Mahan Veisi – 400243081 – MLHW4

پاسخ سوال 1:

به مشکلی گفته می‌شود که در کار با داده‌های با ابعاد بالا، به ویژه در الگوریتم‌های یادگیری ماشین و خوشه‌بندی، به وجود می‌آید. با افزایش تعداد ابعاد یا ویژگی‌ها، مقدار داده مورد نیاز به طور بسیار زیادی افزایش می‌یابد، که منجر به پراکندگی و چالش‌های مختلف در تجزیه و تحلیل می‌شود. این پدیده می‌تواند تأثیر قابل توجهی بر عملکرد الگوریتم‌های خوشه‌بندی داشته باشد. برخی مشکلات اصلی آن:

Increased Distance between Points:

زمانی که ابعاد زیادی داشته باشیم، در فضای داده ها، نقاط داده‌هایمان به شکلی می‌رسند که در ظاهر از یکدیگر دور هستند. با افزایش تعداد ابعاد، مفهوم فاصله کم‌معنی می‌شود و همه نقاط به نظر می‌آیند که مساوی فاصله دارند. این می‌تواند بر دقت الگوریتم‌های خوشه‌بندی مبتنی بر فاصله تأثیر بگذارد.

Computational Complexity:

داده با ابعاد بالا نیاز به منابع محاسباتی و زمان بیشتری برای پردازش دارد.

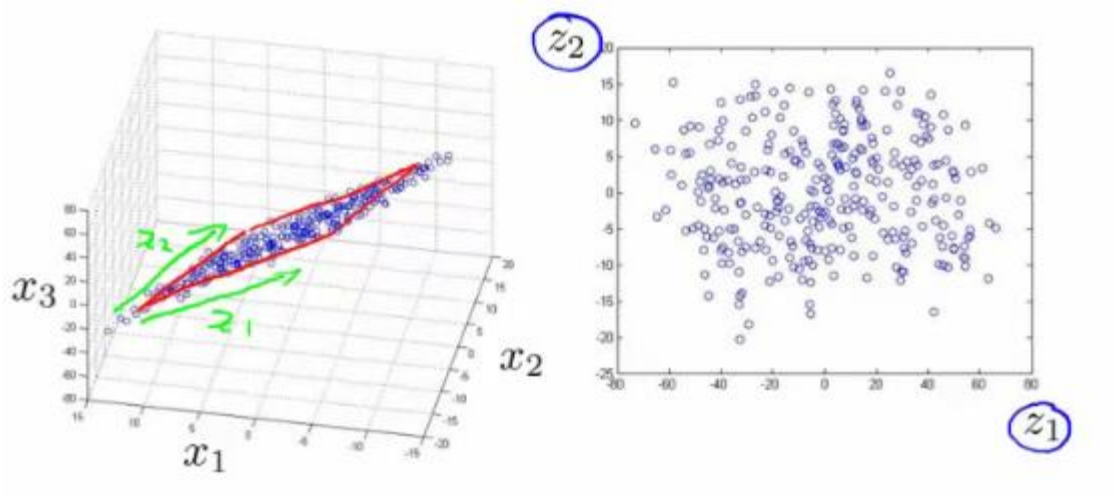
Overfitting:

در فضاهای با ابعاد بالا، ممکن است به راحتی اورفیت رخ دهد، زیر الگوریتم اطلاعات نویز را به جای الگوهای واقعی در داده‌ها ثبت می‌کند و شاید متوجه تفاوتی میان داده های پراهمیت و کم اهمیت نشود که در نهایت منجر به ناکارآیی الگوریتم شود.

حل:

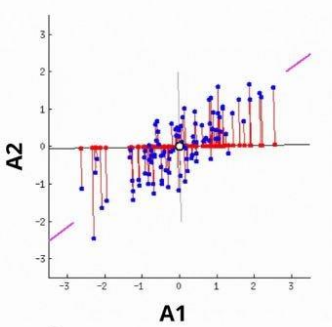
یکی از راه حل ها کاهش بعد (Dimensionality Reduction) است:

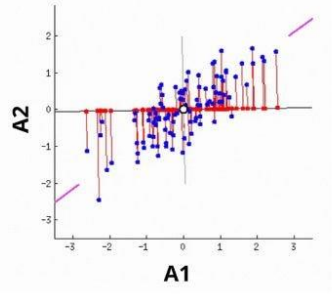
در این حالت روشهایی مانند تجزیه و تحلیل ارکان اصلی (PCA) یا تجزیه و تحلیل مختلط (t-SNE) می‌توانند به پروژه داده با ابعاد بالا به فضای با بعد کمتر کمک کنند و در عین حفظ اطلاعات اساسی، ابعاد را کاهش دهند.

