Pattern Recognition

Second Assignment

Dr. Ahmad Ali Abin

writer:Mahdie Dolatabadi

Student Id:400443077

Faculty of Computer Engineering

مقدمه:

DBSCAN یک روش خوشه بندی مبتنی براتراکم است که خوشه هارا به عنوان مناطقی با چگالی بالا میبیند که توسط مناطقی با چگالی کم از هم جدا شده‌اند. برخلاف K-means، که مجموع مربع های درون خوشه ای را به حداقل می‌رساند و طی این روند بیشتر اشکال محدب را تشخیص می دهد، DBSCAN میتواند خوشه هارا با هر شکلی پیدا کند.

این روش با دو پارامتر کار میکند:

i. minPts:

تعداد نقاطی که لازم است در همسایگی یک نقطه باشند تا آن نقطه به عنوان نمونه اصلی در نظر گرفته شود. در واقع این پارامتر میزان تحمل الگوریتم را در برابر نویز کنترل می‌کند.

ii. eps

بیشترین فاصله بین دو نقطه که به عنوان همسایگی در نظر گرفته می شود.

اگر این دو پارامتر را اشتباها مقداردهی کنیم ممکن است باعث خطای الگوریتم شود. یه این معنی که یا یک دیتای واقعی که عضو یک خوشه است را به عنوان نویز در نظر بگیرد یا نویز را خوشه بندی کند.

جز اصلی DBSCAN همین نمونه های اصلی هستند. نمونه های اصلی حداقل به تعداد minPts داده با فاصله کمتر یا مساوی eps در همسایگی خود دارند. هر خوشه مجموعه ای از نمونه های اصلی است که می‌توان با گرفتن یک نمونه هسته به صورت بازگشتی، یافتن همه همسایه‌هایی که نمونه‌های اصلی هستند و سپس یافتن نمونه‌های هسته همسایشان و ... ایجاد می‌شود. هر خوشه همچنین دارای مجموعه ای از نمونه های غیر اصلیند که در واقع فقط همسایه های نمونه های اصلی هستند و خود شرایط نمونه اصلی بودن را ندارند. به طور شهودی این نمونه ها معمولا در لبه یک خوشه قرار می‌گیرند و نقاط مرزی هستند.

مزایای DBSCAN:

- ✓ این الگوریتم قبل از اجرا نیازی به دانستن تعداد خوشه‌ها ندارد.
- ✓ با خوشه‌هایی با اشکال دلخواه به خوبی عمل می‌کند.
- ✓ نویز را پیدا می‌کند و نسبت ب موارد دور افتاده قوی است.

معایب DBSCAN:

- ✓ به دلیل اینکه یک eps برای کل داده ها تعریف می‌شود، نمی‌تواند خوشه‌ها را با چگالی متفاوت تشخیص دهد.
- ✓ به دلیل اینکه اطلاعی از داده‌های ورودی نداریم انتخاب معنی‌دار eps کار سختی است.
- ✓ ونهایتا نتایجش کاملا قطعی نیست زیرا الگوریتم کارش را از یک نقطه به صورت رندوم شروع می‌کند.

توضیح کد:

برای این تمرین دو قسمت جداگانه کد زده‌ام.قسنت اول دیتاست‌هایی ساخته شده‌اند که در آن ها خوشه‌ها دارای چگالی برابر هستند و DBSCAN برای تشخیص این خوشه ها از یکدیگر به مشکل نمی‌خورد.اما در قسمت دوم نمونه‌ها یا دارای نویز هستند و یا چگالی خوشه‌ها در آن‌ها متفاوت است که در ادامه توضیح خواهیم‌داد چگونه این مساله را حل کرده‌ایم.

قسمت اول:

دیتاست‌های مورد نیاز را فرامیخوانیم. همانطور که در گزارش‌های قبلی گفته شده:

- numpy برای محاسبات سنگین ریاضی (مانند ضرب دو ماتریس n*n) استفاده می‌شود
- matplotlib برای ترسیم نمودار
- sklearnکاربردهای متفاوتی دارد که در اینجا از ماژول‌های DBSCAN برای پیاده‌سازی الگوریتم، از datasets برای ساخت نمونه های ورودی و از standardScaler برای فیت کردن داده‌ها استفاده می‌شود.

DBSCAN is a density-based unsupervised machine learning algorithm to clustring

```
[ ]  import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.cluster import DBSCAN
     from sklearn import metrics
     from sklearn.datasets import make_blobs
     from sklearn.preprocessing import StandardScaler
```

در این قسمت دیتاستی به شکل حباب به مرکزیت‌های (3,3), (1,1) ساخته‌شده:

Blobs data

Generate sample data

```
[ ]  # Generate sample data
     centers = [[1,1], [3,3]]
     X, labels_true = make_blobs(
         n_samples=750, centers=centers, cluster_std=0.4, random_state=0)

     X = StandardScaler().fit_transform(X)
```

وسپس الگوریتم DBSCAN برروی آن اجرا شده‌است.

