فرهاد امان ۹۹۳۱۰۰۶

۱-

الف) چون تعداد کل تراکنش‌ها برابر ۵ است تعداد حداقل تراکنش برای پشتیبانی ۳ است. در ابتدا itemset های یک عضوی را به دست آورده و سپس آنهایی که frequent هستند را نگه می‌داریم.

itemset	count
M	3
O	3
N	2
K	5
E	3
Y	3
D	1
A	1
U	1
I	1

پس از نگه داشتن frequent ها

itemset	count
M	3
O	3
K	5
E	3
Y	3

حال با استفاده از روش $F(k-1) * F(1)$ که روش ساده‌تری است itemset های دو عضوی را به دست می‌آوریم

itemset	count
M,0	1
M,K	3
M,E	2
M,Y	2
O,K	3
O,E	2
O,Y	2
K,E	3
K,Y	3
E,Y	2

پس از نگه داشتن frequent ها

itemset	count
M,K	3
O,K	3
K,E	3
K,Y	3

سپس itemset های سه عضوی را محاسبه می‌کنیم.

itemset	count
M,K,O	1
M,K,E	2
M,K,Y	1
O,K,E	2
O,K,Y	2
K,E,Y	2

تمام آنها حذف می‌شوند چون هیچکدام frequent نیستند.

جواب نهایی:

frequent-itemset	count
M,K	3
O,K	3
K,E	3
K,Y	3
M	3
O	3
K	5
E	3
Y	3

ب) تنها از itemset های دو عضوی قانون‌های ممکن را استخراج می‌کنیم.

rule	confidence
K -> M	3/5 = 0.6
K -> O	3/5 = 0.6
E -> K	3/3 = 1
Y -> K	3/3 = 1
M-> K	3/3 = 1
O -> K	3/3 = 1
K -> E	3/5 = 0.6
K -> Y	3/5 = 0.6

پس از نگه داشتن rule هایی با confidence حداقل

rule	confidence
E -> K	3/3 = 1
Y -> K	3/3 = 1
M-> K	3/3 = 1
O -> K	3/3 = 1

ج)

در ابتدا باید itemset ها را برحسب support مرتب کنیم و ایتم های غیر frequent را حذف کنیم.

itemset	count
K	5
E	3
M	5
O	3
Y	3

حالا از هر تراکنش، آیتم‌هایی که frequent نباشد را حذف می‌کنیم.

transaction	items
1	K, E, M, O, Y
2	K, E, O, Y
3	K, E, M
4	K, M, Y
5	K, O

سپس fp-tree را تشکیل می‌دهیم.

suffix	frequent_itemset
K	{K}
E	{E}, {E, K}
M	{M}, {M, K}
O	{O}, {O, K}
Y	{Y}, {Y, K}

حالا itemset های دوتایی را از آن استخراج می‌کنیم.

{E, K}, {M, K}, {O, K}, {Y, K}

د) هر دوی این الگوریتم‌ها برای پیدا کردن الگوهای مکرر در دیتابیس‌های بزرگ استفاده می‌شوند، اما تفاوت‌های اساسی بین این دو وجود دارد که باعث می‌شود FP-growth در بسیاری از موقعیت‌ها ترجیح داده شود.

سرعت و کارایی:

FP-growth فقط دو بار دیتابیس تراکنش‌ها را اسکن می‌کند. یک بار برای به دست آوردن frequency هر آیتم به تنهایی و بار دوم حین ساخت FP-tree برای وارد کردن تراکنش‌ها به درخت. این روش سبب می‌شود که FP-growth سریع‌تر از Apriori عمل کند. در مقابل، Apriori در هر دور از اجرای خود کل دیتابیس را اسکن می‌کند تا itemsets غیر frequent را حذف کند، که این امر منجر به تکرار زیادی در اسکن‌ها و افزایش زمان اجرا می‌شود.

استفاده از حافظه:

ساختار درختی FP-Tree باعث می‌شود که این الگوریتم به طور قابل توجهی کمتر از حافظه استفاده کند. این ساختار درختی اطلاعات مربوط به تراکنش‌ها را به شکل فشرده‌ای ذخیره می‌کند و از تولید و ذخیره همه ترکیب‌های ممکن از آیتم‌ها جلوگیری می‌کند. در نتیجه، FP-growth برای مجموعه‌های داده بزرگتر مقیاس‌پذیرتر است.