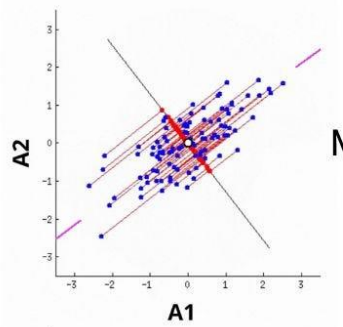


مثالی از PCA:

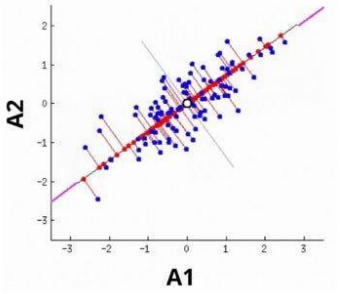
enjoy algorithms







Most unsuitable



Most suitable

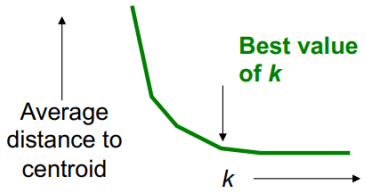
پاسخ سوال 2:

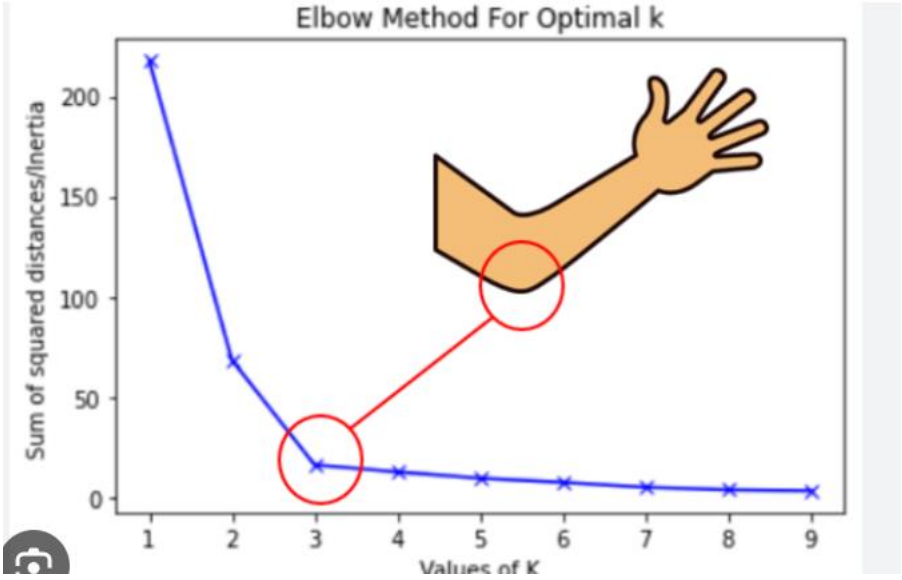
یک روش معمول برای تعیین تعداد بهینه خوشه‌ها دراین الگوریتم، Elbow Method است. این روش به شما کمک می‌کند نقطه‌ای را شناسایی کنید که افزودن خوشه‌های بیشتر (افزیش k) به طور قابل ملاحظه‌ای به بهبود عملکرد مدل کمک نمی‌کند.
در این الگوریتم ابتدا باید الگوریتم را برای k های متفاوت در یک بازه، اجرا کنیم. برای هر مقدار k، جمع مربع فاصله‌ها(اینرشیا) را محاسبه میکنیم. این عبارت برابر با مجموع مربع فاصله‌ها بین هر نقطه داده و مرکز خوشه اختصاص یافته به آن است.
سپس به سراغ رسم نمودار میرویم:

Getting the *k* right

How to select *k*?