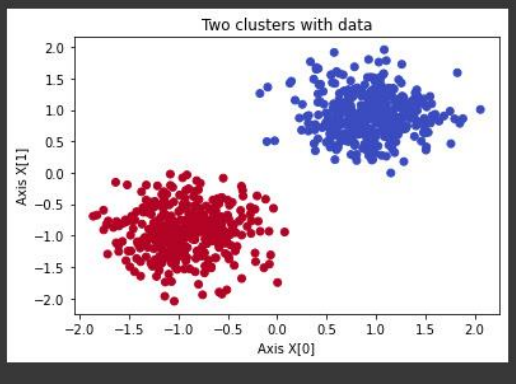
Compute DBSCAN

```
[ ]  db = DBSCAN(eps=0.5, min_samples=10).fit(X)
```

در نمودار زیر می‌بینیم که به خوبی DBSCAN این دو خوشه را از هم جدا کرده.

plotting clustering data

```
[ ]  colors = list(map(lambda x: '#3b4cc0' if x == 1 else '#b40426', labels))
     plt.scatter(X[:,0], X[:,1], c=colors, marker="o", picker=True)
     plt.title('Two clusters with data')
     plt.xlabel('Axis X[0]')
     plt.ylabel('Axis X[1]')
     plt.show()
```



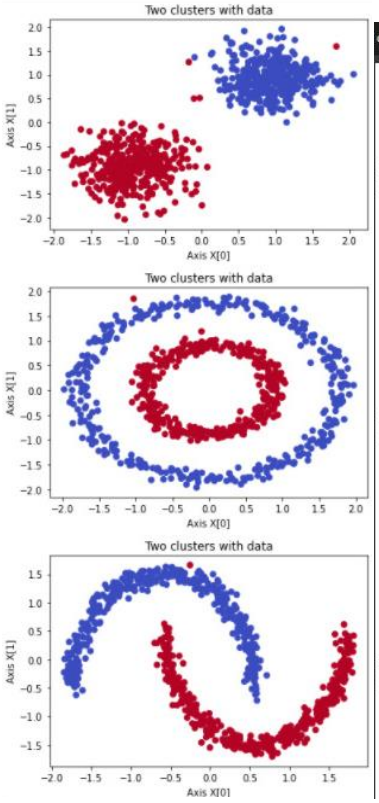
حال سه نوع دیتاست با پراکندگی‌های متفاوت حباب شکل،ماه و دایره ساخته‌ایم:

```
[ ]  from sklearn import datasets ,cluster
     n_samples = 750

[ ]  blobs = datasets.make_blobs(n_samples=n_samples, centers=centers,cluster_std=0.4, random_state=0)
     noisy_circles = datasets.make_circles(n_samples=n_samples, noise = 0.05, factor = 0.5)
     noisy_moons = datasets.make_moons(n_samples = n_samples,noise = 0.05, random_state= 1 )

[ ]  eps0 =np.array([0.5, 0.2, 0.5])
     eps1 =np.array([0.73, 0.43, 0.56])
     eps2 =np.array([0.3, 0.19, 0.15])
     eps3 =np.array([0.2, 0.15, 0.1])
     eps4 =np.array([0.8, 0.5, 0.6])
     datasets = [
                blobs,
         noisy_circles,
         noisy_moons,
     ]
```

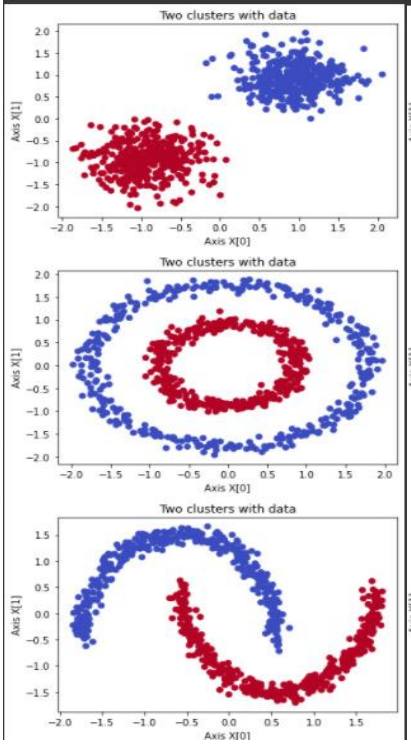
سپس با فاصله همسایگی‌های متفاوت الگوریتم DBSCANرا بر روی اینها اجرا کرده و نتایج خوشه‌بندی را در صفحه بعد رسم-کرده‌ایم:

```
[ ]  def dbscan(datasets,eps):
       for i_dataset, dataset in enumerate(datasets):

         X, y = dataset
         # normalize dataset
         X = StandardScaler().fit_transform(X)
         dbscan = cluster.DBSCAN(eps=eps[i_dataset],min_samples=5).fit(X)
         y = dbscan.labels_.astype(np.int)
         colors = list(map(lambda x: '#3b4cc0' if x == 1 else '#b40426', y))
         plt.scatter(X[:,0], X[:,1], c=colors, marker="o", picker=True)
         plt.title('Two clusters with data')
         plt.xlabel('Axis X[0]')
         plt.ylabel('Axis X[1]')
         plt.show()
```

