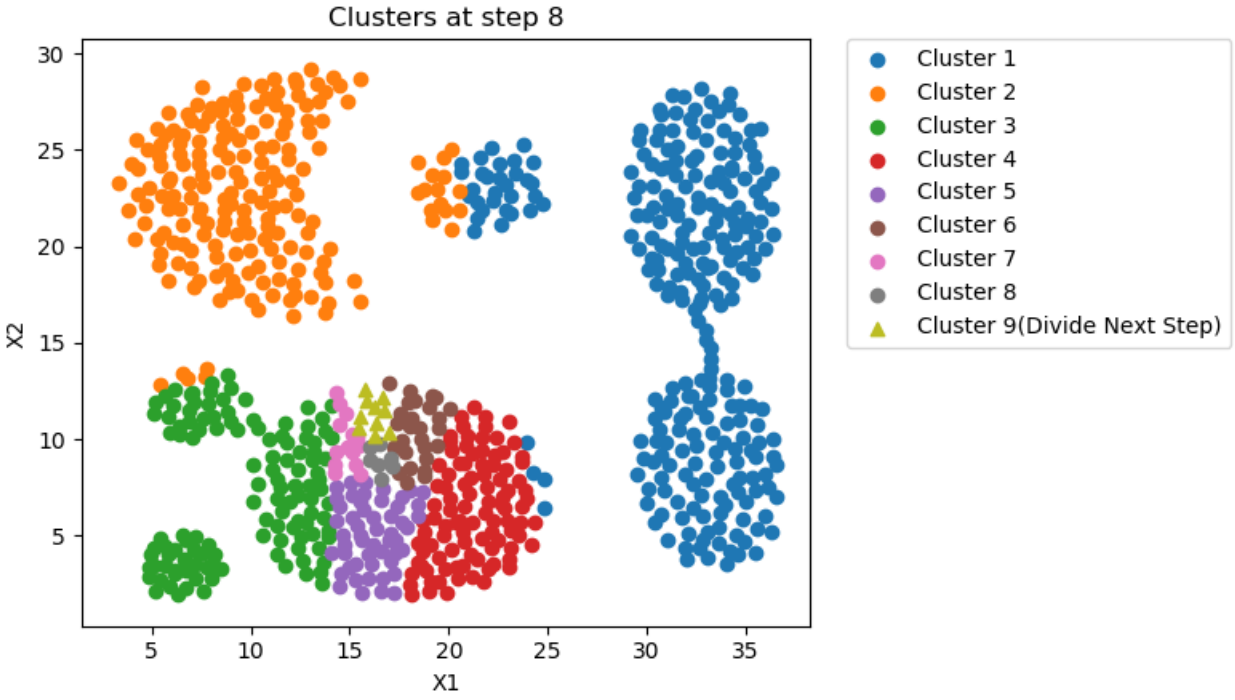
کلاسترها در مرحله‌ی پنجم تقسیم دیتاست



کلاسترها در مرحله‌ی ششم تقسیم دیتاست



کلاسترها در مرحله‌ی هفتم تقسیم دیتاست



کلاسترها در مرحله‌ی هشتم تقسیم دیتاست

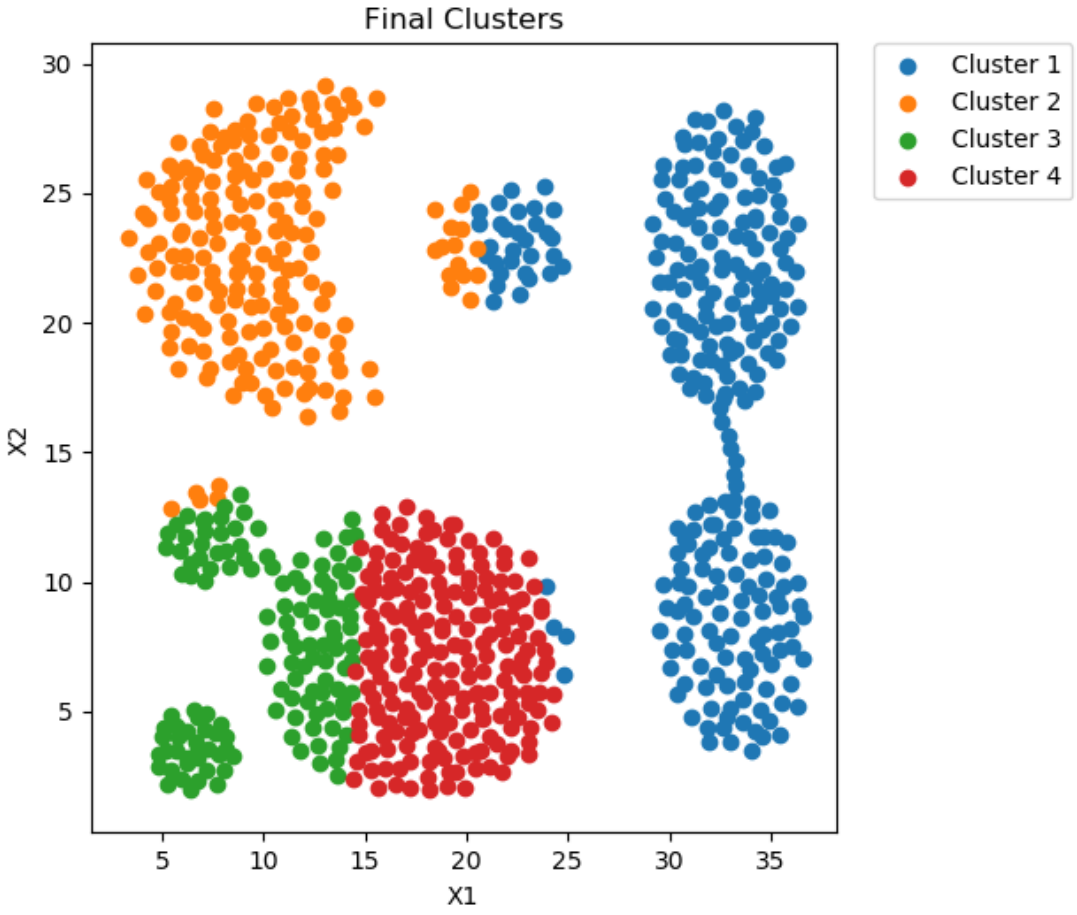
کلاسترها در مرحله‌ی نهم تقسیم دیتاست

بخش ۶-b)

کدهای این بخش در فایل problem6-b.py موجود است. برای انتخاب خودکار تعداد خوشه‌ها طبق روش زیر عمل کرده‌ام:

بعد از تقسیم داده‌ها در هر مرحله دو خوشه به دست می‌آید که در بخش قبل سوال خوشه‌ای را که SSE بزرگ‌تری داشت را انتخاب می‌کردیم و در مرحله بعد آن را به دو قسمت تقسیم می‌کردیم ولی در این قسمت هر گاه SSE دو خوشه بهم نزدیک باشد از تقسیم بیشتر خودداری می‌کنیم و کار را متوقف می‌کنیم. زیرا اگر SSE دو خوشه نزدیک به هم باشد و یک خوشه را انتخاب کنیم و آن را به دو خوشه با استفاده از K-means تقسیم کنیم، دو خوشه به دست آمده SSE بسیار

کمتری نسبت به خوشه‌ای که تقسیمش نکردیم دارند. در صورتی که اختلاف SSE دو خوشه کمتر از ۲۰ درصد SEE بزرگ‌تر باشد، تقسیم خوشه را متوقف می‌کنیم. بجز شرط بالا باقی کد مانند بخش قبل است به همین دلیل آن‌ها را دوباره توضیح نمی‌دهم. خروجی حاصل از اجرای برنامه که منجر به ایجاد ۴ خوشه شده است:



بخش۶ –c)

کدهای این بخش از سوال در فایل problem6-c.py قرار دارد. برای پیاده سازی این بخش از سوال، تابع‌های زیر را پیاده سازی کرده‌ام:

```
def euclidean_distance(p, q):
    """
    Euclidean Distance Between two points
    :param p: a d-dimentional point
    :param q: a d-dimentional point
    :return: a scalar, Euclidean Distance Between point p and q
    """
```

این تابع فاصله‌ی اقلیدسی دو نقطه‌ی دو بعدی در فضا را محاسبه می‌کند.

```
def create_distance_matrix(X):
    """
    This function creates a m*m matrix, which element i,j is
    distance between point i and j of dataset.
    :param X: Input dataset, a m*d numpy ndarray, m observations, each d features
    :return: distance matrix, headers of columns, number of distances
    """
```

این تابع یک ماتریس m*m ایجاد می‌کند که مقدار درایه‌ی i,j برابر با فاصله‌ی اقلیدسی نقطه‌ی i ام دیتاست تا نقطه‌ی j ام دیتاست است. همین طور این تابع یک آرایه ایجاد می‌کند که نشان می‌دهد هر ستون حاوی کدام نقاط است. از آن جایی که در روش bottm-up در مرحله‌ی اول هر نقطه یک کلاستر است هر ستون معادل یک نقطه است.

```
def update_distance_matrix(distance_matrix, headers, num_dis, distance_function, p, q):
    """
    This function gets distance matrix and deletes rows p and q,
    also it deletes columns p and q.
    After omitting rows and columns, it updates headers.
    Then it adds row and column {p, q} which is obtained by merging
    clusters corresponding p and q. Finally it calculates distance of this new cluster
    to other cluster
    :param distance_matrix:  A m*m matrix, which element i,j is
                distance between cluster i and j
    :param headers: cluster corresponding to each row and col
    :param num_dis: this matrix used when we want to calculate average link
                    it holds n1*n2 in (1/(n1*n2)) * sum of all distances of two clusters
    :param distance_function: can be numpy.max, numpy.min, numpy.mean function and etc.
    :param p: row and column p
    :param q: row and column q
    :return: new distance matrix and headers
    """
```

در الگوریتم های پایین به بالا در هر مرحله دو کلاستر که بیشترین شباهت را به هم دارند را با هم merge می‌کنیم برای همین منظور باید ماتریس فاصله را بعد از هر بار merge کردن به روز کنیم. این تابع کار آپدیت کردن ماتریس فاصله را بر عهده دارد. این تابع ماتریس فاصله را می‌گیرد و سطرها و ستون‌های p و q را حذف می‌کند. سپس یک سطر و ستون اضافه می‌کند که معادل کلاستر حاصل از merge دو کلاستر معادل ستون‌های p و q است. سپس فاصله‌ی کلاستر جدید را بر اساس تابع distance_function با سایر کلاسترها به دست می‌آورد که distance_function می توان تابع np.min, np.max و mean باشد. همین طور لیست headers که نشان دهنده‌ی کلاستر معادل هر ستون هست را نیز آپدیت می‌کند و در انتها ماتریس فاصله و لیستheaders آپدیت شده را باز می‌گرداند.