- Try different **k**, looking at the change in the average distance to centroid as **k** increases
- Average falls rapidly until right **k**, then changes little

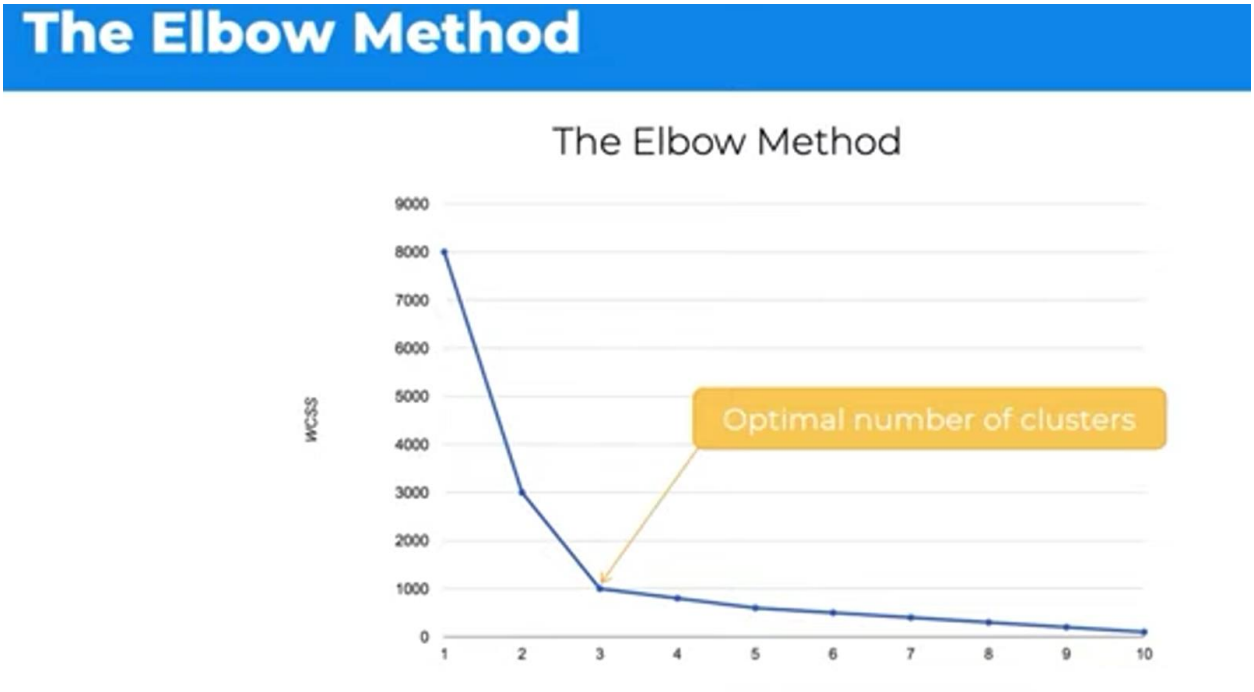




همانطور که تعداد خوشه‌ها (k) افزایش می‌یابد، اینرشیا به طور کلی کاهش می‌یابد زیرا نقاط داده نزدیک به مراکز خوشه‌های خود هستند. با این حال، نرخ کاهش(شیب نمودار) اینرشیا پس از یک نقطه خاص معمولاً کاهش می‌یابد.