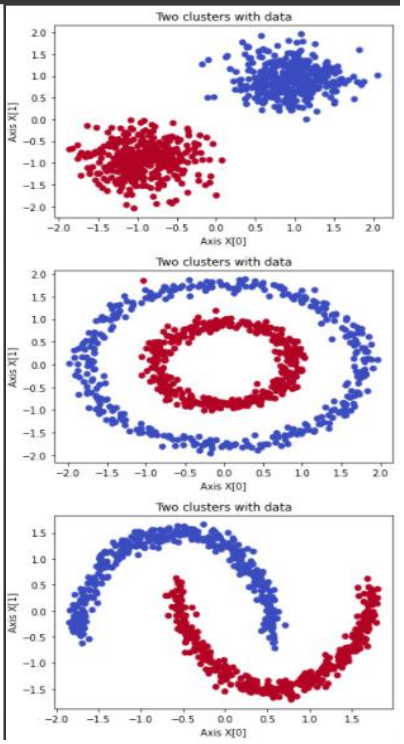
```
dbscan(datasets,eps2)
```

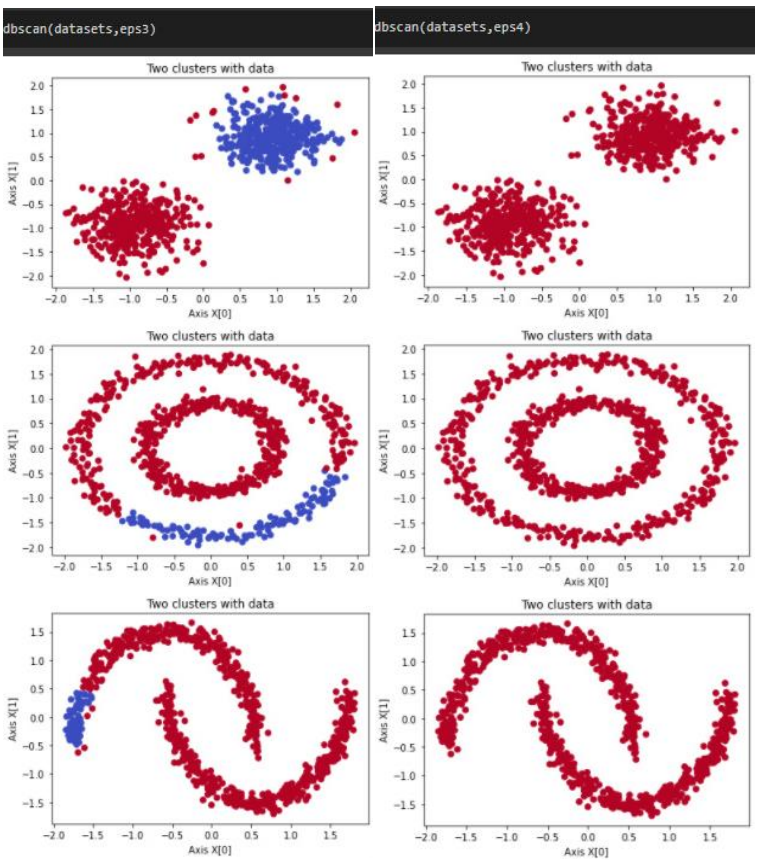



```
dbscan(datasets,eps1)
```




```
dbscan(datasets,eps0)
```





می‌بینیم که DBSCAN برای دیتا ست اول(حباب) تا مرز eps = 0.73، دومی(دایره) eps = [0.19,0.43] و اخری(ماه) eps = [0.15.0.56] به درستی جواب می‌دهد. البته تین کرزها به صورت دستی و با آزمون و خطا تعیین شده‌اند.

قسمت دوم:

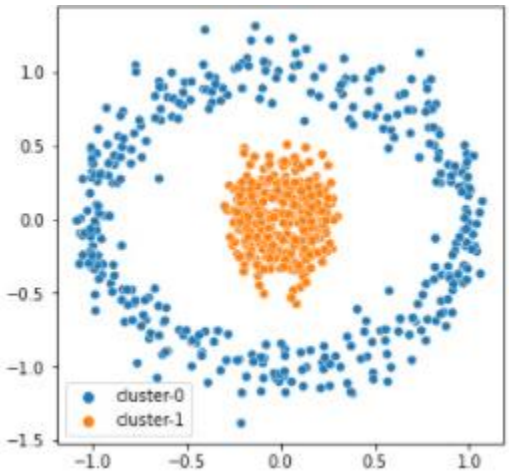
دوباره ابتدا چند کتابخانه لازم اضافه می‌کنیم:

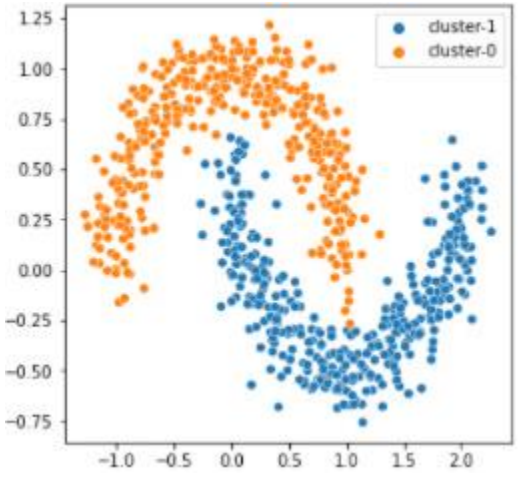
- Seaborn کتابخانه دیگری برای رسم نمودارهاست.
- کتابخانه‌هایی که از sklearn فراخوانده‌شده‌اند به ترتیب برای اجرای الگوریتم DBSCAN، ساخت دیتاست و پیداکردن کمترین فاصله همسایگی برای تعدادی خاص.
- Kneed برای پیداکردن زانو یا knee در نمودار به کار میرود.

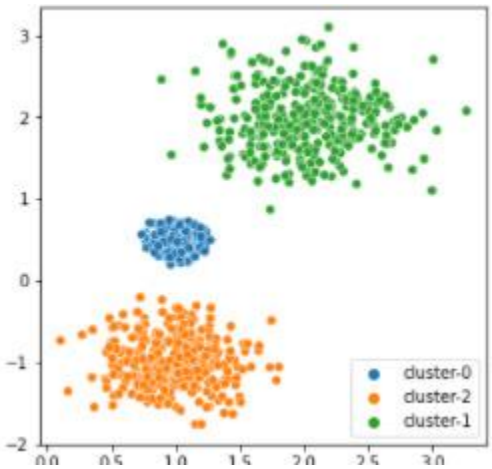
```
import seaborn as sns

from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
from sklearn.datasets import make_blobs, make_circles, make_moons
from kneed import KneeLocator
```

ابتدا مانند قبل نمونه‌های ورودی ساخته می‌شوند. اما نمونه‌ها با چگالی‌های متفاوت و نویز 0.12 و 0.15 درصد ساخته شده‌اند.

```
centers = [[1, 0.5], [2, 2], [1, -1]]
stds = [0.1, 0.4, 0.3]
X, labels_true = make_blobs(n_samples=1000, centers=centers, cluster_std=stds, random_stat

fig = plt.figure(figsize=(5, 5))
sns.scatterplot(X[:,0], X[:,1], hue=["cluster-{}".format(x) for x in labels_true])
plt.savefig("blobs.png", dpi=300)
```