دقت و compaction factor:

FP-growth نه تنها سریع‌تر و کم‌حافظه‌تر است، بلکه به دلیل ساختار درختی‌اش، دقت بالایی هم دارد. هر چند، میزان کارایی و سرعت FP-growth به compaction factor بستگی دارد، که همان میزان فشرده‌سازی تراکنش‌هاست. اگر درختی همه شاخه‌ها را داشته باشد، بهینگی کاهش می‌یابد، چون نیاز است که پاسخ‌های زیادی با هم ترکیب شوند تا در نهایت مجموعه‌ای از itemsets مکرر به دست آید.

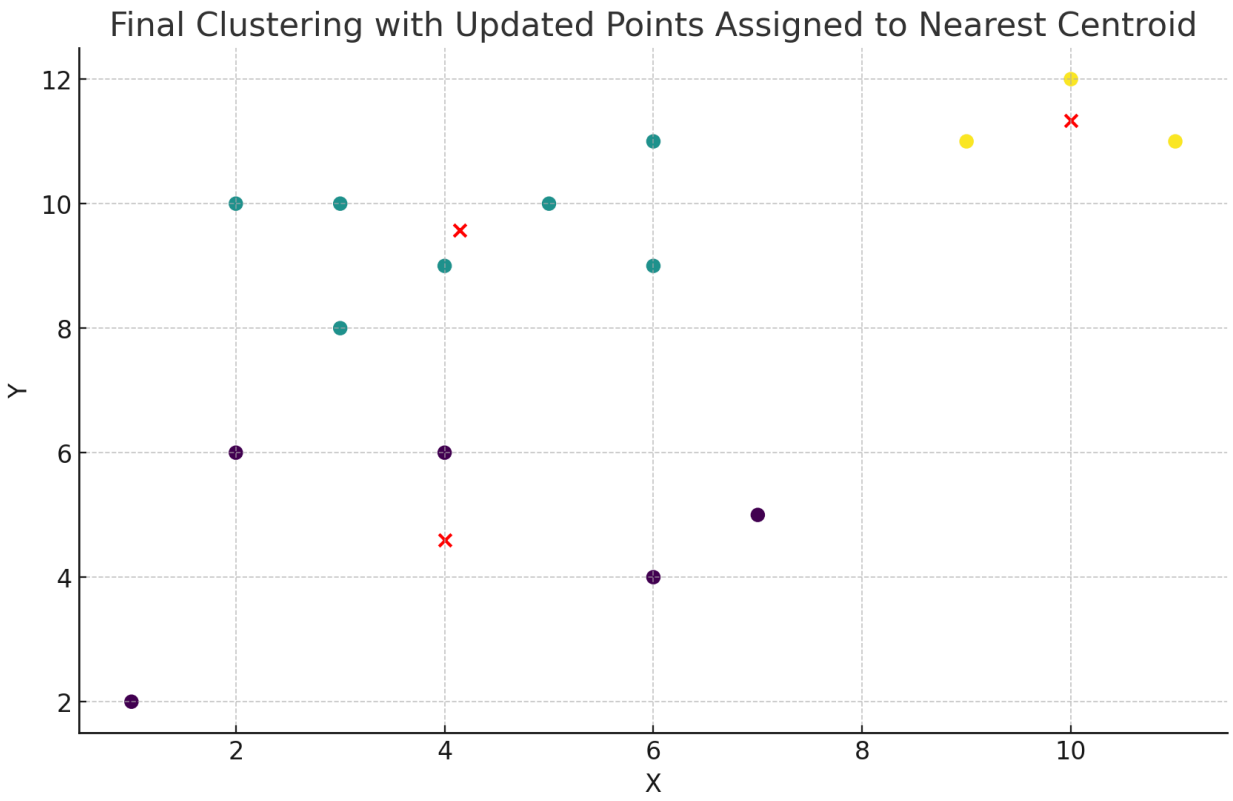
نتیجه‌گیری:

در کل، FP-growth یک پیشرفت قابل توجه در مقایسه با Apriori محسوب می‌شود. این الگوریتم نه تنها سریع‌تر و کارآمدتر است، بلکه از حافظه کمتری نیز استفاده می‌کند و دقت بالاتری دارد، به خصوص در مجموعه‌های داده بزرگ. این ویژگی‌ها FP-growth را به گزینه‌ای مطلوب برای کاربردهای داده کاوی در مقیاس بزرگ تبدیل کرده است.