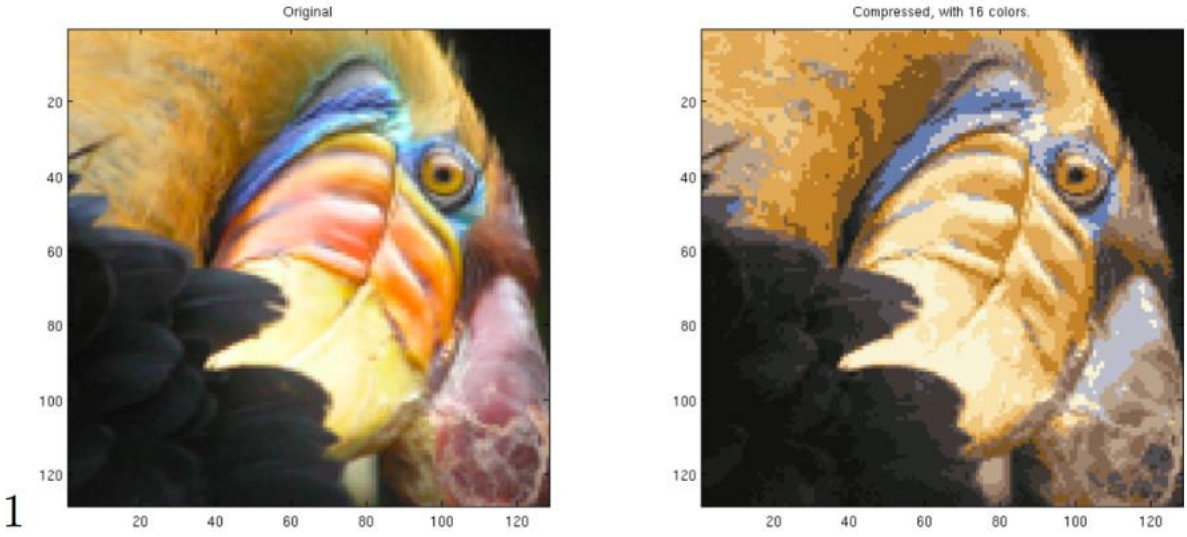
حال باید نقطه شکستگی نمودار یا همان (elbow) را پیدا کنیم. "کمر" منحنی نقطه ای است که در آن نرخ کاهش اینرشیا به طور قابل توجهی تغییر می‌کند و یک منحنی شکل مانند کمر ایجاد می‌کند.

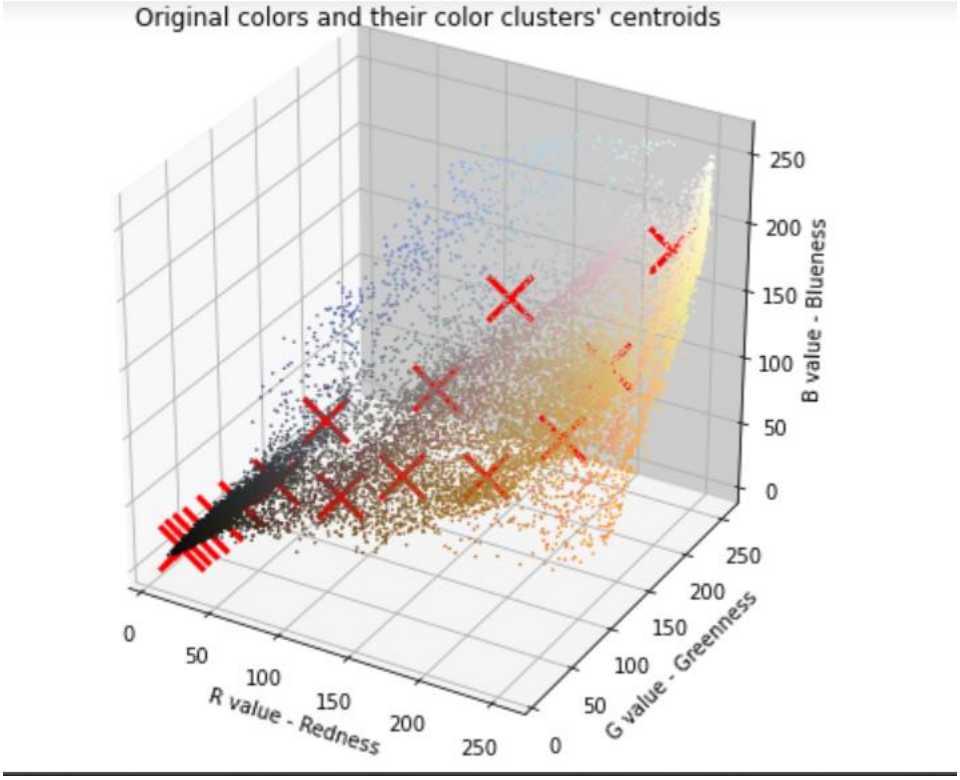
سپس در مرحله آخر، مقدار k متناظر با این نقطه را انتخاب میکنیم.
توجه داریم که این نقطه یک مقدار بهینه برای سایر داده های ممکن نیز به ما میدهد، شاید اگر هرچقدرk را بیشتر کنیم مقدار اینرشیا کمتر شود اما در عمل میتوانیم k را برابر با کل داده ها قرار داده و اینرشیا صفر میشود. اما هدف اصلی این است که داده ها را به بهترین نحوه و جامع ترین آن دسته بندی کنیم پس بهترین k مقداری است اولا تا حد ممکن زیاد بزرگ نباشد از طرفی اینرشیای کمتری داشته باشد.