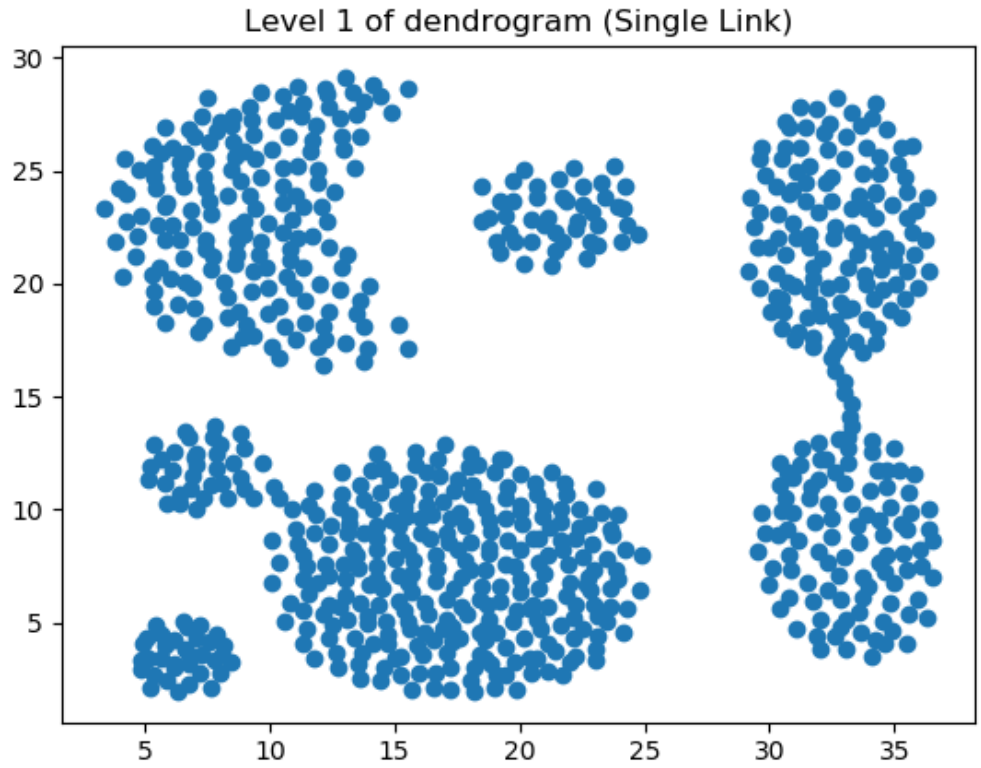
```
def agglomerative_clustering(X, distance_function, begin_level_dendrogram, end_level_dendrogram):
    """
    This function does agglomerative clustering, depends on distance_function, it is
    distance_function==np.min ==> single link
    distance_function==np.max ==> complete link
    distance_function==mean ==> average link
    distance_function==arbitrary function  ==> arbitrary link

    :param X: Input dataset, a m*d numpy ndarray, m observations d features
    :param distance_function: for selecting distance between two clusters
           can be man, mean, average(mean), ...
    :param begin_level_dendrogram, end_level_dendrogram: it returns clusters for different levels of
           dendrogram which are between thoese input argument
    :return: clusters at different levels of dendrogram. it return a dictionary which key i is
           equal clusters at level i.
    """
```

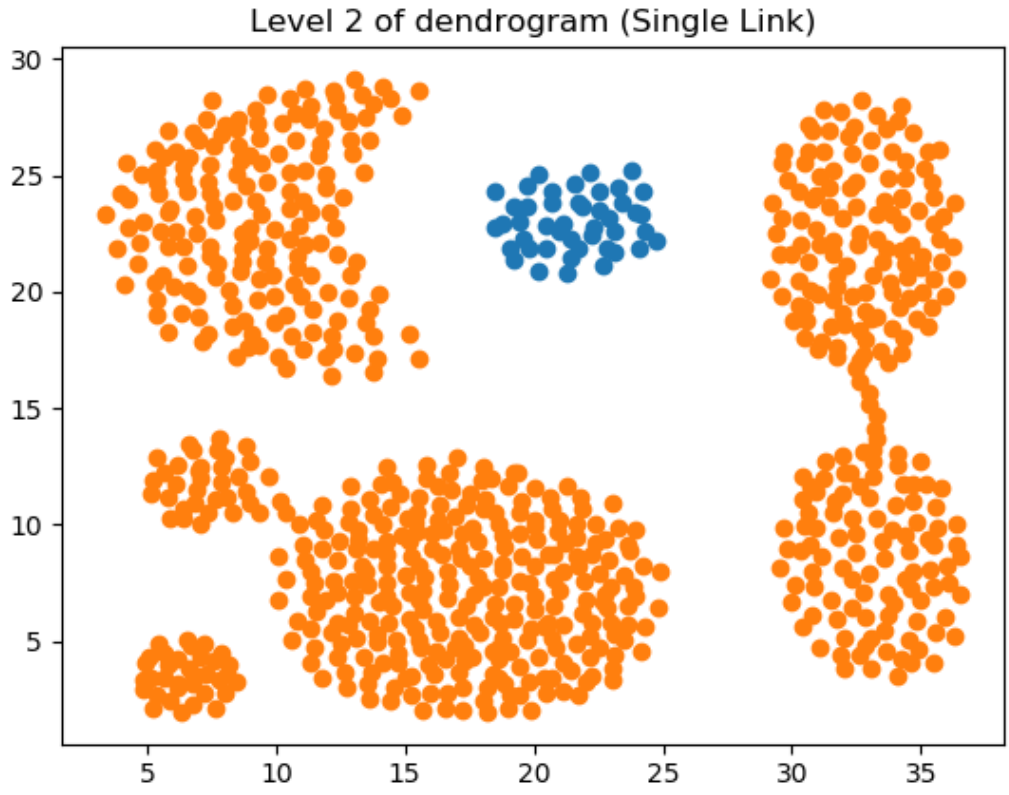
این تابع خوشه‌بندی پایین به بالا است که به توجه به آرگمان ورودی distance_function می‌تواند Single Link, Complete Link و یا Average Link باشد.

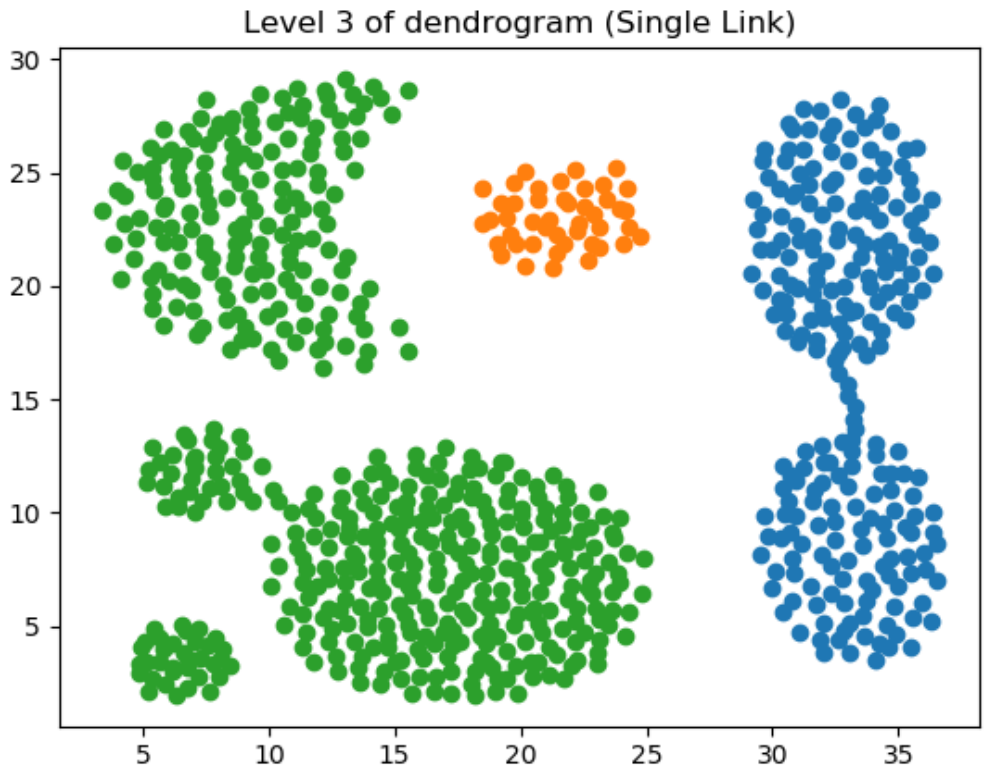
این تابع در ابتدا با فراخوانی تابع create_distance_matrix ماتریس فاصله بین نقاط مختلف دیتاست را می‌سازد سپس به تعداد در هر مرحله تا زمانی که به یک خوشه برسد هر بار دو کلاستر که کمترین فاصله را دارند را با استفاده از ماتریس فاصله پیدا می‌کند سپس آن‌ها را باهم merge می‌کند و این کار را با آپدیت کردن ماتریس فاصله، با استفاده از تابع update_distance_matrix انجام می‌دهد. و در آخر کلاسترهای level های مختلف dendrogram را که کاربر مشخص کرده است را برمی‌گرداند.

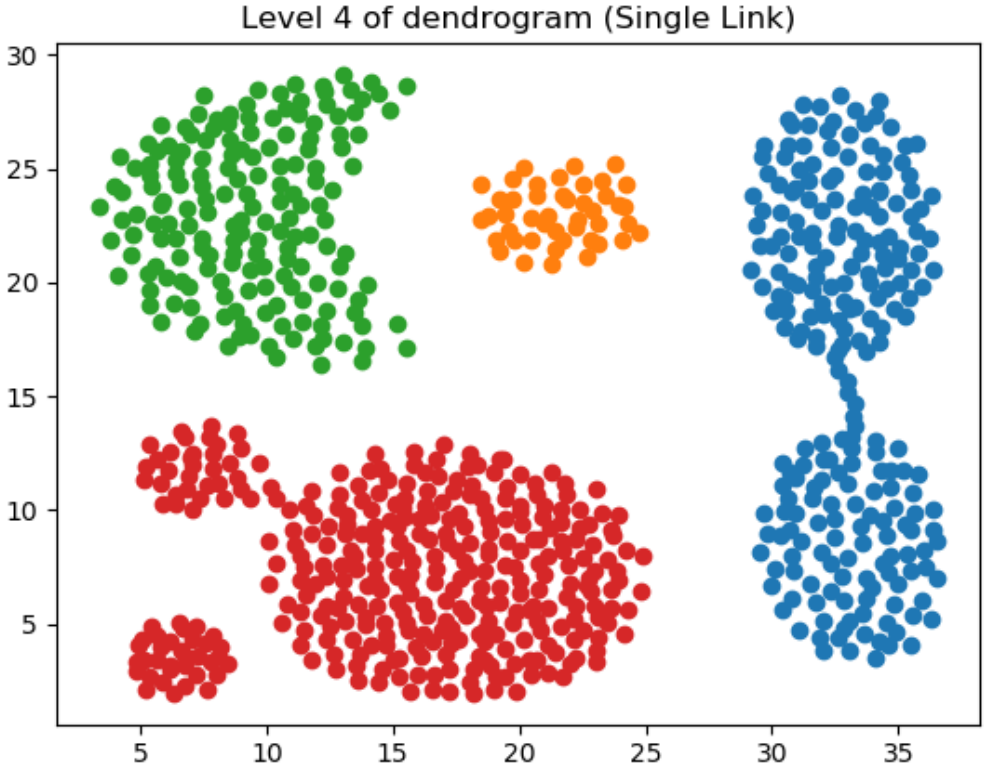
در ادامه‌ی کد، فایل داده‌ها خوانده شده است و سپس تابع agglomerative_clustering با تابع های مختلف فاصله فراخوانده شده است و سپس خوشه‌های levelهای یک تا ده Single Link, Complete Link و Average Link رسم شده است. در زیر خروجی‌های این قست را مشاهده می‌کنید:

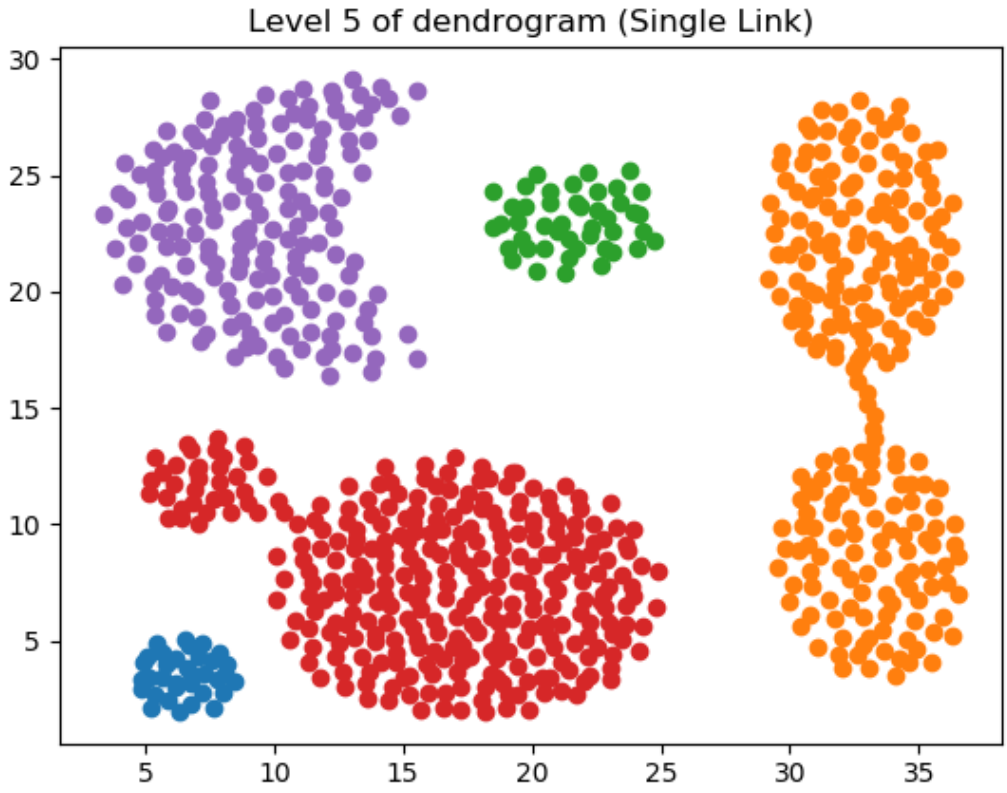
Level های یک تا ده Dendrogram الگوریتم خوشه‌بندی Single Link:

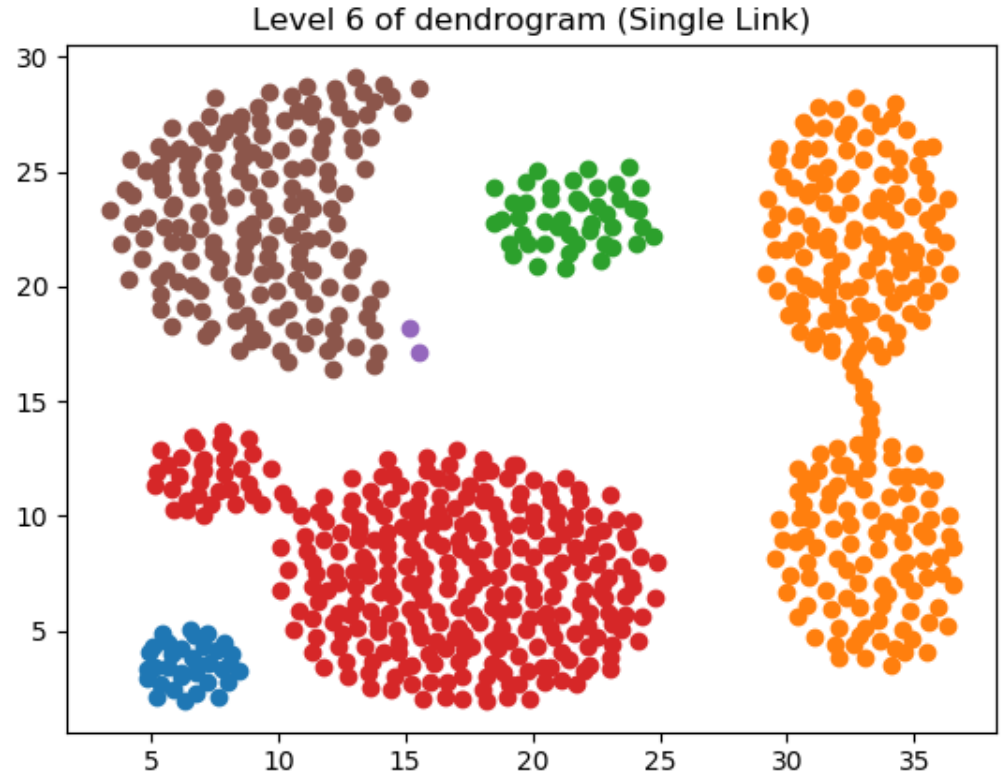

Level 1 of dendrogram (Single Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

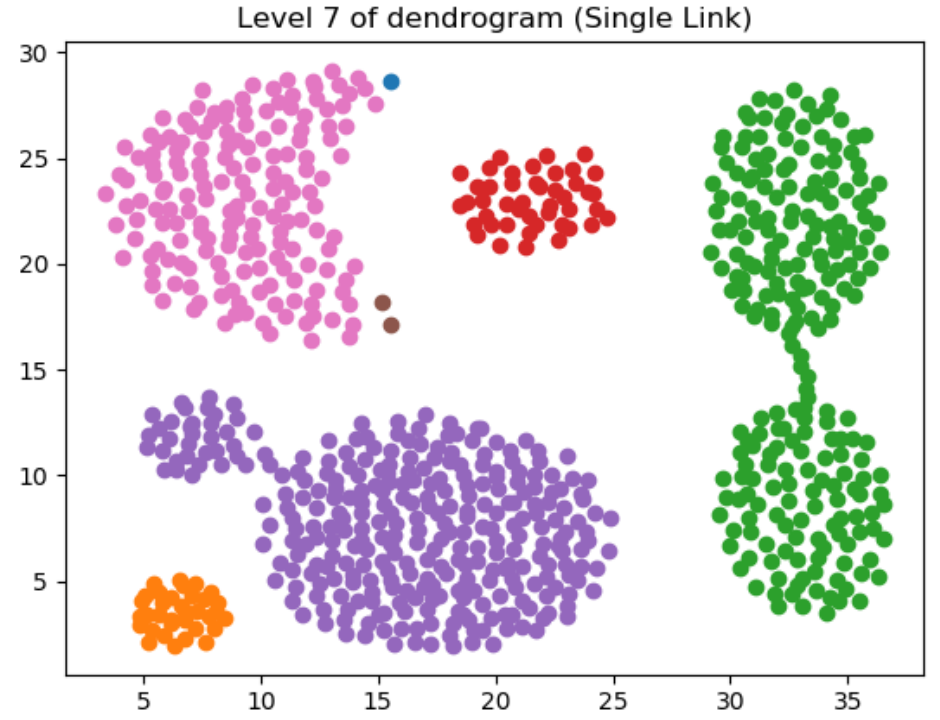

Level 2 of dendrogram (Single Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

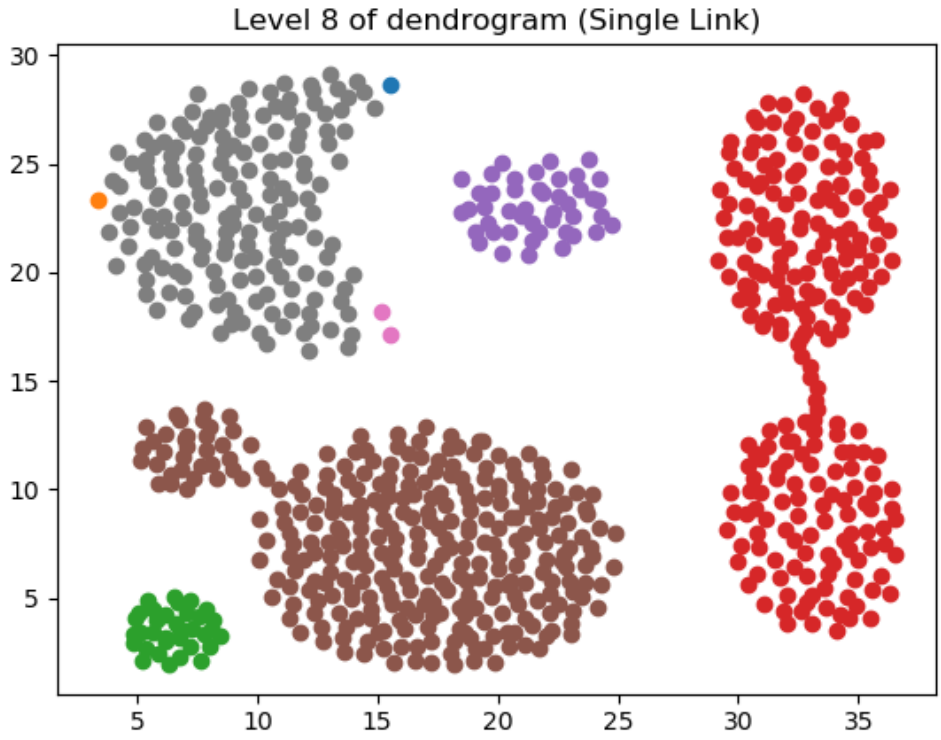

Level 3 of dendrogram (Single Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

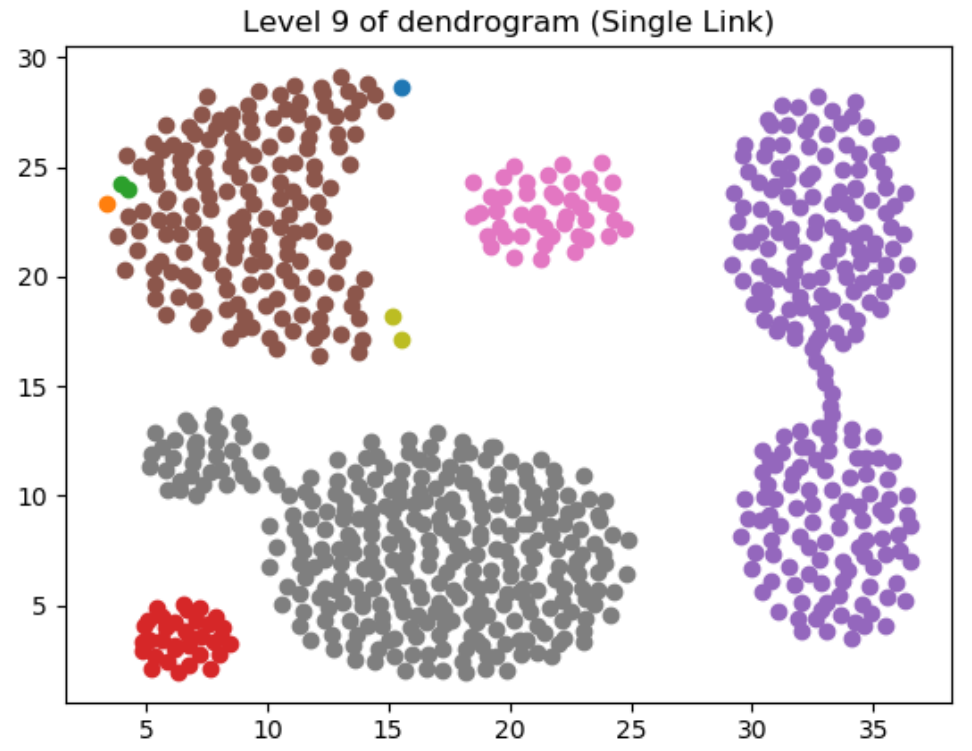

Level 4 of dendrogram (Single Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

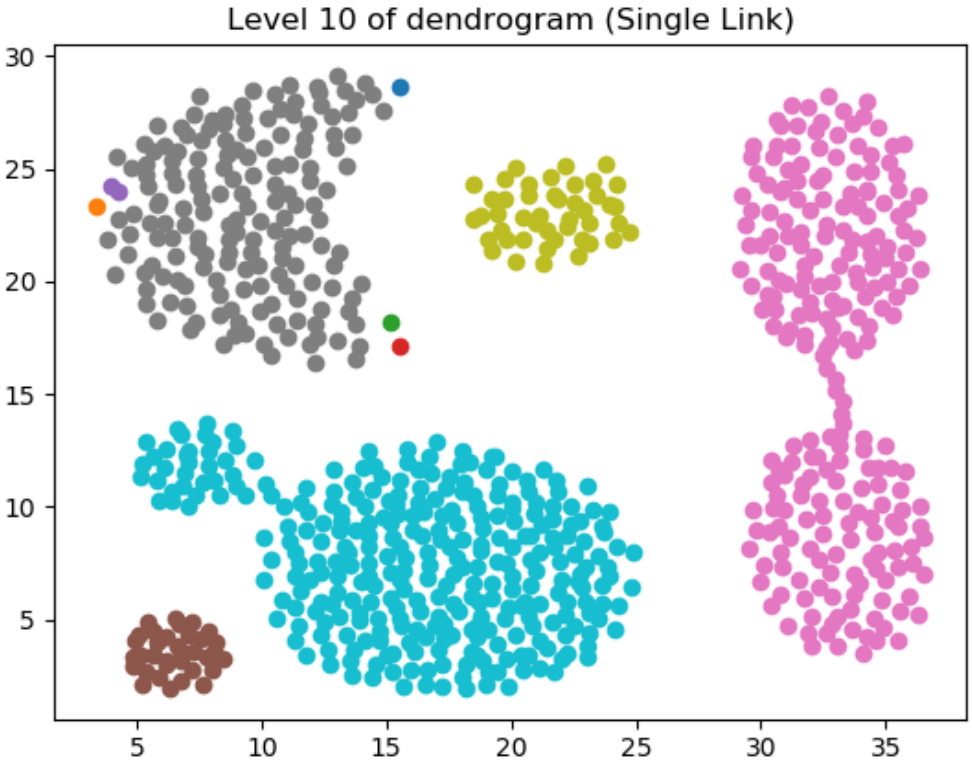

Level 5 of dendrogram (Single Link)
30
25
20
15
10
5
5
10
15
20
25
30
35


Level 6 of dendrogram (Single Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