۲- در ابتدا به ازای هر نقطه، فاصله اقلیدسی تا 3 مرکز را محاسبه می‌کنیم سپس هر نقطه را به کلاستر باکمترین فاصله مرکز نسبت می‌دهیم.
در مرحله بعد مراکز خوشه‌ها را به روزرسانی می‌کنیم به این شکل که مرکز هر خوشه برابر میانگین داده‌های آن خوشه است.
این مراحل را آنقدر ادامه می‌دهیم که مراکز خوشه‌ها تغییر نکنند.

$$C1 = (4, 4.6) \quad C2 = (4.143, 9.57) \quad C3 = (10, 11.334)$$

این مراکز نهایی پس از ۳ iteration است.



در این شکل مراکز خوشه‌ها و همچنین خوشه‌های هر داده قابل مشاهده است.

۳- ماتریس مشابهت در صورت سوال داده شده حالا ماتریس incidence را رسم می‌کنیم.

point	p1	p2	p3	p4
p1	1	1	0	0
p2	1	1	0	0
p3	0	0	1	1
p4	0	0	1	1

ابتدا covariance را محاسبه می‌کنیم.

فرمول covariance برابر است با $\frac{\sum_{I=1}^{N}(P_i - mean_P)(I_i - mean_I)}{N-1}$ بنابراین $cov(I,P)$ به شکل زیر محاسبه می‌شود.

$$\sum_{I=1}^{N}(P_i - mean_P)(I_i - mean_I) = (0.8 - 0.7)(1 - 0.34) + (0.65 - 0.7)(-0.34) + (0.55 - 0.7)(-0.34) + (0.7 - 0.7)(-0.34) + (0.6 - 0.7)(-0.34) + (0.9 - 0.7)(1 - 0.34) = 0.166$$

به مقدار

$$cov(I,P) = \frac{\sum_{I=1}^{N}(P_i - mean_P)(I_i - mean_I)}{N-1} = 0.3/5 = 0.06$$

می‌رسیم.

حالا باید انحراف معیار را محاسبه کنیم.

$$std_P = \sqrt{(0.1^2 + 0.05^2 + 0.15^2 + 0 + 0.1^2 + 0.2^2)/5}$$

$$std_I = \sqrt{(0.66^2 + 0.34^2 + 0.34^2 + 0.34^2 + 0.34^2 + 0.66^2)/5}$$

در نهایت مقادیر محاسبه شده را در فرمول نهایی جایگذاری می‌کنیم و مقدار correlation را محاسبه می‌کنیم.

$$correlation = 0.06(std_p * std_I) = (5 * 0.06)/(0.291547 * 1.15481) = 0.89105$$

۴-

الف) (Incremental K-Means) یک روش خوشه‌بندی است که برای داده‌هایی که به طور پیوسته دریافت می‌شوند، مفید است. این روش از الگوریتم k-means استاندارد الهام گرفته اما با این تفاوت که قادر است به روزرسانی‌های مداوم بر اساس داده‌های جدید را انجام دهد. این ویژگی آن را برای موقعیت‌هایی که داده‌ها به طور مداوم تغییر می‌کنند، مانند جریان‌های داده، مناسب می‌سازد.

توضیحات کلی: در k-means افزایشی، هر بار که دسته جدیدی از داده‌ها وارد می‌شود، مراکز خوشه‌ها (Centroids) به‌روزرسانی می‌شوند تا این داده‌های جدید را منعکس کنند. این به‌روزرسانی می‌تواند بر اساس یک الگوریتم ساده‌تر نسبت به اجرای مجدد کامل k-means باشد. این رویکرد می‌تواند به ویژه در موقعیت‌هایی که حجم داده‌ها بسیار زیاد است یا داده‌ها به طور مداوم تغییر می‌کنند، مفید باشد.

مراحل الگوریتم:

1. **مقداردهی اولیه:** انتخاب تصادفی k نقطه به عنوان مراکز اولیه خوشه‌ها.
2. **تخصیص خوشه‌ها:** برای هر نمونه داده، خوشه‌ای که به نزدیکترین مرکز خوشه دارد، تعیین می‌شود.
3. **به‌روزرسانی مراکز خوشه‌ها:** مرکز هر خوشه بر اساس میانگین موقعیت نمونه‌های درون آن خوشه به‌روزرسانی می‌شود.
4. **تکرار:** تکرار مراحل 2 و 3 تا زمانی که مراکز خوشه‌ها تغییر نکنند یا تغییر آن‌ها کمتر از یک حد آستانه مشخص شود.
5. **به‌روزرسانی افزایشی:** وقتی داده‌های جدید وارد می‌شوند، مراکز خوشه‌ها بر اساس این داده‌های جدید به‌روزرسانی می‌شوند. این ممکن است شامل اضافه کردن خوشه‌های جدید یا حذف خوشه‌های قدیمی باشد.