پاسخ سوال 3:

استفاده از الگوریتم k-means برای فشرده کردن عکس یکی از ایده های جذاب بوده که بنده در ماه های گذشته در یک تمرین آن را انجام داده ام:





در این روش هر پیکسل را به عنوان یک نقطه در فضای دو بعدی (رنگ قرمز، سبز و آبی) مدل میکنیم.
دلیل این کار این است که تابع k-means خود را روی داده های دو بعدی انجام داده ایم در حالی که تصاویر دارای سه بعد میباشند. پس ابتدا تصویر (128, 128, 3) سایزی خود را با این کد زیر به 16384 * 3 تبدیل میکنیم:

```
X_img = np.reshape(original_img, (original_img.shape[0] * original_img.shape[1], 3))
```

سپس برای مثال در تمرین خود، با انتخاب k=16، الگوریتم K-Means را بر روی داده‌های تصویر که دارای بود اجرا میکنیم. K. هر خوشه نمایانگر یک رنگ جدید (مرکز خوشه یا همان centroid) خواهد بود. در نهایت یک آرایه داریم دارای 16 مقدار داریم که هر پیکسل به یکی از مقادیر آن آرایه مپ میشود. تصویر جدید نزدیک‌ترین مرکز خوشه را به عنوان رنگ خود می‌پذیرد.
در مثال خود حجم هرعکس اولیه اینگونه بود:

128×128×24=393,216 bits

* 24 برای این است که هرکدام از کانال ها مقادیر از 0 تا 255 را میپذیرند پس هریک در 8 بیت جا میشوند.

اما پس از اجرای الگوریتم، بخش ضرب در 24 آن کاهش میابد چون در حقیقت فقط 16 مقدار برای نمایش رنگ ها داریم که در 4 بیت جا میشود. از طرفی هر یک از این مقادیر، باید به 24 بیت مپ بشوند تا حاصل رنگ RGB را بدست اوریم، پس در کل حجم مورد نیاز برابر:

bits 65,920 = 4×128×128 + 24×16

است.

قسمتی از کد:

In [13]:

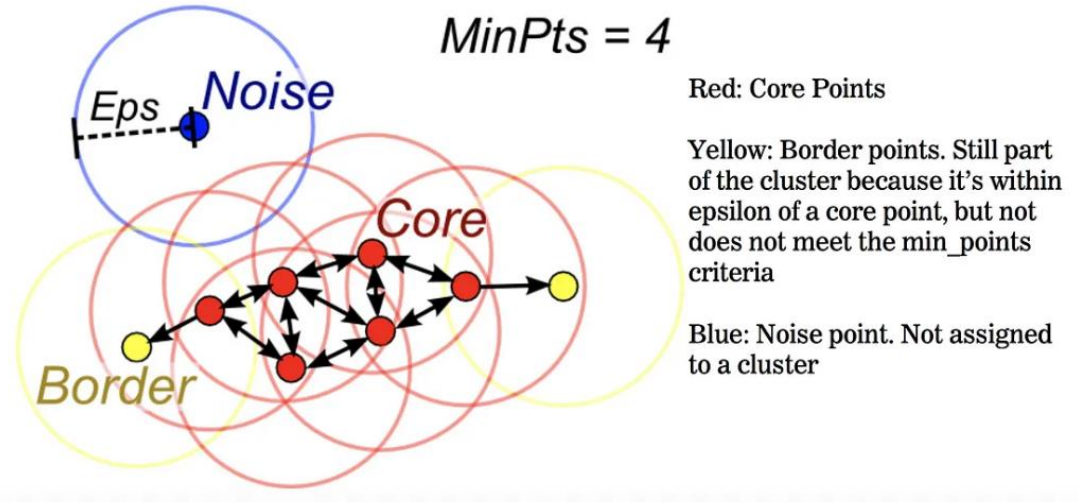
```
def run_kMeans(X, initial_centroids, max_iters=10, plot_progress=False):
    m, n = X.shape
    K = initial_centroids.shape[0]
    centroids = initial_centroids
    previous_centroids = centroids
    idx = np.zeros(m)
    plt.figure(figsize=(8, 6))
    for i in range(max_iters):
        print("K-Means iteration %d/%d" % (i, max_iters-1))
        idx = find_closest_centroids(X, centroids)
        if plot_progress:
            plot_progress_kMeans(X, centroids, previous_centroids, idx, K, i)
            previous_centroids = centroids
        centroids = compute_centroids(X, idx, K)
    plt.show()
    return centroids, idx
```