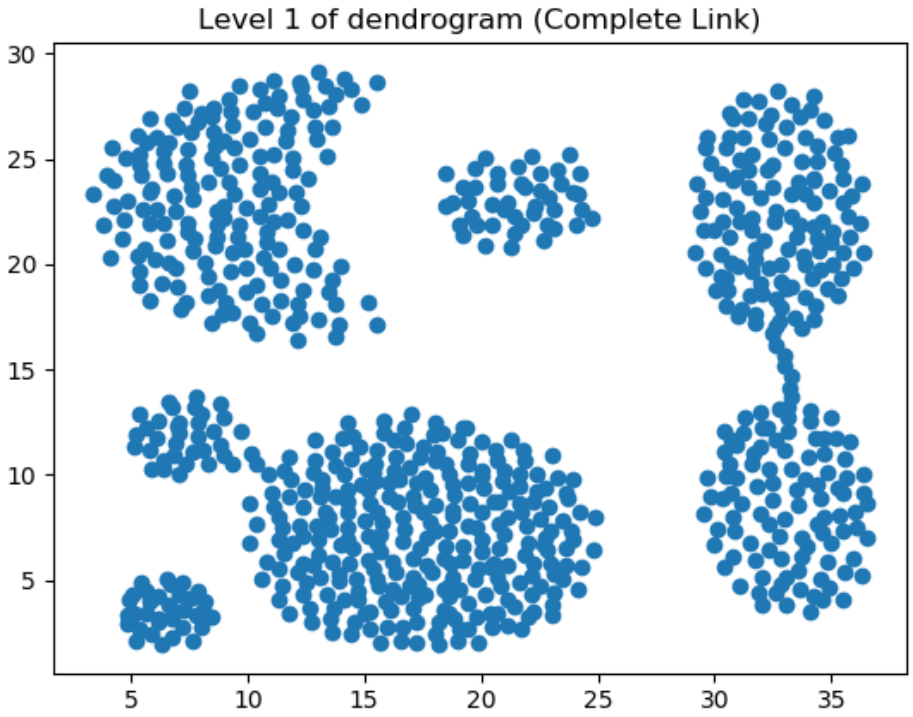

Level 7 of dendrogram (Single Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

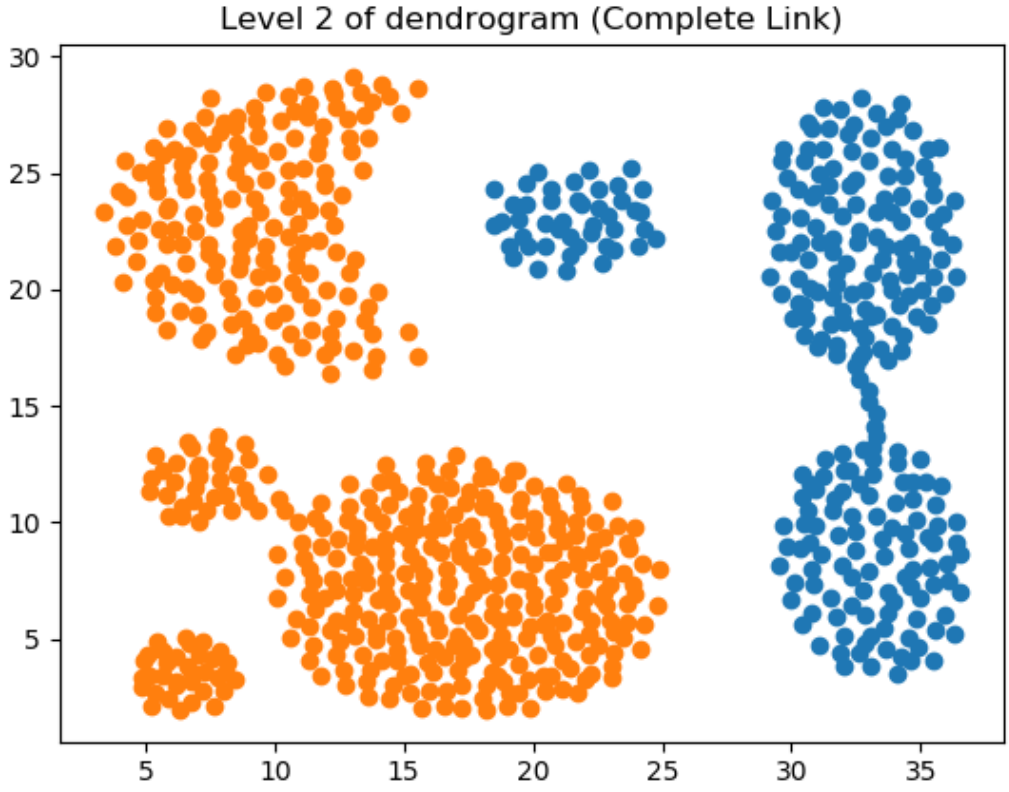

Level 8 of dendrogram (Single Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

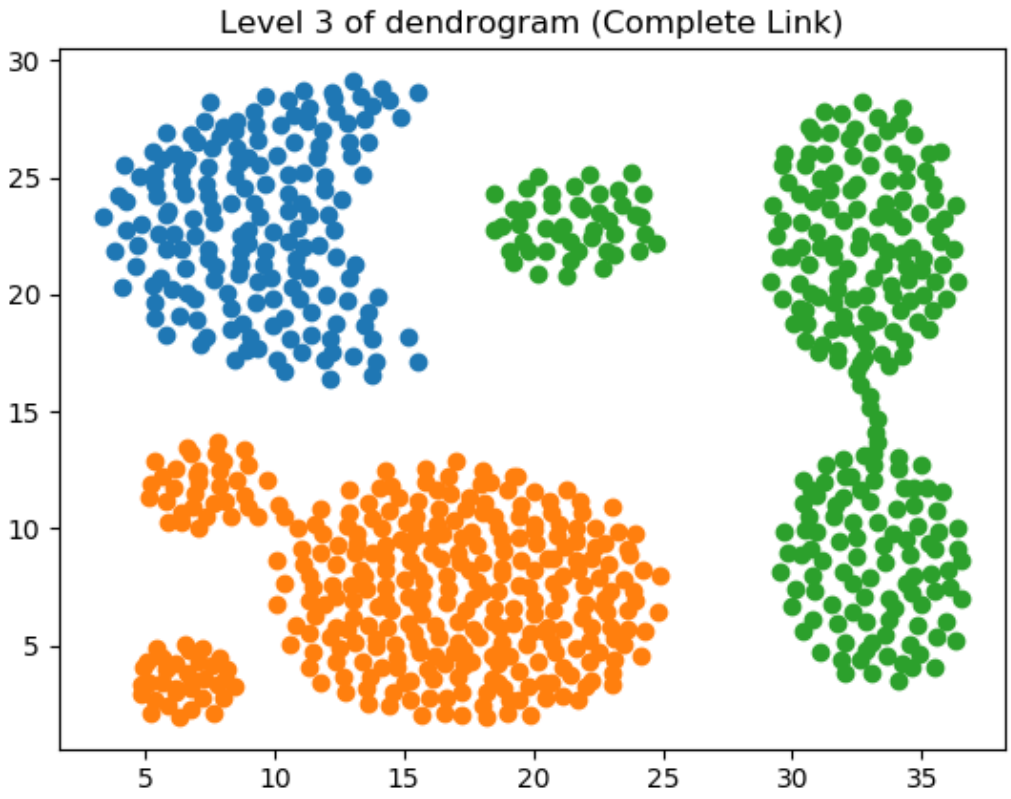

Level 9 of dendrogram (Single Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

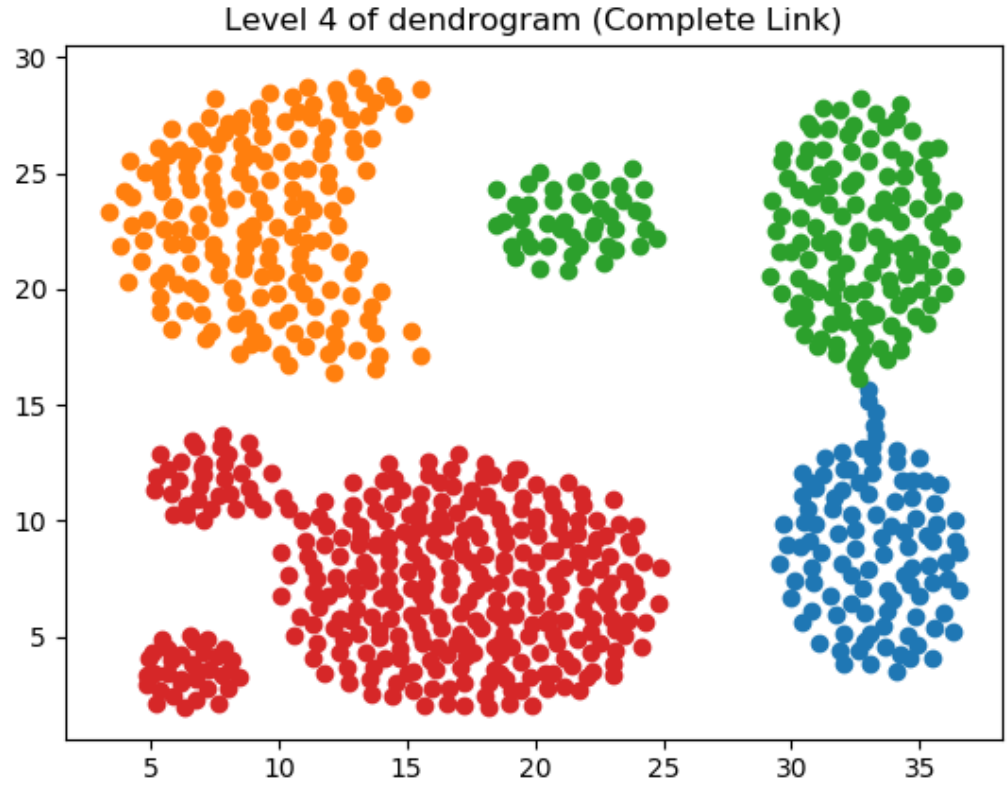

Level 10 of dendrogram (Single Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

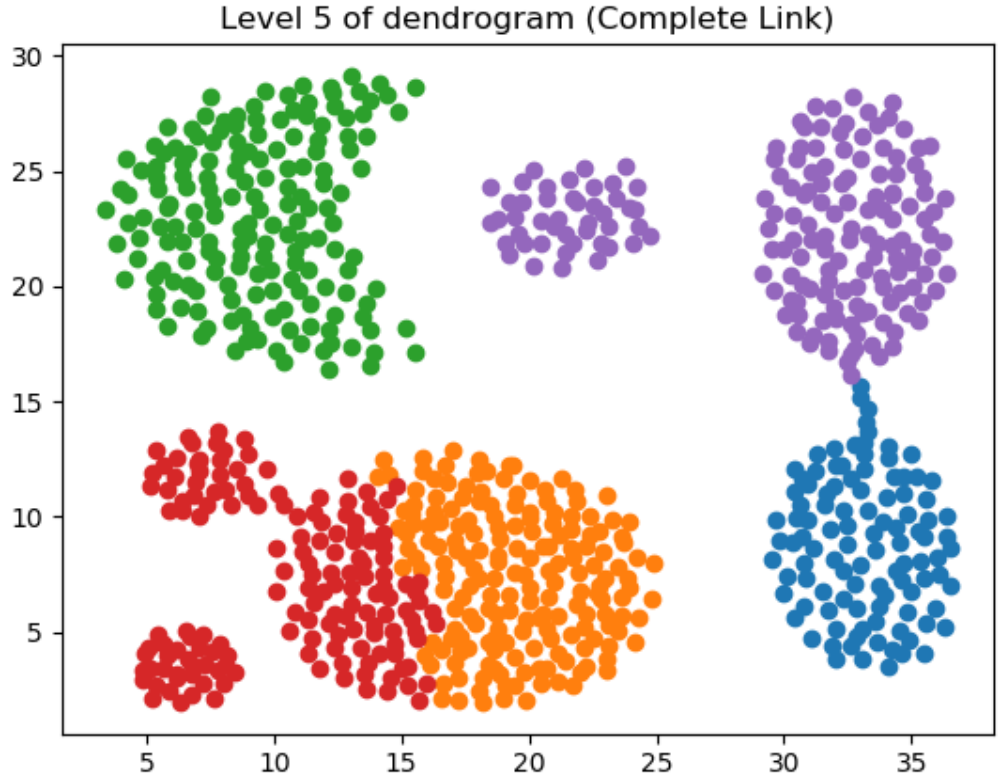
Level های یک تا ده Dendrogram الگوریتم خوشه بندی Complete Link:

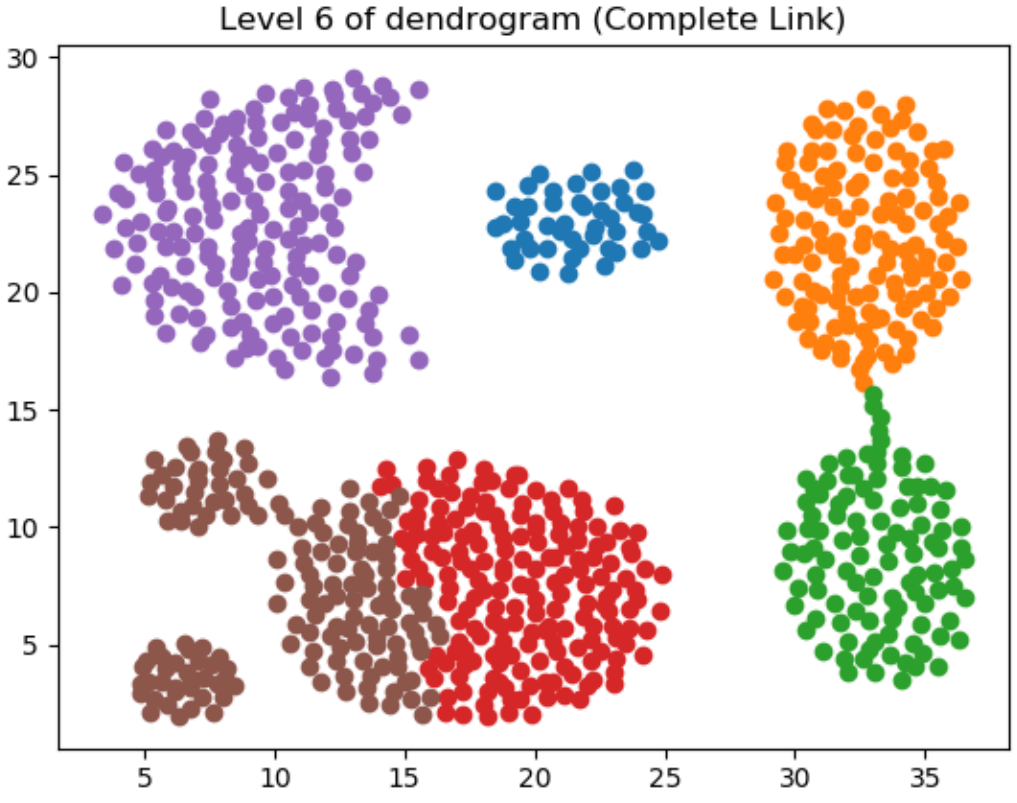


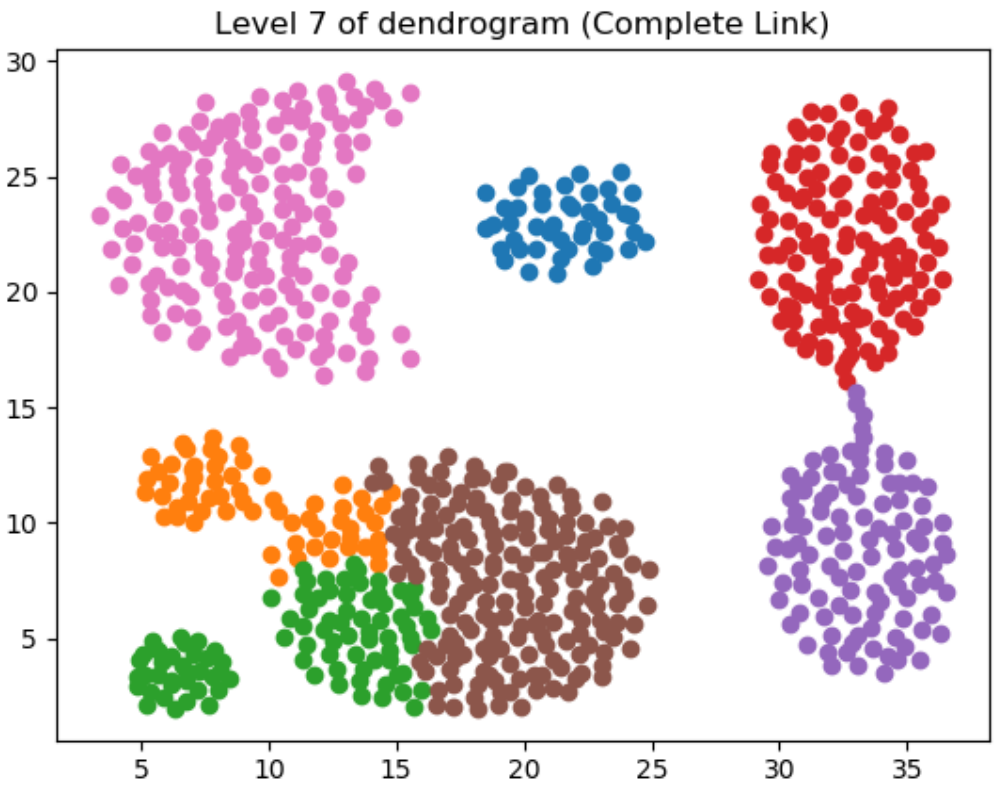


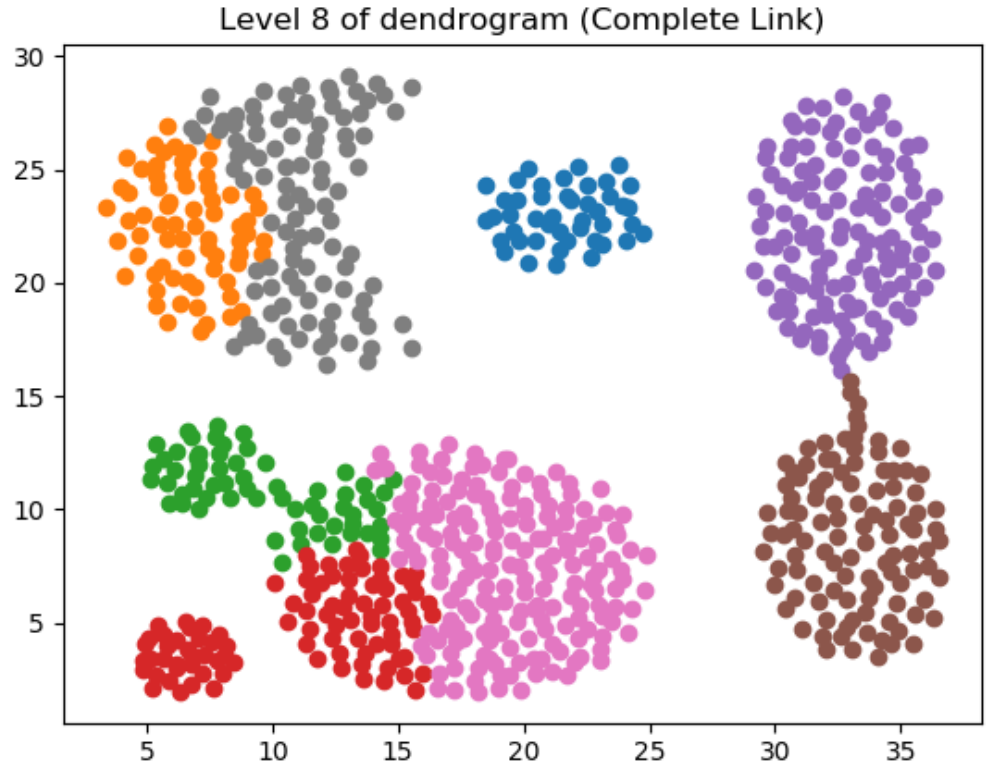

Level 3 of dendrogram (Complete Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

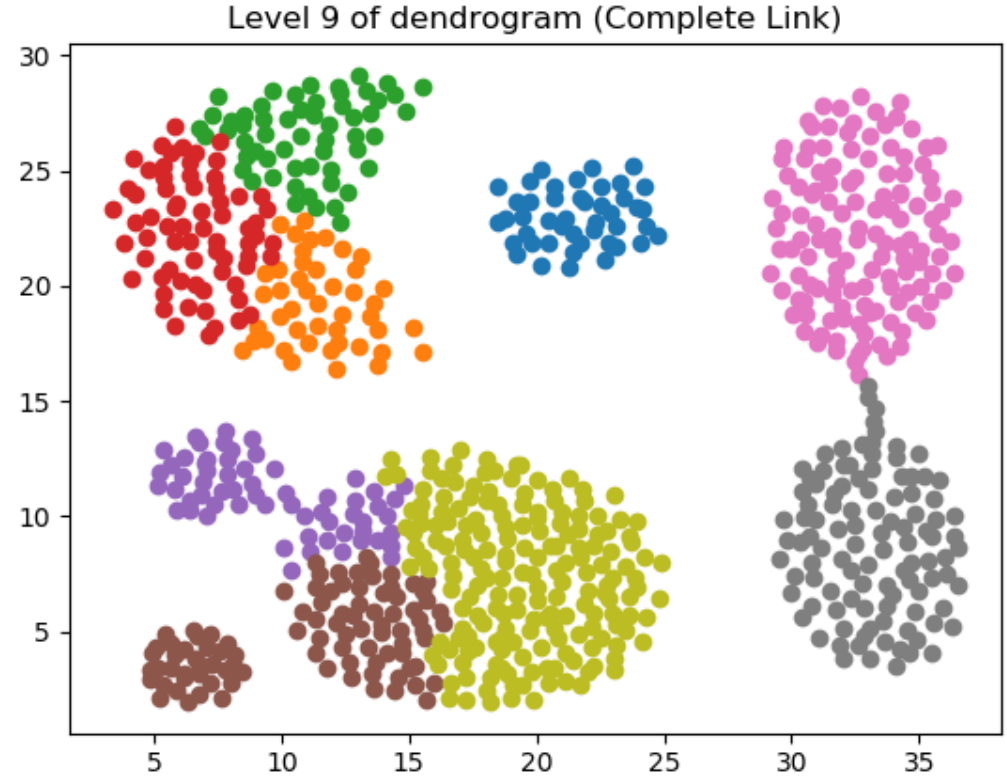

Level 4 of dendrogram (Complete Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

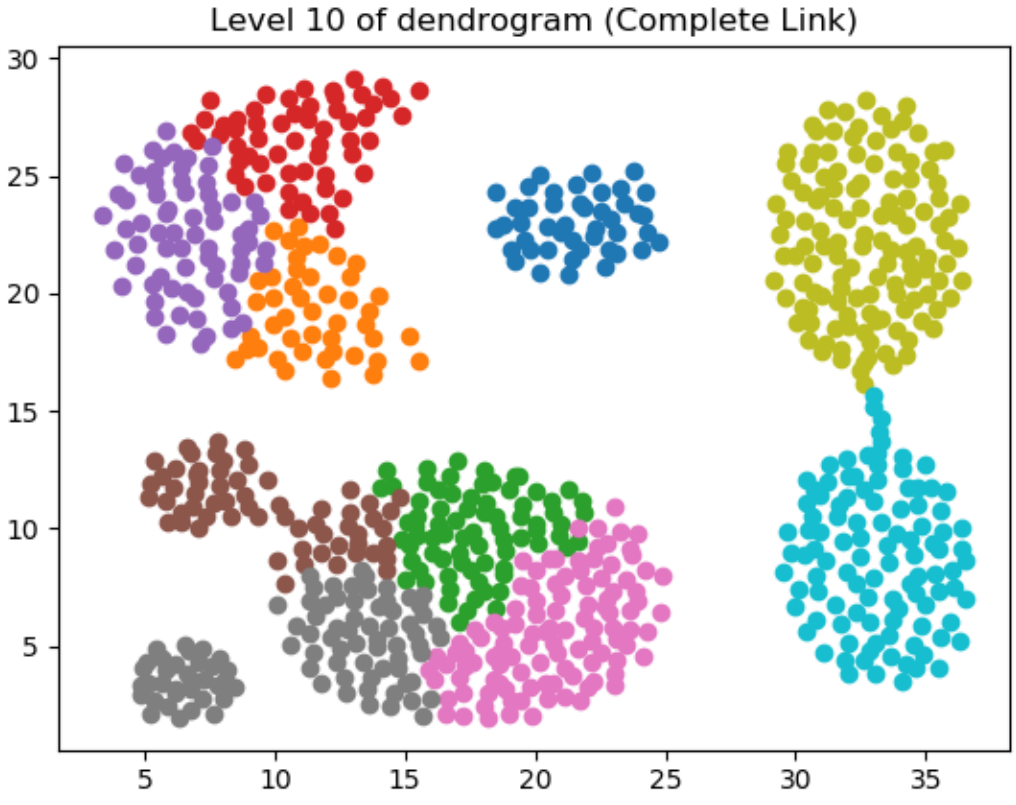

Level 5 of dendrogram (Complete Link)
30
25
20
15
10
5
5
10
15
20
25
30
35


Level 6 of dendrogram (Complete Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