ویژگی‌ها:

- **مقیاس‌پذیری:** مناسب برای داده‌هایی با حجم بالا.
- **انعطاف‌پذیری:** قابلیت سازگاری با داده‌های در حال تغییر.
- **کارآمدی زمانی:** کاهش زمان اجرا نسبت به اجرای مجدد کامل k-means.

مشکلات:

- **انتخاب k:** تعیین تعداد خوشه‌ها (k) همچنان یک چالش است.

- **حساسیت به مقداردهی اولیه:** نتایج می‌توانند به شدت تحت تأثیر انتخاب اولیه مراکز خوشه قرار گیرند.
- **داده‌های پرت:** مانند k-means استاندارد، k-means افزایشی نیز ممکن است تحت تأثیر داده‌های پرت (Outliers) قرار گیرد.

این الگوریتم در محیط‌هایی که داده‌ها به صورت جریانی وارد می‌شوند بسیار مفید است، اما همیشه نیازمند تنظیمات و بهینه‌سازی بر اساس مشخصات خاص داده‌هاست.

Algorithm 1: incrementalKMN(D,k) Clustering Method

Input: Datasets $D \leftarrow \{d_1, d_2, \ldots\ldots\ldots\ldots d_n\}$ and the value of k.
Output: k number of desired clusters.

```
1  i ← 1
2  C ← {Φ}
3  c_i ← Find_Mean(D)                    // c_i is the mean data object of D.
4  C ← C ∪ {c_i}
5  Assign_Cluster_id(D, C, i)            // Data objects in D are assigned
     Cluster_id starting from 1 to i based on the centroids in C. At
     any point of time C consists of i number of centroids.
6  while i < k do
7      i ← i + 1
8      c_p ← Max_SSE_partial(C)          // c_p is the centroid of the cluster
         with maximum SSE_partial value.
9      c_i ← Max_Dist_from_Centroid(D, c_p)   // c_i is the object at
         maximum distance from c_p.
10     C ← C ∪ {c_i}                     // Update the Centroid List C.
11     Assign_Cluster_id(D, C, i)
12     Compute_Centroid(D, C)            // Centroids are recomputed based on
         the cluster_ids assigned to objects.
13 Compute SSE_total                     // Compute SSE_total of desired clusters
14 End
```

ب)

تجمع داده‌ها در یک نقطه: اگر تقریبا تمام داده‌ها در یک منطقه متمرکز شده باشند و تفاوت قابل توجهی بین آن‌ها وجود نداشته باشد، k-means ممکن است نتواند خوشه‌های معنی‌داری را تشخیص دهد.

انتخاب نامناسب مراکز اولیه: در k-means، انتخاب مراکز اولیه می‌تواند بر نتایج تأثیر بگذارد. اگر این مراکز به شکل نامناسبی انتخاب شوند، ممکن است منجر به خوشه‌بندی نامناسب شود.

ویژگی‌های داده‌ها: ویژگی‌های داده‌ها مانند مقیاس‌بندی، توزیع، و وجود داده‌های پرت می‌تواند بر عملکرد k-means تأثیر بگذارد.

DBSCAN یک روش خوشه‌بندی مبتنی بر چگالی است که می‌تواند برای حل این مشکل مفید باشد. DBSCAN در مقایسه با k-means چندین مزیت دارد:

1. **تشخیص خوشه‌ها با شکل‌های مختلف:** DBSCAN قادر است خوشه‌ها را با هر شکلی تشخیص دهد و به چگالی داده‌ها تکیه دارد.
2. **مقاومت در برابر داده‌های پرت:** DBSCAN به خوبی می‌تواند با داده‌های پرت کنار بیاید و آن‌ها را به عنوان نویز تشخیص دهد.
3. **نیازی به تعیین تعداد خوشه‌ها نیست:** در DBSCAN نیازی به تعیین پیشین تعداد خوشه‌ها نیست، که این می‌تواند در موقعیت‌هایی که تعداد خوشه‌ها نامشخص است، مفید باشد.