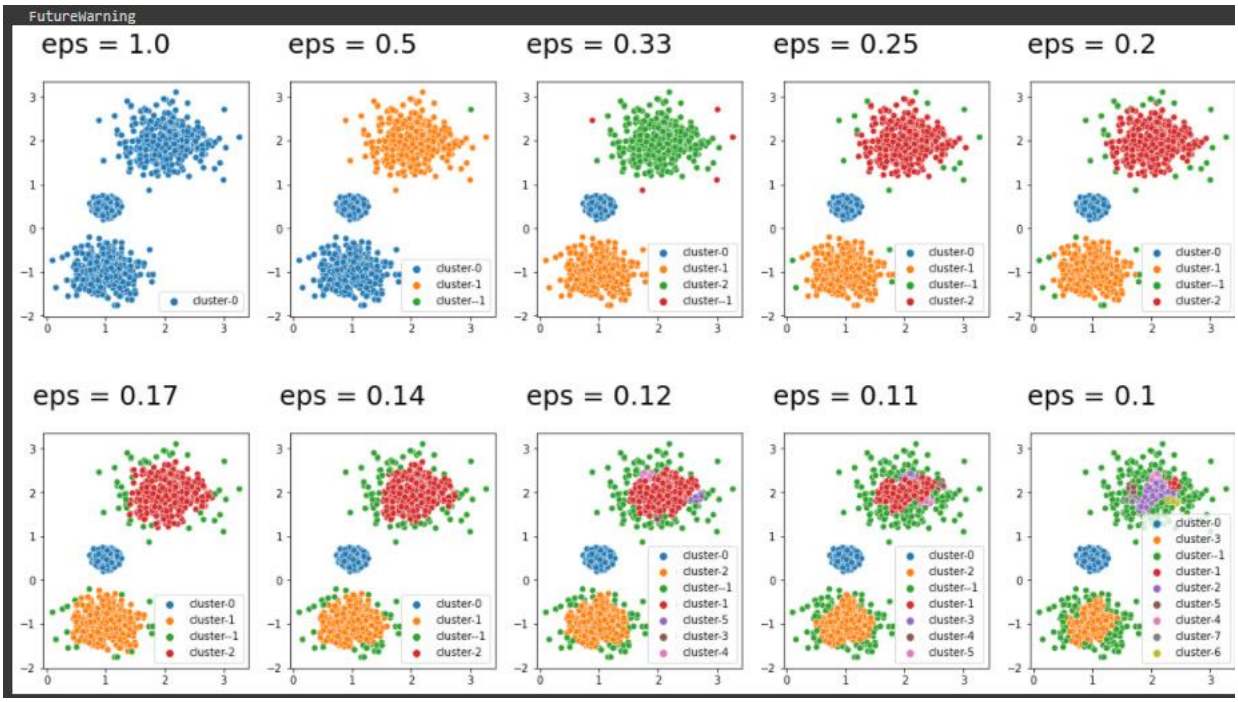




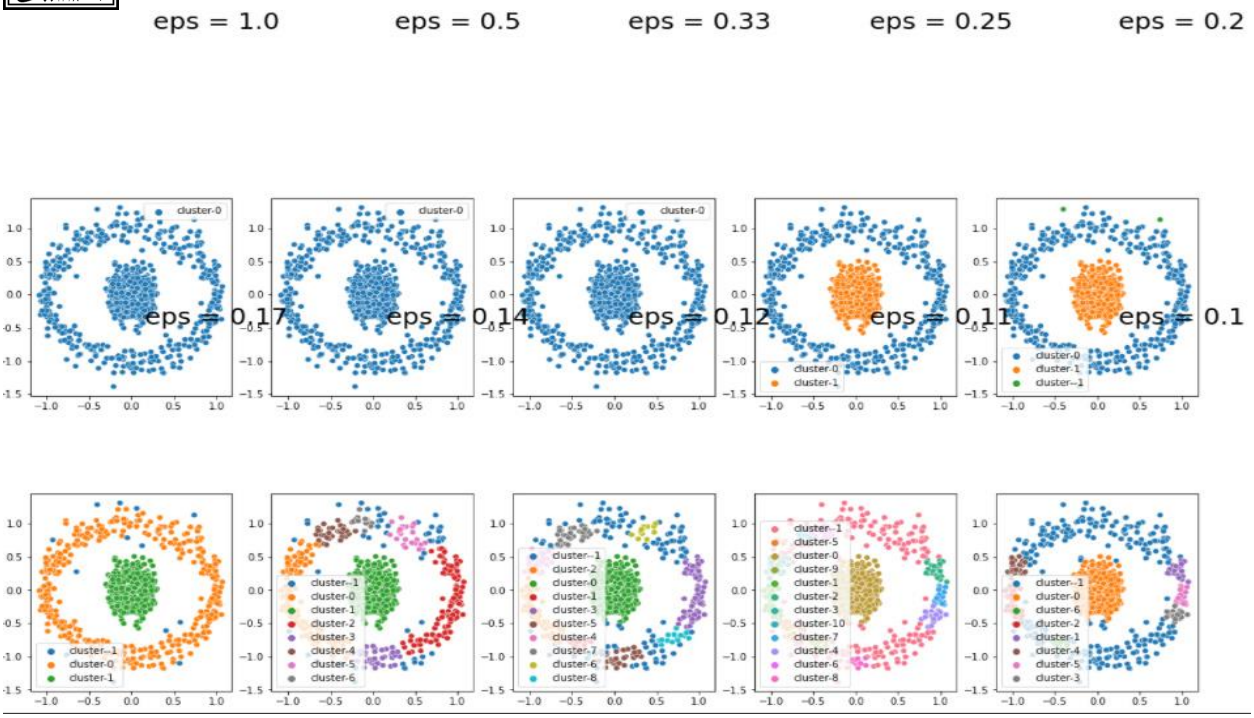
بین پارامترها epsپارامتر مهمی است که تعیین آن در این نوع دیتا ها با چگالی متفاوت به صورت دستی غبرممکن است. درکد اصلی ابتدا یک حدس زده‌ایم و دیدیم که الگوریتم دوخوشه را اشتباها یک خوشه در نظر گرفته‌است. در قسمت زیر تلاش کرده‌ایم که با یک حلقه و تغییر دادن eps مقدار مناسب را پیدا کنیم.

```python
fig = plt.figure(figsize=(20, 10))
fig.subplots_adjust(hspace=.5, wspace=.2)
i = 1

for x in range(10, 0, -1):
    eps = 1/(11-x)
    db = DBSCAN(eps=eps, min_samples=10).fit(X)
    core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
    core_samples_mask[db.core_sample_indices_] = True
    labels = db.labels_

    ax = fig.add_subplot(2, 5, i)
    ax.text(1, 4, "eps = {}".format(round(eps, 2)), fontsize=25, ha="center")
    sns.scatterplot(X[:,0], X[:,1], hue=["cluster-{}".format(x) for x in labels])

    i += 1

plt.savefig("multi_eps.png", dpi=300)
```