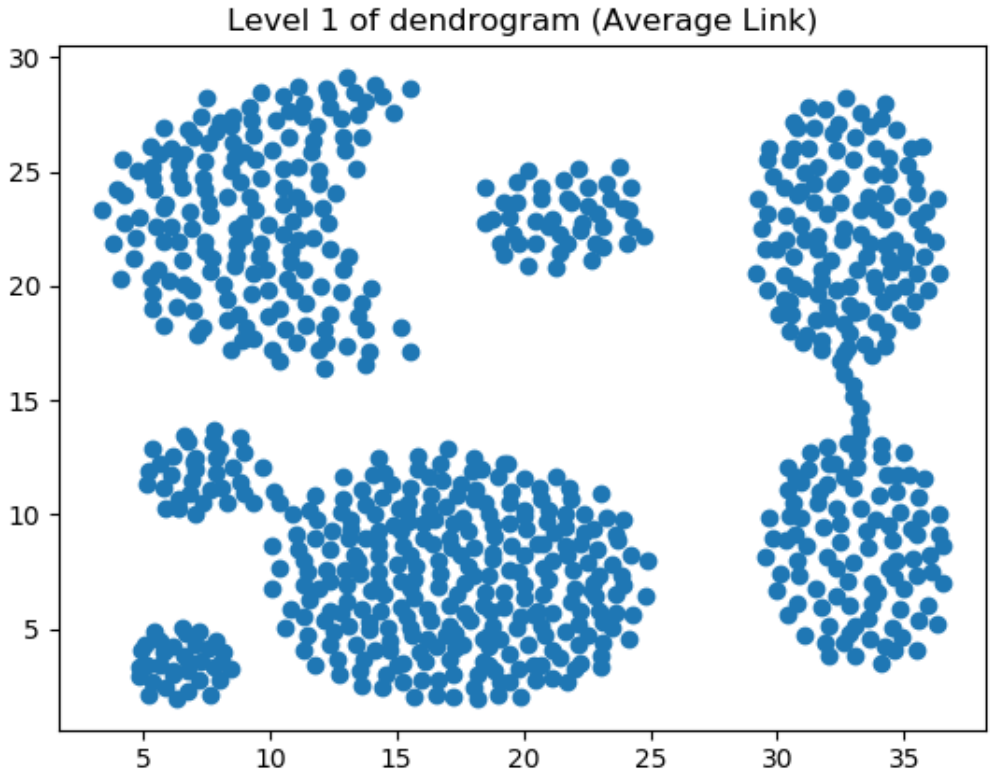

Level 7 of dendrogram (Complete Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

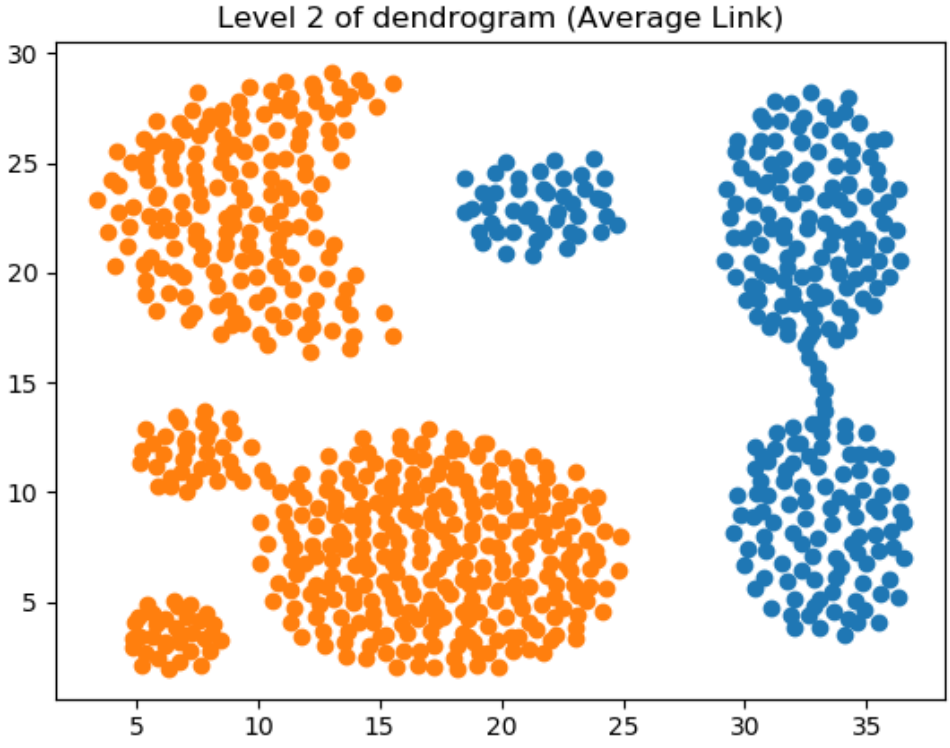

Level 8 of dendrogram (Complete Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

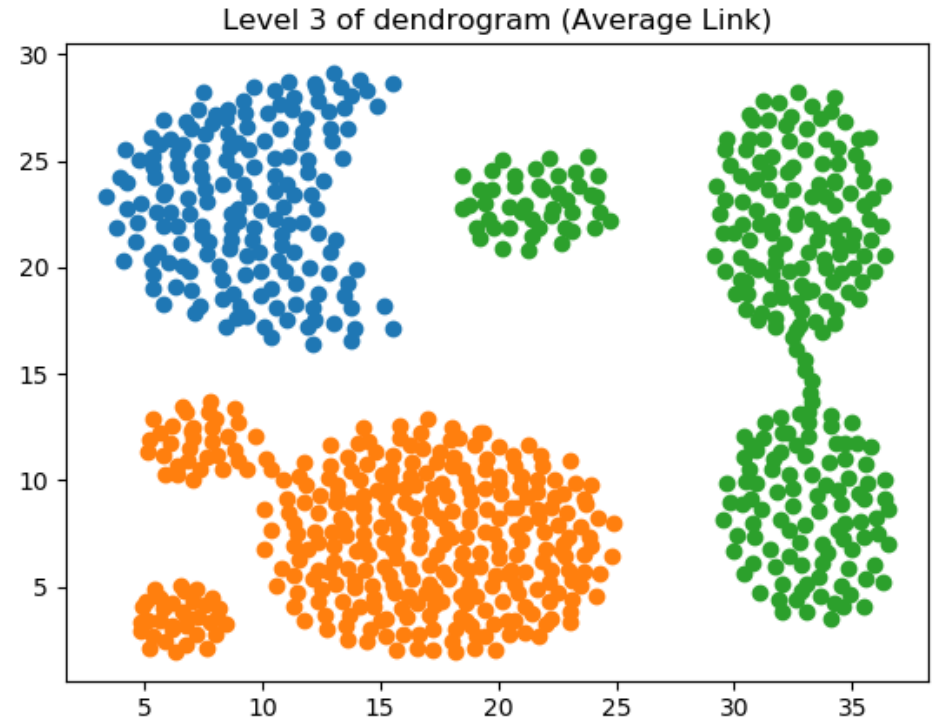

Level 9 of dendrogram (Complete Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

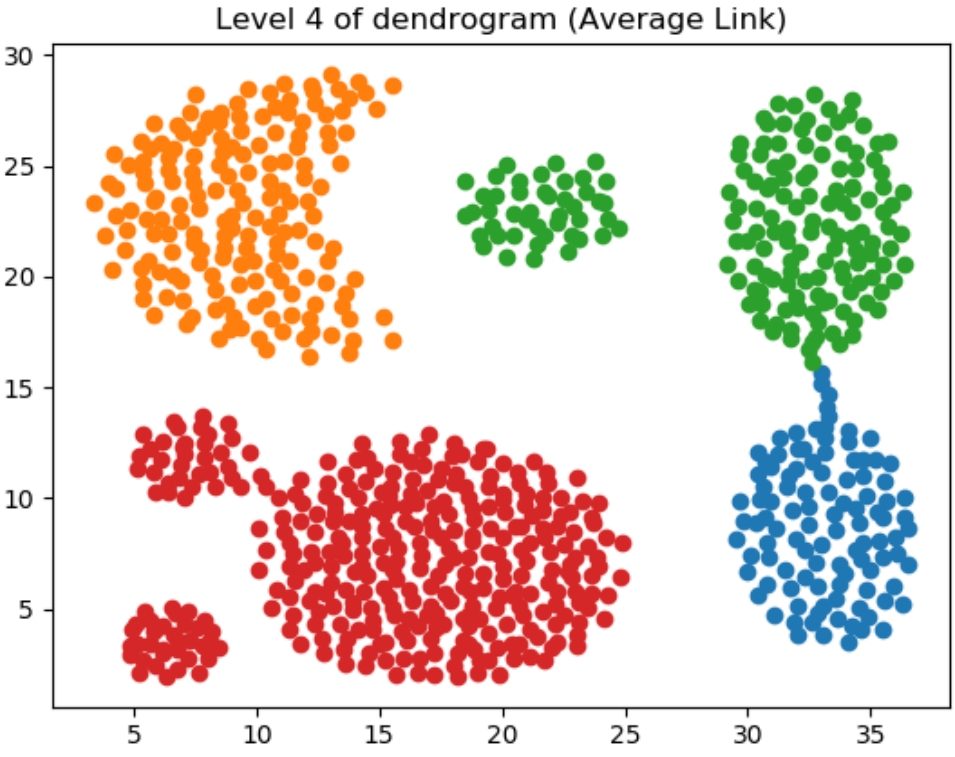

Level 10 of dendrogram (Complete Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

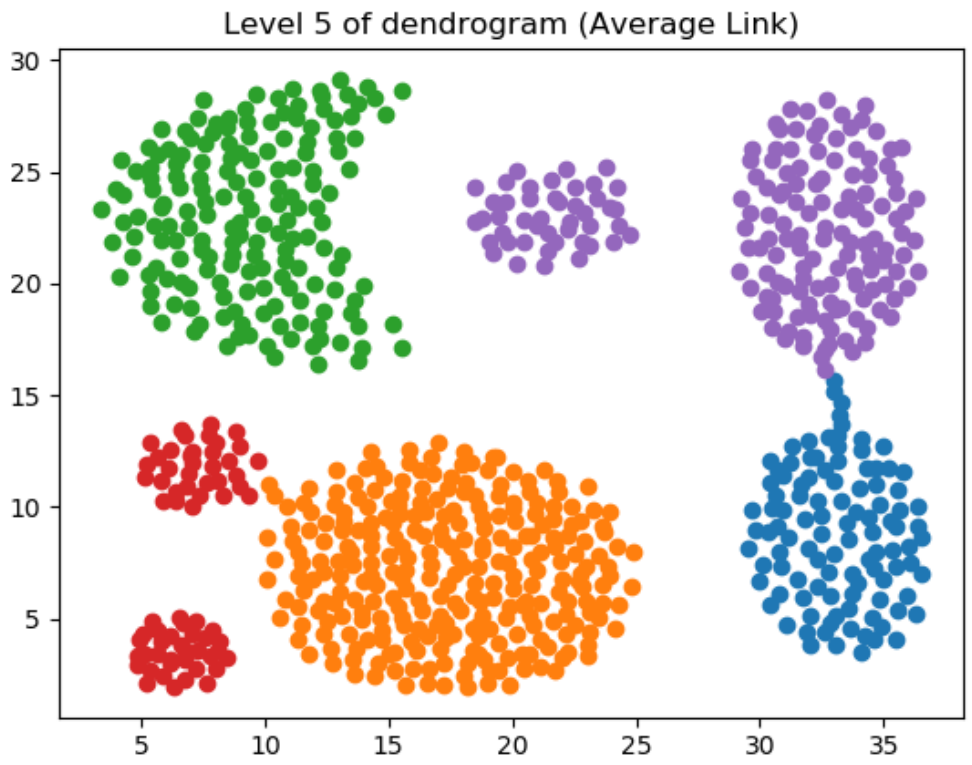
Level های یک تا ده Dendrogram الگوریتم خوشه بندی Average Link:

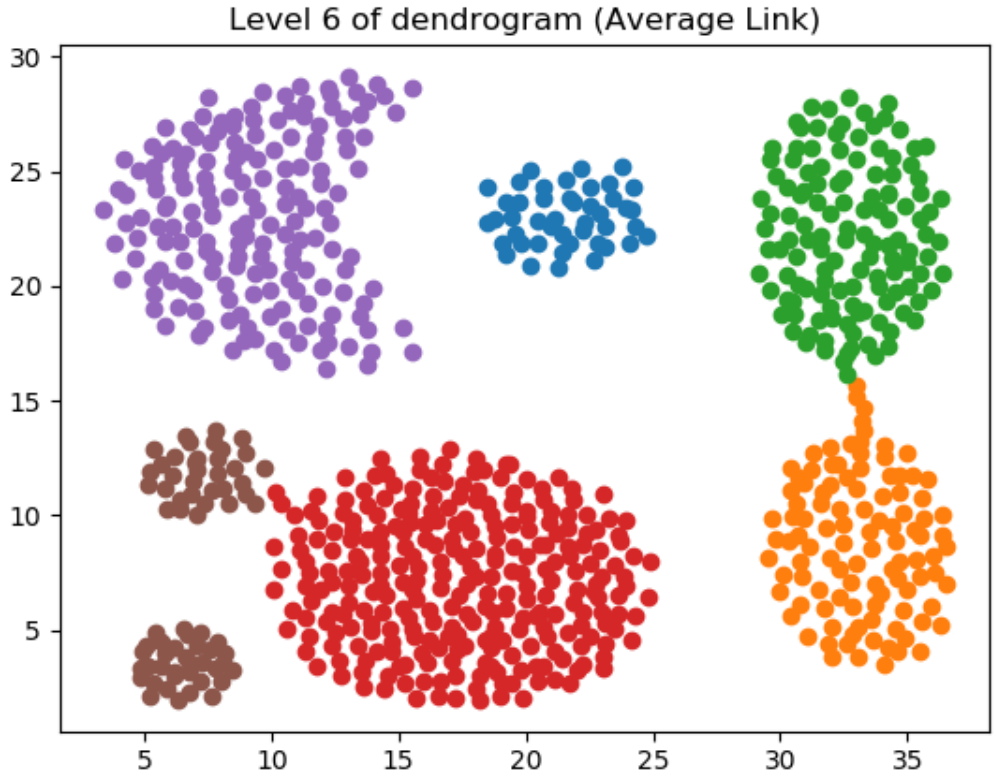


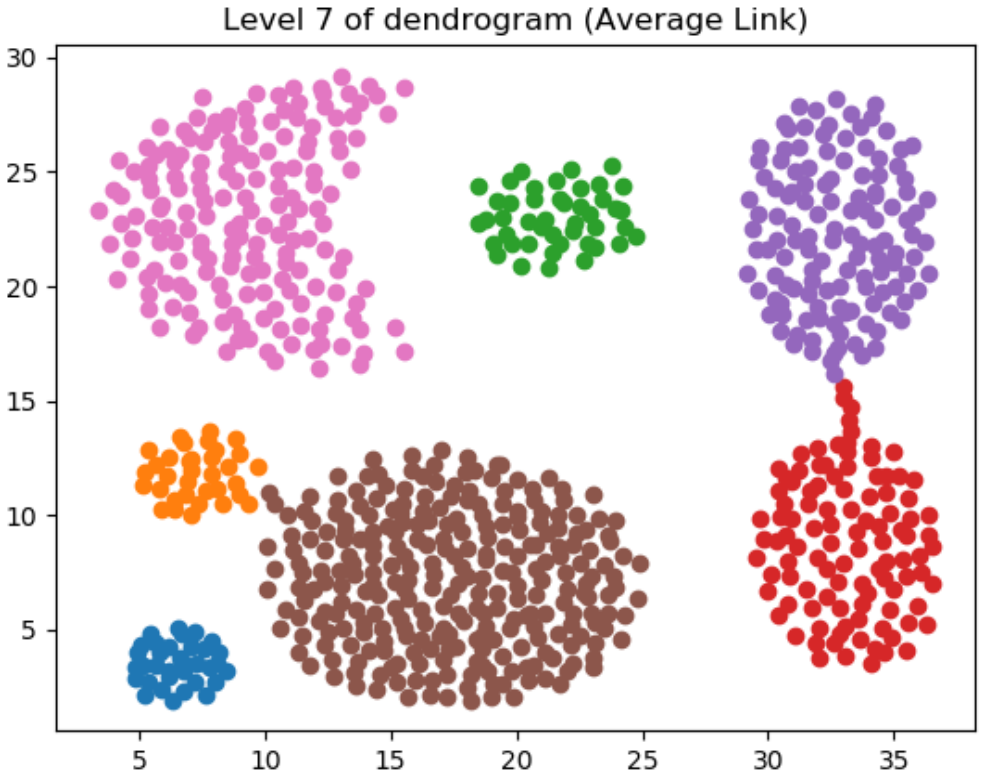


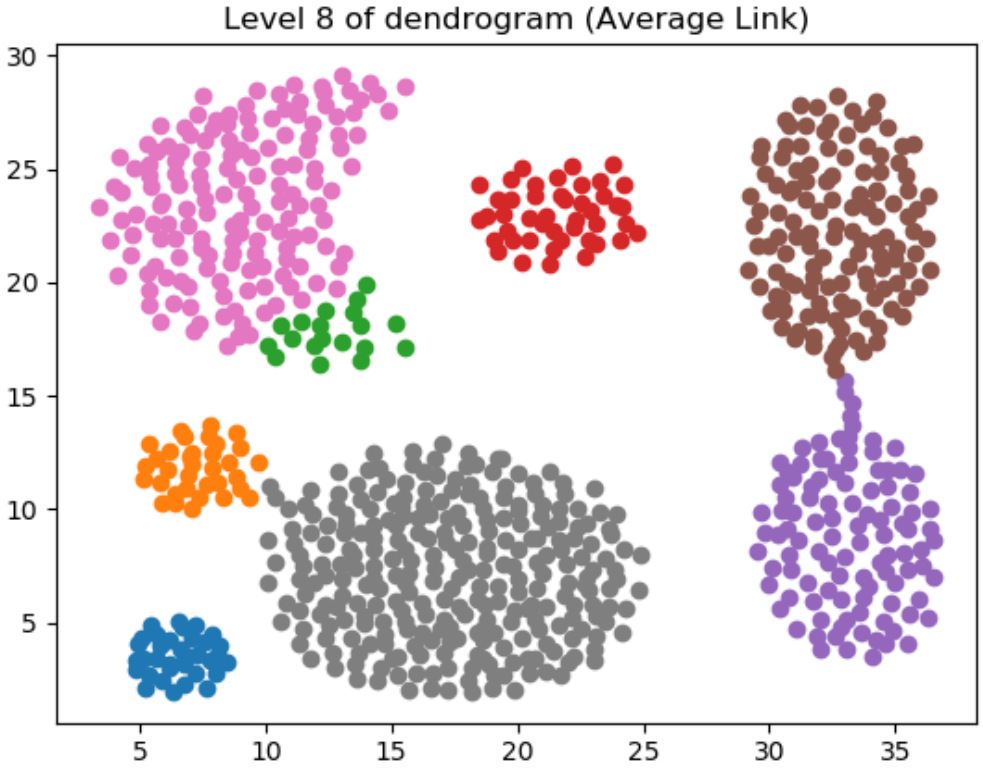

Level 3 of dendrogram (Average Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

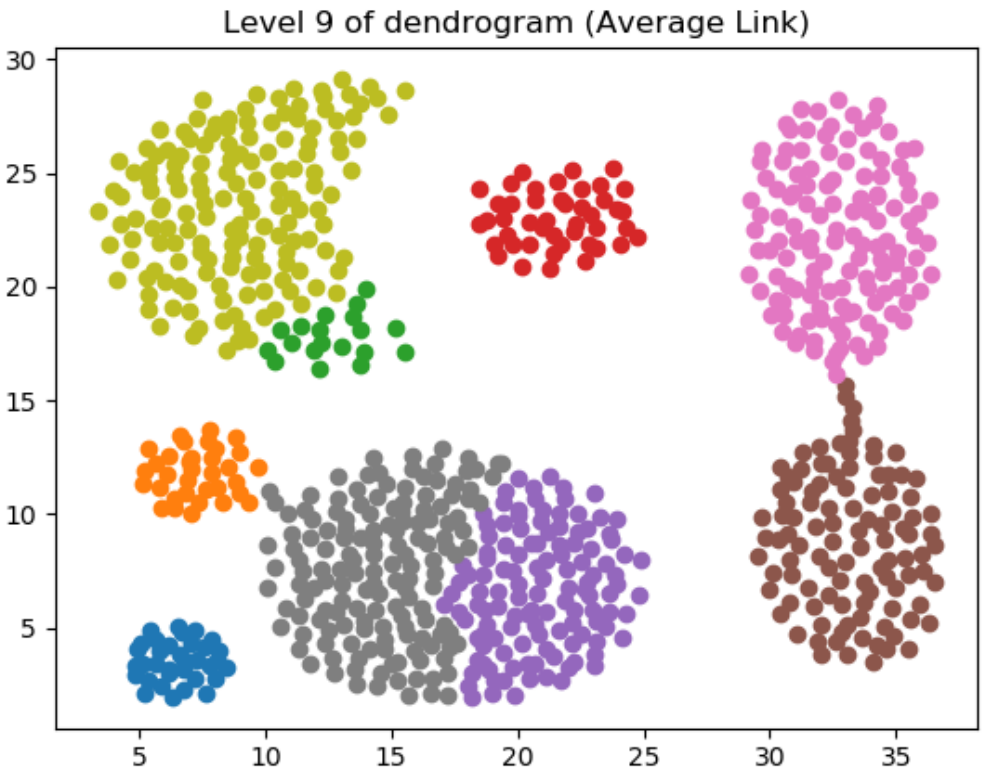

Level 4 of dendrogram (Average Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

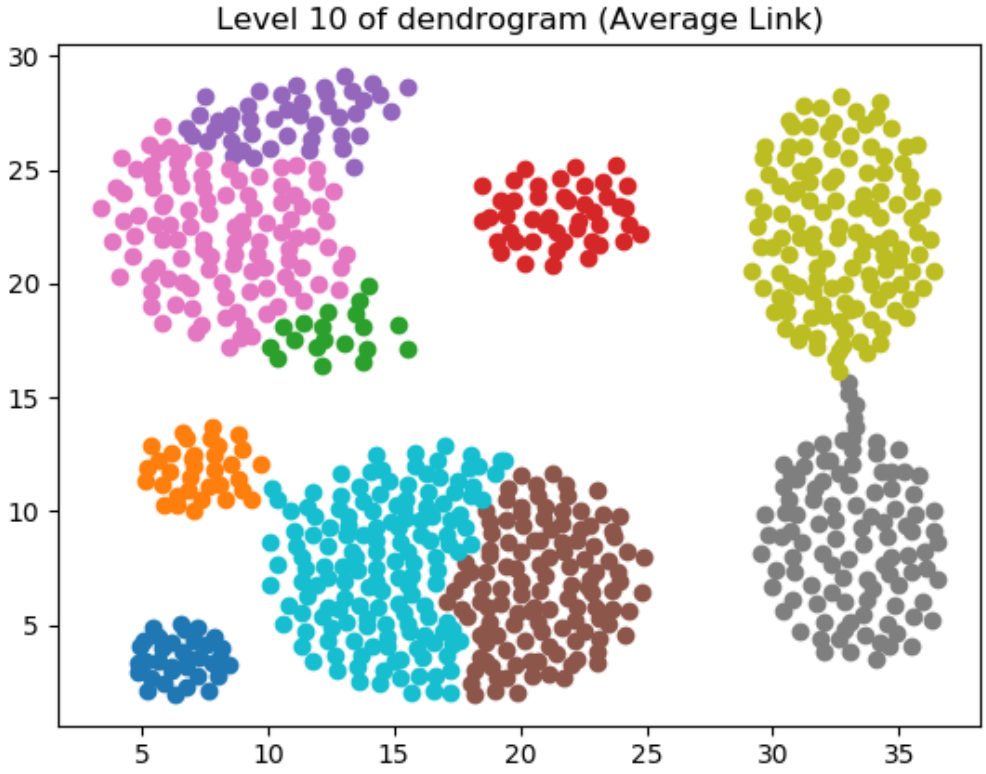

Level 5 of dendrogram (Average Link)
30
25
20
15
10
5
5
10
15
20
25
30
35


Level 6 of dendrogram (Average Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

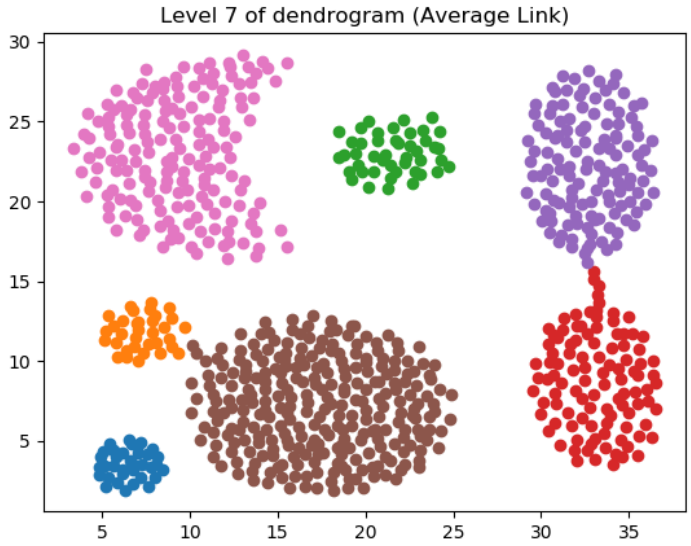

Level 7 of dendrogram (Average Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