برای استفاده از DBSCAN، باید دو پارامتر اصلی تنظیم شوند: حداقل نقاط در یک خوشه و شعاع همسایگی. انتخاب درست این پارامترها برای دستیابی به نتایج خوب حیاتی است.

۵-

1. **مقاومت در برابر نقاط پرت (Outliers) در DBSCAN:** الگوریتم DBSCAN، که بر اساس چگالی داده‌ها عمل می‌کند، برای شناسایی و جداسازی نقاط پرت از داده‌های اصلی بسیار کارآمد است. در این الگوریتم، نقاطی که چگالی کمتری از حد معینی دارند به عنوان نویز یا نقاط پرت شناسایی شده و در نتیجه از مجموعه داده‌های اصلی حذف می‌گردند. این ویژگی اطمینان می‌دهد که خوشه‌بندی تحت تأثیر داده‌های نامرتبط یا نامتعارف قرار نگیرد. درست
2. **شرایط قرار گرفتن داده‌ها در خوشه‌ها:** در DBSCAN، برای اینکه یک نقطه داده در یک خوشه قرار بگیرد، باید نزدیک یک نقطه مرکزی (Core Point) باشد. نقاط مرزی (Border Points)، که در فاصله مشخصی از نقاط مرکزی قرار دارند ولی تعداد همسایگان کافی برای تبدیل شدن به نقطه مرکزی را ندارند، به خوشه‌ای که نزدیک‌ترین نقطه مرکزی را دارند اختصاص داده می‌شوند. این مکانیزم اطمینان حاصل می‌کند که داده‌ها به صورت منطقی و بر اساس فاصله و چگالی در خوشه‌های مربوطه قرار گیرند. درست
3. **فرضیات توزیع داده در DBSCAN:** یکی از مزایای بارز DBSCAN نسبت به سایر الگوریتم‌های خوشه‌بندی، عدم وابستگی آن به فرضیات قبلی در مورد توزیع داده‌ها در فضاست. این الگوریتم بر پایه تشخیص چگالی نقاط عمل می‌کند و به همین دلیل، قادر به شناسایی خوشه‌های با اشکال و اندازه‌های متفاوت است، که این امر اجازه می‌دهد تا با داده‌های متنوع و پیچیده به شیوه‌ای اثربخش مقابله شود. نادرست
4. **عدم نیاز به تعیین تعداد خوشه‌ها در DBSCAN:** در مقابل بسیاری از الگوریتم‌های خوشه‌بندی مانند k-means نیازی به پیش‌تعیین تعداد خوشه‌ها ندارد. این الگوریتم بر اساس چگالی موجود در داده‌ها به طور خودکار خوشه‌ها را شناسایی و تشکیل می‌دهد. این ویژگی اجازه می‌دهد تا الگوریتم به صورت

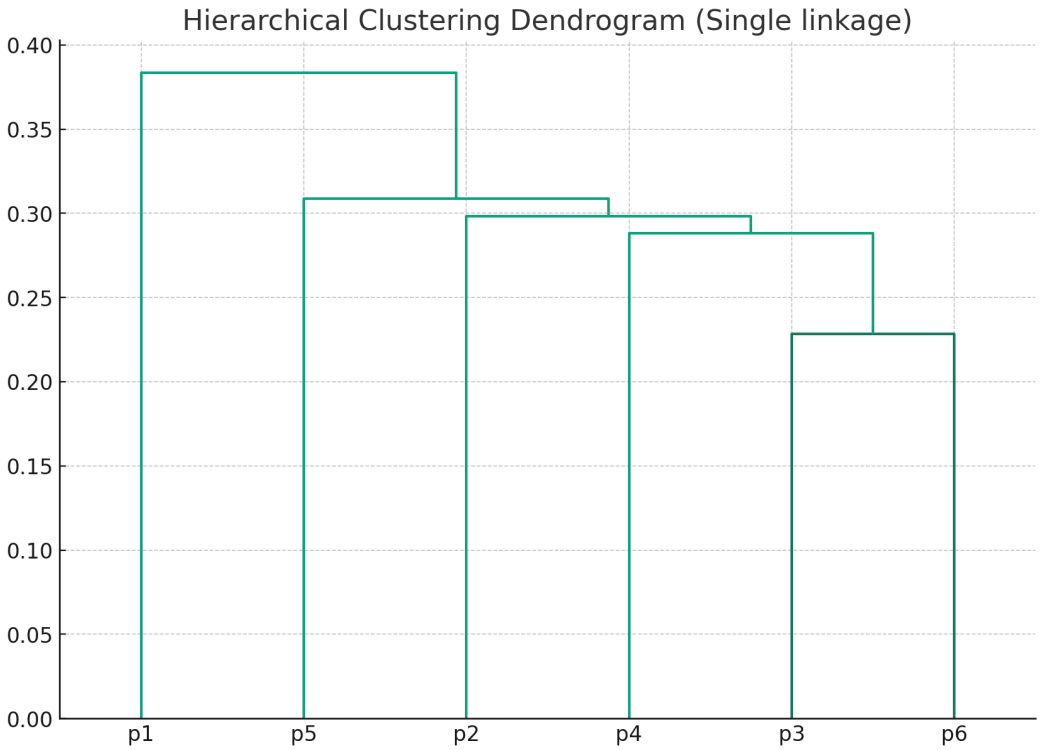
انعطاف‌پذیر با داده‌های مختلف کار کند و خوشه‌هایی را ایجاد نماید که به طور دقیق بازتاب‌دهنده ساختار واقعی داده‌ها باشند. درست