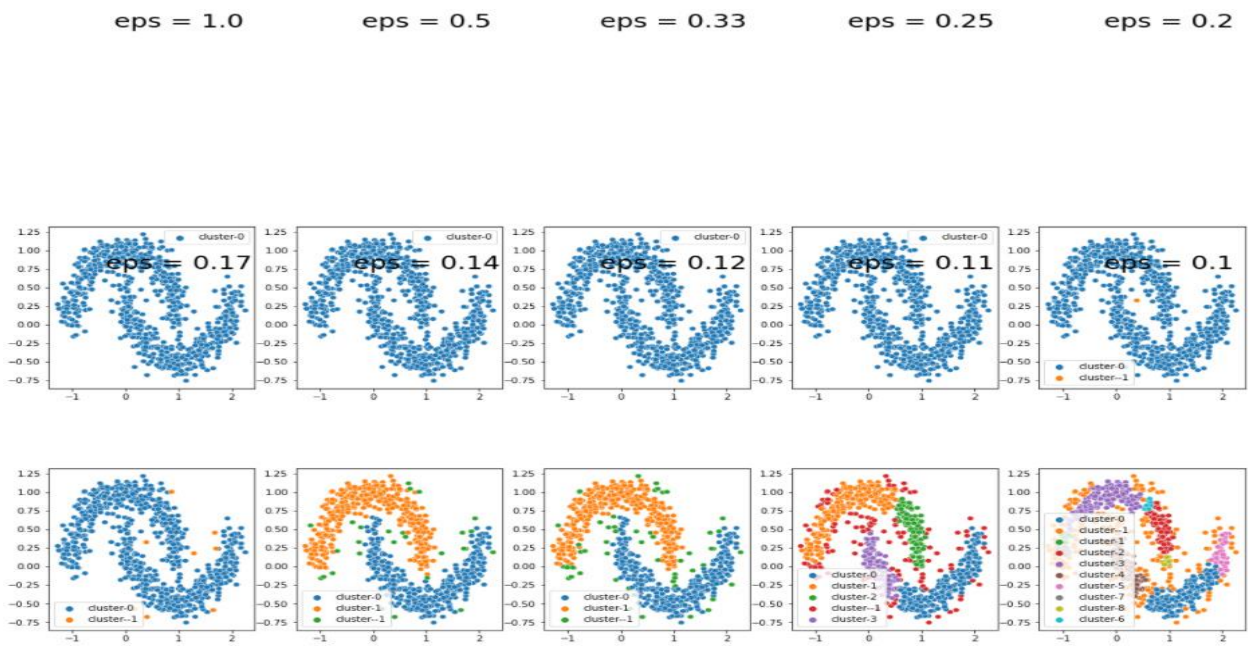
در اینجا میبینیم با حدود eps=0.17 تا eps=0.33 خوشه ها با احتساب کمی خطا به درستی تشخیص داده‌شده‌اند.



برای این نوع دیتا ست 0.1 و 0.11 و 0.17 مناسب است.

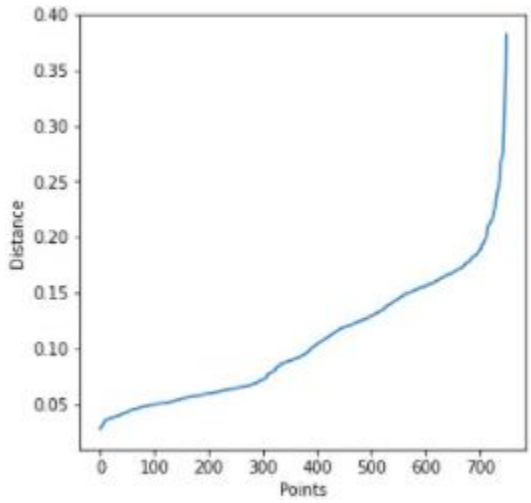


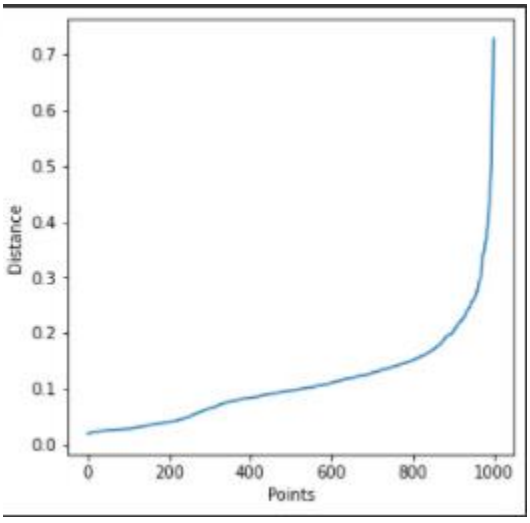
و اینجا 0.14 و 0.17:

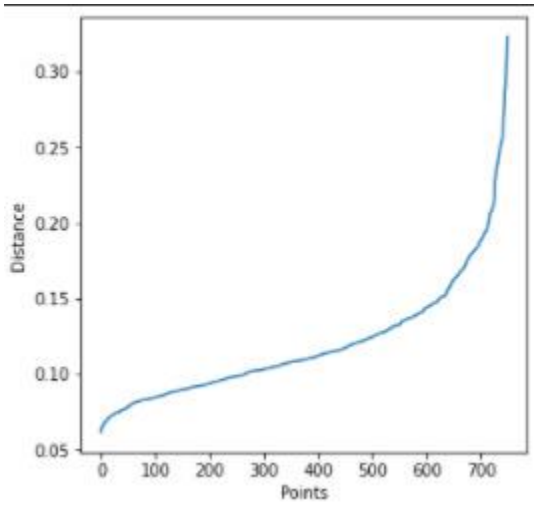


از اونجایی که epsهمون کمترین فاصله‌ای هست که تعداد مثلا 11 تا همسایه برای نمونه‌های اصلی وجود داشته‌باشد، ما هم میایم برای نمونه‌ها فاصله ای رو که براشون این تعداد همسایه وجود دارد میسنجیم و توی یک نمودار میکشیم. در سال 2011 مقاله‌ای منتشر شد مبنی بر اینکه توی مواردی مانند این نمودار بهترین نقطه برای انتخاب eps نطقه ای هست که تا قبل از اون نمودار به طور پیوسته افزایش پیدامی‌کند و بعد از آن ناگهان شیب به بی‌نهایت می‌رود. به این نقطه اصطلاحا knee می‌گویند برای پیدا کردن این نقطه ار کتابخانه kneed استفاده کرده‌ایم که در نمودارهای صفحه بعد مشخص شده‌اند.

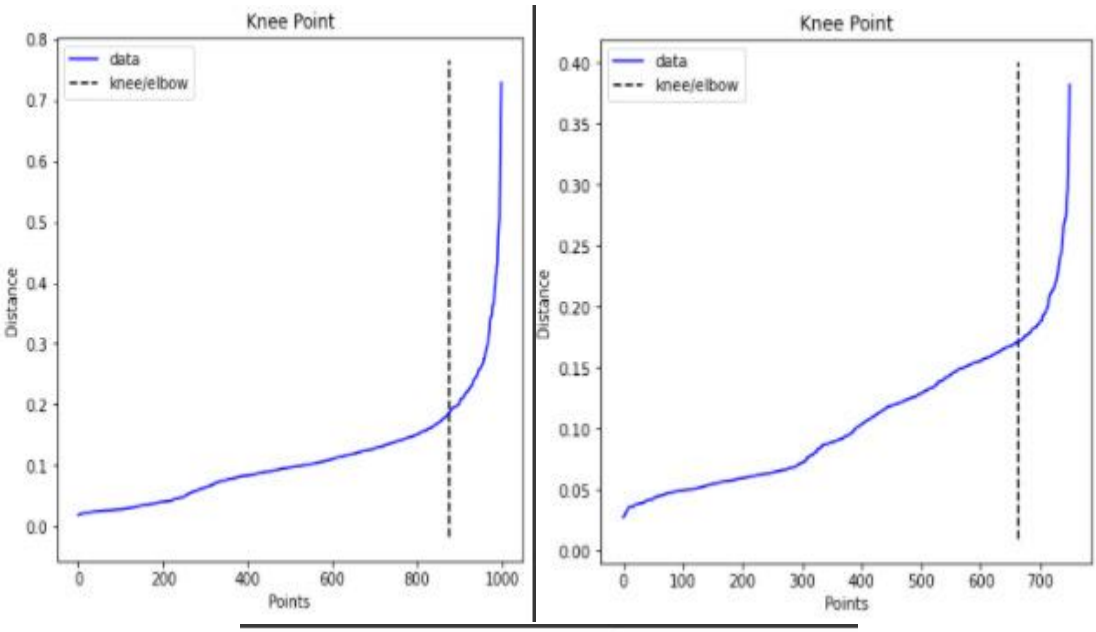
```python
nearest_neighbors = NearestNeighbors(n_neighbors=11)
neighbors = nearest_neighbors.fit(X)
distances, indices = neighbors.kneighbors(X)

distances = np.sort(distances[:,10], axis=0)
i = np.arange(len(distances))
knee = KneeLocator(i, distances, S=1, curve='convex', direction='increasing', interp_method='polynomial')

fig = plt.figure(figsize=(5, 5))
plt.plot(distances)
plt.xlabel("Points")
plt.ylabel("Distance")

plt.savefig("Distance_curve.png", dpi=300)
```