پاسخ سوال 4:

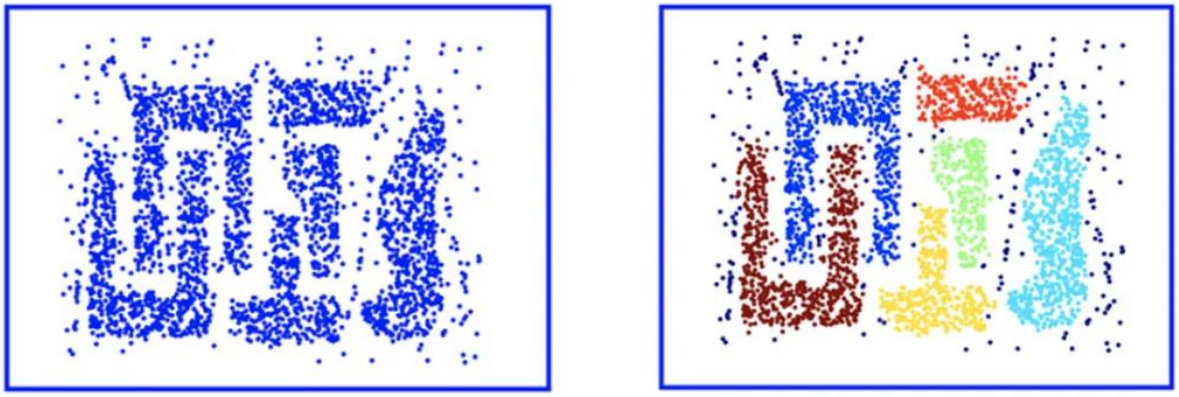
همانند آنچه که در تمرین کدی انجام داده ایم، الگوریتم DBSCAN برای این کار بسیار مناسب است. به طور کلی اگر بخواهیم در مقابل یک الگوریتم unsupervised learning دیگر به آن نگاه کنیم، میتوانیم بگوییم که دلایل انتخاب این الگوریتم عبارت است از:

- زمانی که با داده هایی که دارای شکل خاص و شناخته شده ای نیستند (arbitrary-shaped) یا (Irregularly Shaped Clusters)، DBSCAN قادر است آن ها را راحتتر شناسایی کرده و به خوشه‌های با اشکال، اندازه‌ها و چگالی‌های مختلف در آورد.

- مدیریت نویز: DBSCAN در مدیریت نویز مؤثر است و در پایان الگوریتم خود، این نقاط را به عنوان نقاط نویزی شناسایی می‌کند. زمانی که با مجموعه داده‌هایی سر و کار داریم که همه نقاط حتما مختص به یک خوشه نیستند، این ویژگی پیدا کردن نویز بسیار مفید است.