۶-

روش Min

1. **مرحله اول:** هر داده رو یک کلاستر در نظر می‌گیریم. به عنوان مثال، اگر داده‌های ما p1 تا p6 باشن، هر کدام یک کلاستر جداگانه هستند: c1 = p1, c2 = p2, و به همین ترتیب.
2. **مرحله دوم:** دو کلاستر با کمترین فاصله رو پیدا می‌کنیم و با هم ترکیب می‌کنیم. این فاصله می‌تونه بر اساس فاصله‌ی نزدیکترین نقاط (یا هر معیار دیگری) بین دو کلاستر محاسبه بشه. در مثال شما، اول p6 و p3 ترکیب شدن و کلاستر c1 رو ساختن.
3. **مرحله سوم:** به همین ترتیب، پیدا کردن دو کلاستر بعدی که کمترین فاصله رو دارن و ترکیب اون‌ها. مثلاً، p5 و p2 ترکیب می‌شن و کلاستر c2 رو می‌سازن.
4. **مرحله چهارم:** حالا کلاسترهای تشکیل شده در مراحل قبل، یعنی c1 و c2، بر اساس فاصله‌شون (که می‌تونه بر اساس نزدیکترین نقاط بینشون باشه) با هم ترکیب می‌شن و کلاستر بزرگتری، مثلاً b1، رو می‌سازن.
5. **مرحله پنجم و بعدی:** این فرایند تکرار می‌شه. برای مثال، p4 با کلاستر b1 ترکیب می‌شه و در نهایت p1 هم به این کلاستر اضافه می‌شه.

در نهایت، همه داده‌ها در یک کلاستر واحد قرار می‌گیرند. این روش یکی از روش‌های مرسوم برای خوشه‌بندی داده‌ها در داده‌کاوی و تحلیل داده است.



روش Max

1. **مرحله اول:** ابتدا هر داده را به عنوان یک کلاستر مستقل در نظر می‌گیریم. مثلاً p6 و p3 ترکیب شده و کلاستر c1 را تشکیل می‌دهند.
2. **مرحله دوم:** سپس، p5 و p2 ترکیب می‌شوند تا کلاستر c2 را بسازند.
3. **مرحله سوم:** در این مرحله، ما باید فاصله بین کلاسترهای موجود (c1 و c2) و داده‌های باقی‌مانده (p4 و p1) را بررسی کنیم. با توجه به مقادیر فاصله‌ای که دادید، کمترین فاصله بین p4 و c1 است (0.2216). بنابراین، p4 با c1 ترکیب می‌شود تا کلاستر جدیدی، بگوییم b1، تشکیل دهد.
4. **مرحله چهارم:** اکنون باید بین کلاسترهای c2 ،b1، و داده p1 تصمیم گیری کنیم. کمترین فاصله بین p1 و c2 است (0.3421). پس این دو ترکیب شده و کلاستر جدیدی، فرضاً k1، را می‌سازند.

5. **مرحله نهایی:** در نهایت، کلاسترهای باقیمانده، b1 و k1، با یکدیگر ترکیب می‌شوند تا یک کلاستر نهایی بسازند.

این الگوریتم روشی مناسب برای تشکیل خوشه‌های مبتنی بر فاصله‌های بیشینه است و می‌تواند به خوبی نشان‌دهنده‌ی توزیع داده‌ها در فضای چندبعدی باشد.