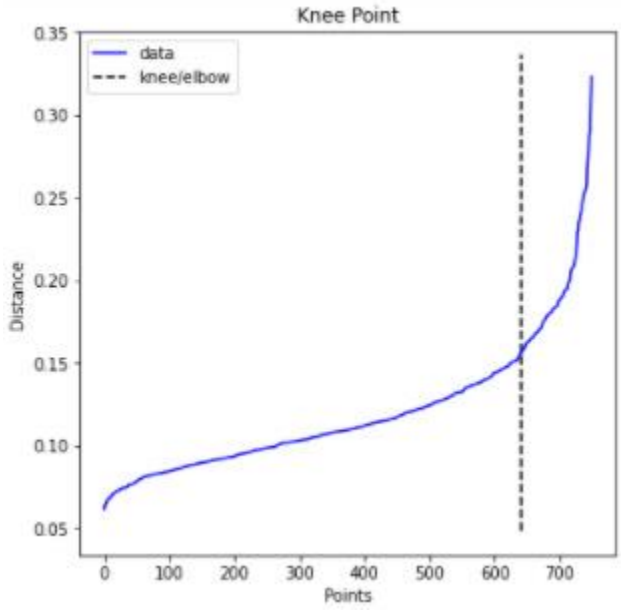






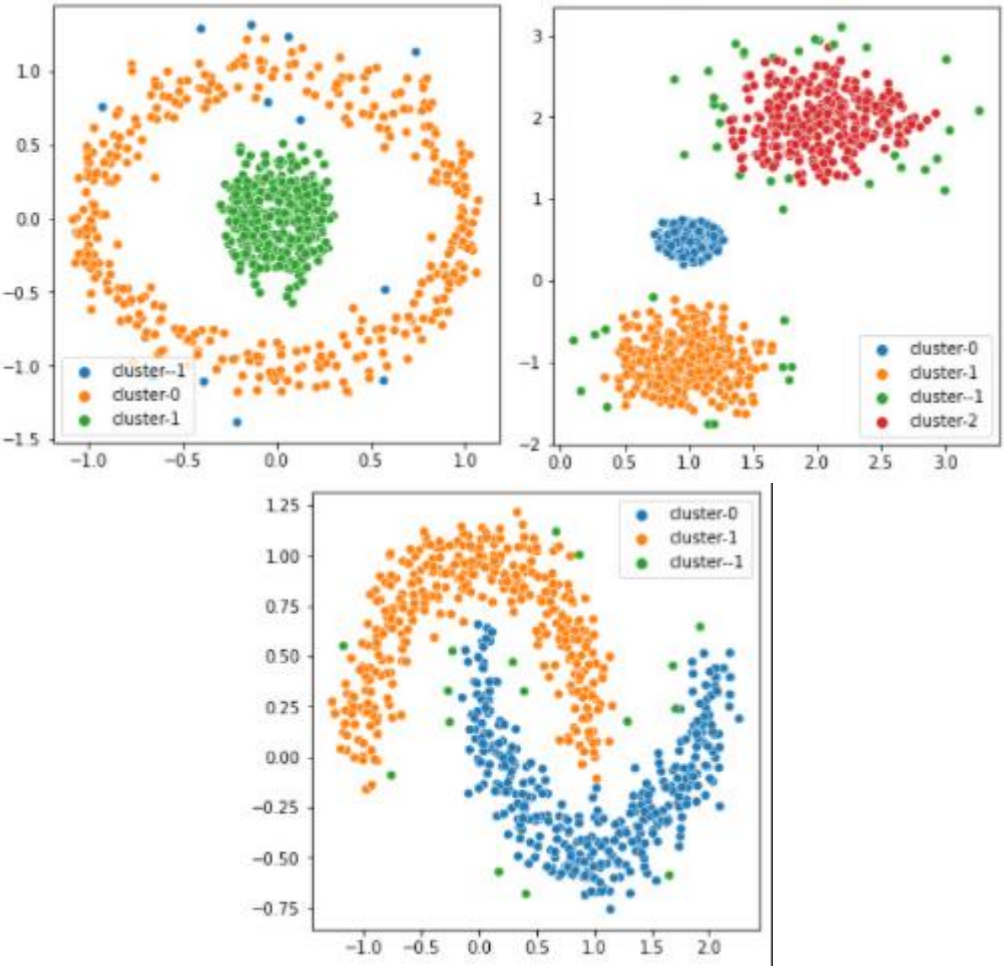
```
fig = plt.figure(figsize=(5, 5))
knee.plot_knee()
plt.xlabel("Points")
plt.ylabel("Distance")
plt.savefig("knee.png", dpi=300)
print(distances[knee.knee])
```





حال که فاصله بهینه را(distance) پیدا کرده‌ایم مقدار آن را به eps می‌دهیم تا با این مقدار خوشه ها را مشخص کند.

```
db = DBSCAN(eps=distances[knee.knee], min_samples=10).fit(X)
labels = db.labels_

fig = plt.figure(figsize=(5, 5))
sns.scatterplot(X[:,0], X[:,1], hue=["cluster-{}".format(x) for x in labels])
plt.savefig("dbscan_with_knee.png", dpi=300)
```



نتیجه:

در اینجا ما می‌بینیم که برآورد معقولی از خوشه‌بندی واقعی داریم. همچنین در اینجا یک سری نقاط به عنوان خوشه -1 در نظر گرفته‌شده اند که نویز محسوب شده‌اند و جزو خوشه‌های اصلی نیستند. می‌توان نهایتا این نقاط را هم با پیداکردن خوشه‌ای که کمترین فاصله را با آنها دارد به باقی خوشه‌ها اختصاص داد. اما در کل به نظر می‌رسد می‌توانیم از ایت تخمین برای پیدا کردن eps استفاده کنیم.