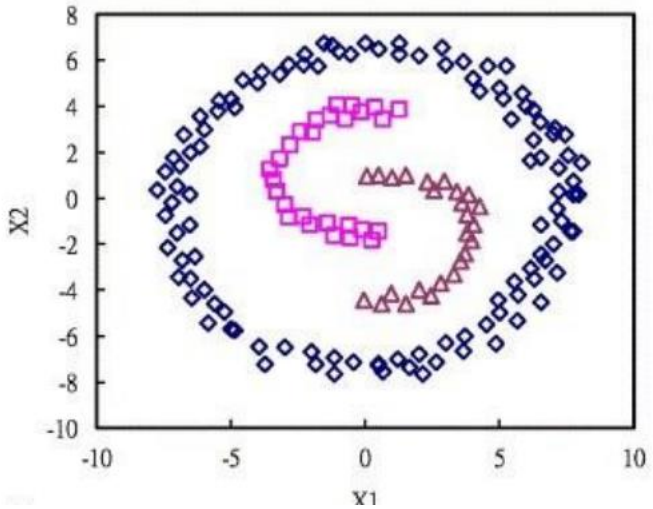


- یکی دیگر از مهمترین خاصیت این الگوریتم برای زمانی است که تعداد خوشه‌ها در داده‌های شما پیش‌از (برخلاف k-means که باید مقدار k را به آن بدهیم) اجرای الگوریتم مشخص نیست. این الگوریتم به طور خودکار تعداد خوشه‌ها را بر اساس چگالی و موقعیت قرارگیری هریک نسبت به یکدیگر پیدا می‌کند.

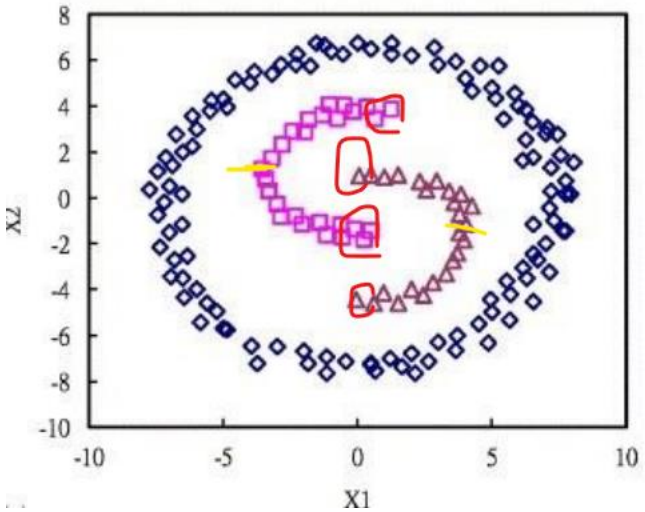


The left image depicts a more traditional clustering method that does not account for multi-dimensionality. Whereas the right image shows how DBSCAN can contort the data into different shapes and dimensions in order to find similar clusters.

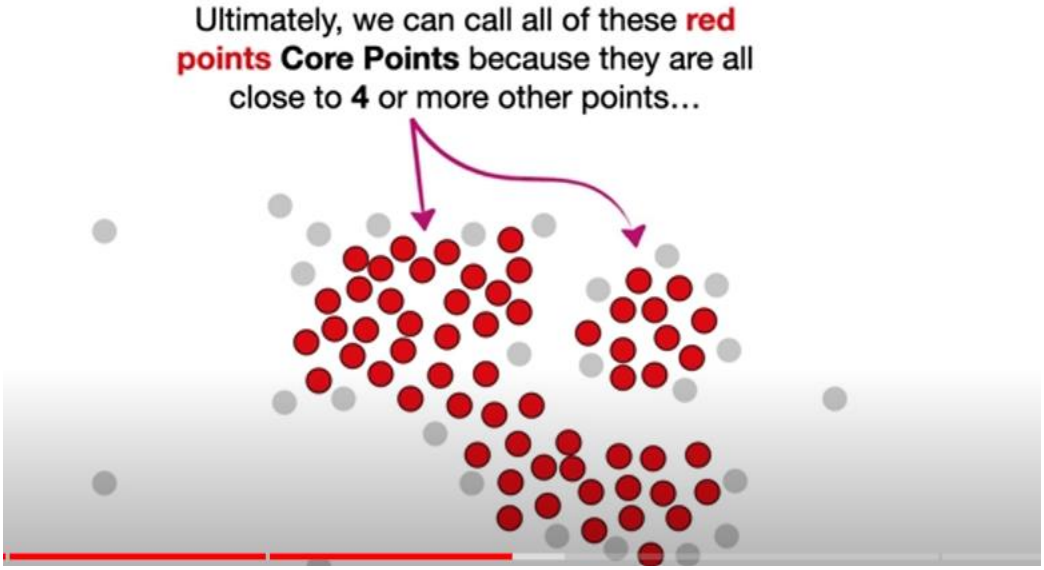
پس با بررسی شکل داده شده:

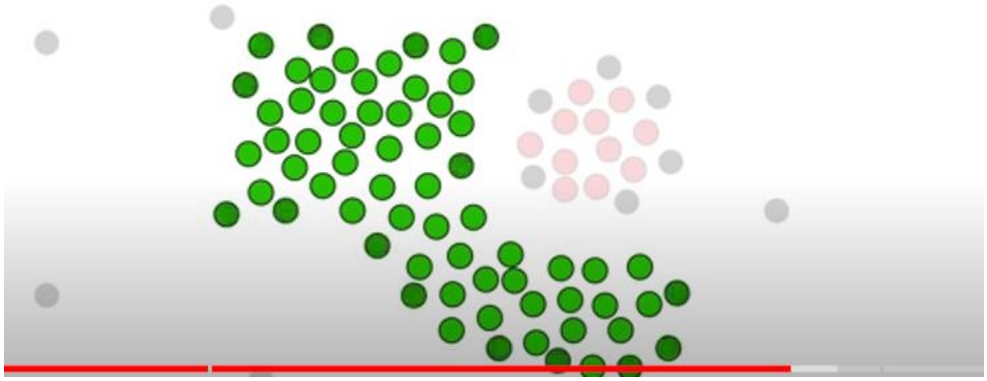


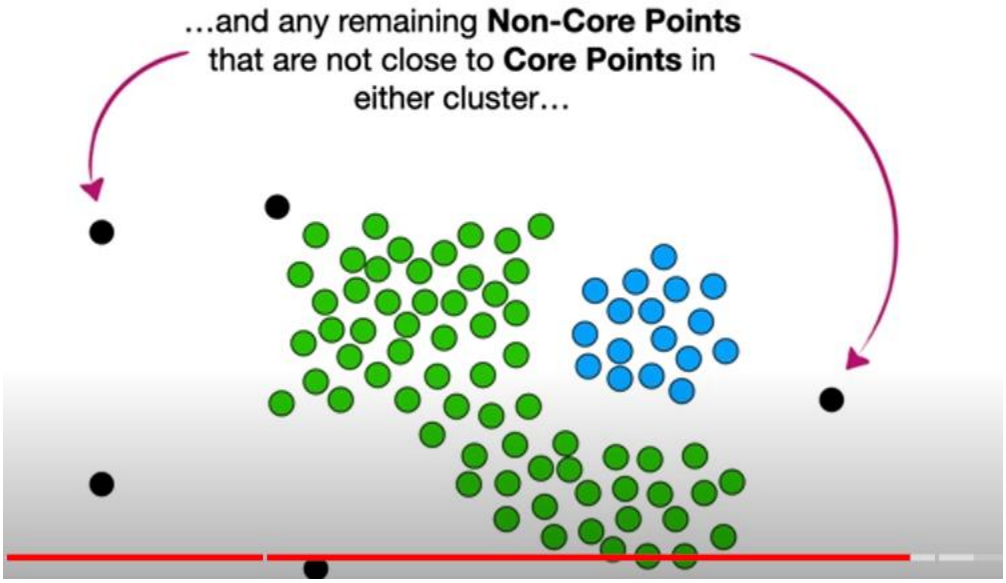
متوجه میشویم که الگوریتم DBSCAN بسیار مناسب است زیرا در سایر الگوریتم ها حتی ممکن است در چنین نقاطی (بخش های قرمز):



دچار اشتباه شویم زیرا اگر بخواهد که centroids هر خوشه را انتخاب کند، انگاه متوجه میشود که این نقاط نزدیک به خوشه های دیگر هم میتوانند باشند. به طورکلی یافتن نقطه مرکز هر خوشه برای این الگوریتم کار اشتباهی است و این الگوریتم که از یک حداکثر شعاع epsilon برای فاصله از هر نقطه و در نهایت خوشه استفاده میکند، به راحتی به پاسخش میرسد.






…and any remaining **Non-Core Points** that are not close to **Core Points** in either cluster…