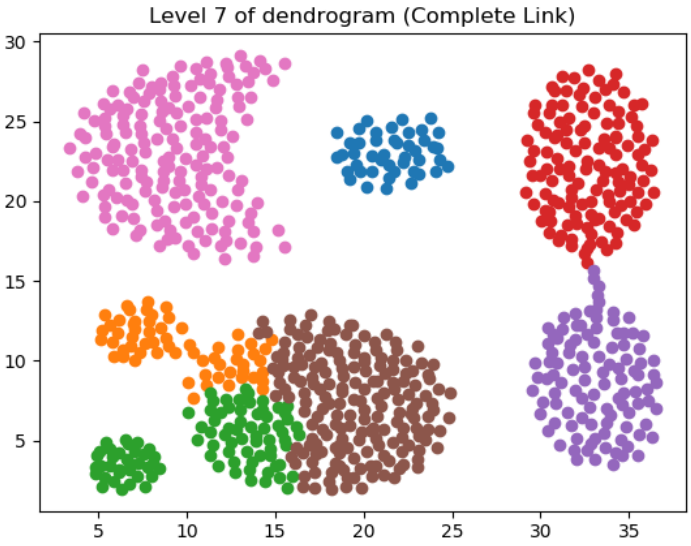

Level 8 of dendrogram (Average Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

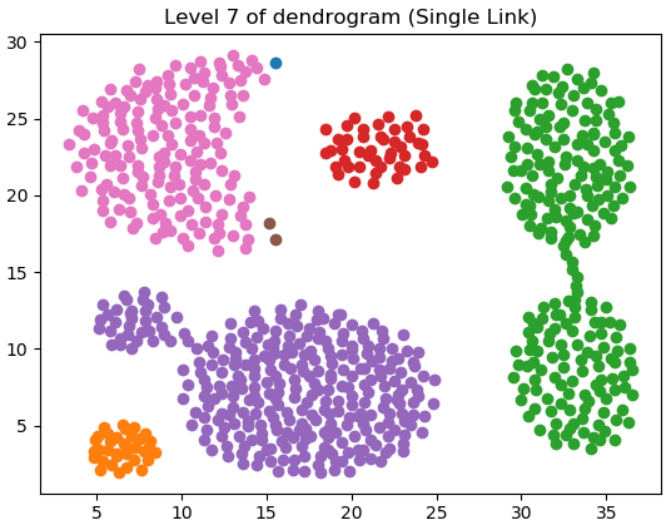

Level 9 of dendrogram (Average Link)
30
25
20
15
10
5
5
10
15
20
25
30
35


Level 10 of dendrogram (Average Link)
30
25
20
15
10
5
5
10
15
20
25
30
35

اگر به داده‌ها نگاه کنیم می‌توان آن‌ها را به ۷ قسمت جدا از هم تقسیم کرد که این کار را Average Link بهتر از آن دو انجام داده است، در زیر عمکرد سه روش برای سطح ۷ Dendrogram را مشاهده می‌کنید:

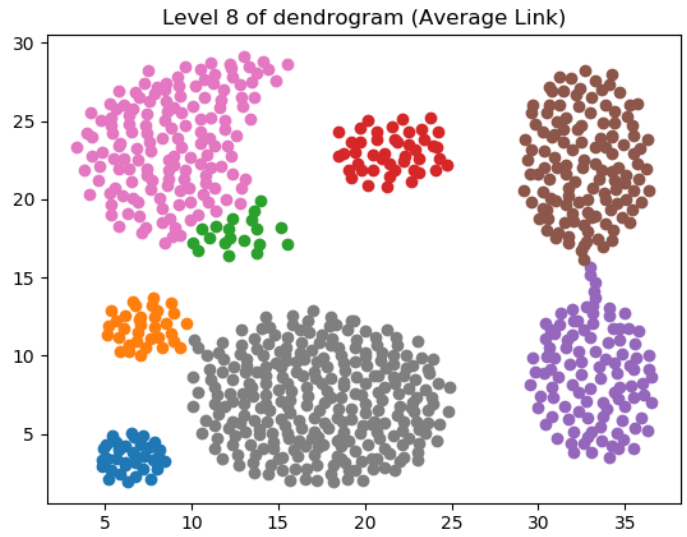


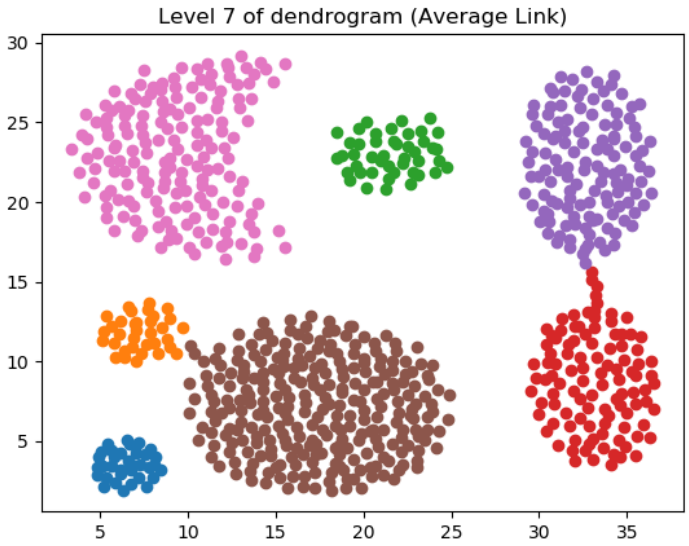




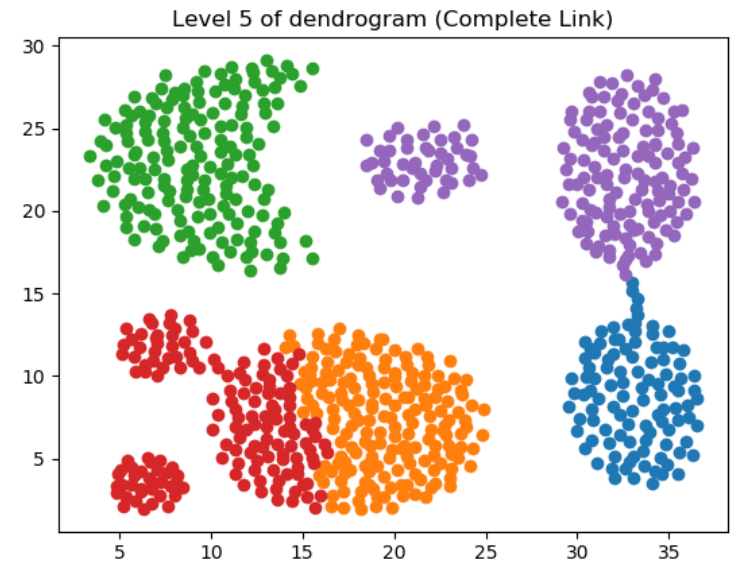
در بین سه روش Complete Link, Average Link و Single Link بر اساس نتیجه‌های به دست آمده Average Link بهتر از آن دو عمل کرده است. در Average Link از سطح هفت Dendrogram داده‌های جدا از هم، به صورت جداگانه‌ای خوشه بندی شده اند در حالی که این اتفاق در Complete Link در سطح چهار Dendrogram رخ داده است:

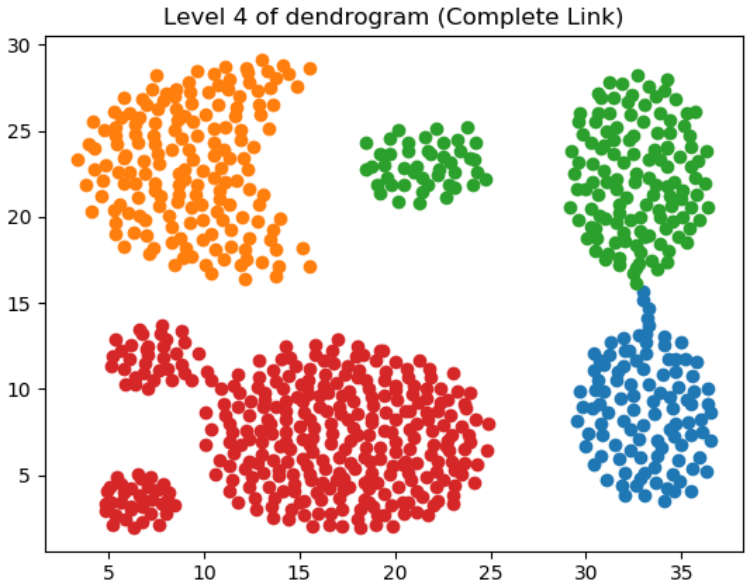
در Average Link بعد از سطح هشت، در سطح هفت داده‌های جدا از هم، به صورت جداگانه‌ای دسته‌بندی شده‌اند:





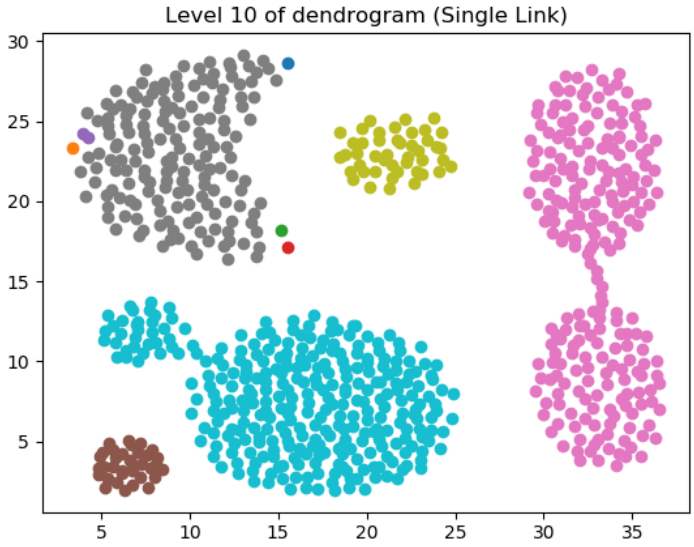
در Complete Link وقتی از سطح ۵ Dendrogram به سطح ۴ می‌رویم داده‌های جدا از هم، به صورت جدا خوشه‌بندی می‌شوند در حالی که در سطح ۵ و قبل از آن داده‌های جدا از هم، کاملا به صورت مجزا خوشه‌بندی نشده‌اند.

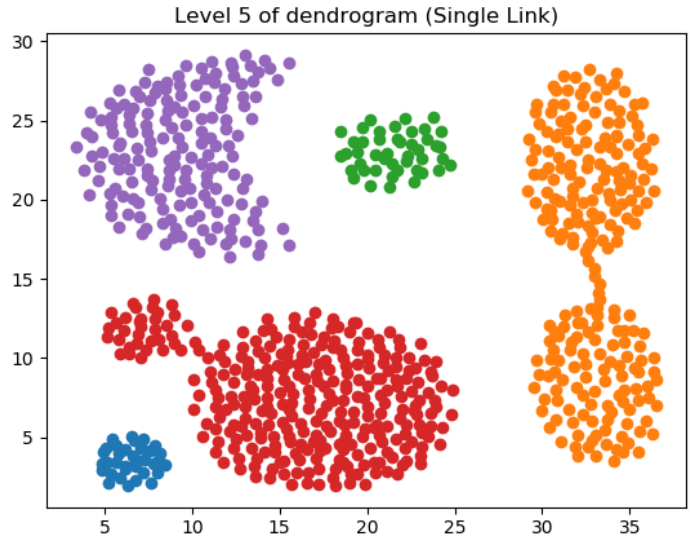




در حالی که این اتفاق در سطح ۷ Average Link رخ می‌دهد.

در Single Link از سطح ۱۰ تقریبا داده‌های جدا از هم به صورت مجزایی خوشه‌بندی شده اند ولی خوشه‌های کوچکی وجود دارند که تا سطح ۶ این خوشه‌های کوچک حضور دارند و از سطح ۵ و پنج به بعد آن‌ها مشاهده نمی‌شوند:





در Complete Link و Average Link اختلاف بزرگی خوشه‌های بزرگ و کوچک نسبت به Single Link بسیار کمتر است و سایز خوشه‌ها در آن دو الگوریتم به هم نزدیک‌تر هستند در حالی که در Single Link در بعضی از سطح‌های بین یک تا ده، خوشه‌هایی با یک نمونه مشاهده می‌شود.