به نام خدا

دانشگاه صنعتی امیرکبیر

Amirkabir University
of Technology

پروژه - تحلیل کلان داده ها

استادمربوطه: دکتر حقیر چهرقانی

نام: زهرا اخلاقی

شماره دانشجویی: ۴۰۱۱۳۱۰۶۴

تابستان ۱۴۰۲

فهرست:

بخش اول - مقدمه

خوشه بندی داده های نامتعادل یک مسئله چالش برانگیز در یادگیری ماشین است که علت آن عدم تعادل در اندازه خوشه و توزیع چگالی می‌باشد، برای پرداختن به این مشکل، الگوریتم خوشه‌بندی LDPI پیشنهاد شده‌است. در این گزارش ابتدا به معرفی الگوریتم LDPI که برای خوشه بندی داده های نامتعادل طراحی شده پرداخته می شود، سپس این الگوریتم به زبان پایتون پیاده سازی میشود و عملکرد آن روی سه دیتاست ارزیابی میشود.

بخش دوم - الگوریتم LDPI

برای خوشه بندی داده های نامتعادل، الگوریتم خوشه‌بندی LDPI را بر اساس چگالی های محلی ارائه شده است. الگوریتم LDPI شامل سه مرحله است:

1. ابتدا یک نمودار تصمیم گیری سه بعدی بر اساس چگالی، فاصله نقاط و RNN طراحی میشود که می تواند به طور خودکار نقاط نویز و مراکز زیر خوشه اولیه را شناسایی کند. نقاط باقی مانده بر اساس چگالی و فاصله میان خوشه های متعدد تقسیم میشوند
2. در این مرحله،مراکز خوشه که در مرحله اول به اشتباه انتخاب شدند، شناسایی می شوند. مراکز خوشه اشتباه و نقاط آن خوشه ها بر اساس معیار فاصله به خوشه های دیگر اختصاص پیدا میکنند و خوشه های اولیه به روز میشوند.
3. زیر خوشه هایِ به روز شده، در این مرحله با یکدیگر ادغام می شوند و خوشه بندی نهایی ایجاد میشود و نقاط نویز بر اساس فاصله میان این خوشه ها تقسیم میشوند.

الگوریتم پیشنهادی دارای مزیت های زیر میباشد:

- به هیچ پارامتر ورودی نیاز ندارد.
- می تواند به طور خودکار مراکز خوشه و تعداد خوشه ها را تعیین کند.
- برای مجموعه داده ها نامتعادل و مجموعه داده های با اشکال و توزیع های دلخواه مناسب است.

بخش سوم - پیاده سازی

۳-۱ تعیین آستانه فاصله مناسب dc و چگالی

در این الگوریتم ابتدا dc مشخص میشود، برای این کار فاصله هر نقطه تا نزدیک ترین همسایه آن در ابتدا محاسبه میشود و max آنها به عنوان dc در نظر گرفته میشود.

```
def nearest_neighbor_distance(self):
    """
            Calculate distance between all points,
            distance each point and its nearest neighbor,
            maximum distance with nearest neighbor.

            :return: distance distances, Di, ds
    """

    distances = euclidean_distances(self.data)

    nearest_neighbor_distances = np.sort(distances, axis=1)[:, 1]

    return distances, nearest_neighbor_distances, max(nearest_neighbor_distances)
```

تابع بالا فاصله اقلیدسی نقاط با یکدیگر و فاصله هر نقطه تا نزدیک ترین همسایه آن و dc را محاسبه میکند.

برای محاسبه چگالی هر نقطه در این الگوریتم، از نقاط همسایگی آن به فاصله حداکثر dc استفاده می شود که باعث بهبود کارایی و پیچیدگی محاسباتی کمتر الگوریتم میشود:

```
def local_density(self):
    """
    Compute all points' local density.

    :return: local density vector that index is the point index
    """

    rho = [0] * self.n_id

    for i in range(self.n_id):
        for j in range(i + 1, self.n_id):
            if self.distances[i, j] <= self.ds:
                temp = pow(math.e, (-1 * ((self.distances[i, j] / self.ds) ** 2)))
                rho[i] += temp
                rho[j] += temp

    return np.array(rho)
```

۲-۳ تعیین نقاط نویز با استفاده از گراف تصمیم گیری

نقاط نویز تاثیر منفی شدیدی بر نتایج خوشه بندی دارند. بنابراین شناسایی آنها ضروری است. برای شناسایی صحیح نقاط نویز در یک مجموعه داده نامتعادل، از گراف تصمیم گیری استفاده میشود. برای ارزیابی اینکه آیا یک نقطه مرکز خوشه است یا نقطه نویز. ابتدا از درخت k-d برای محاسبه k نزدیکترین همسایه هر نقطه استفاده می کنیم. سپس، با استفاده از آن RNN را محاسبه میکنیم.

نقاط مرکزی دارای RNN بزرگی می باشد در حالی که نقاط نویز مقدار کمی دارند. بنابراین می توانیم RNN را به عنوان بعد سوم (غیر از چگالی و فاصله) برای تشخیص نقاط مرکزی و نویز در نظر بگیریم:

```
def reverse_nearest_neighbors(self, k):

    """
    :param k:
    :return list of RNN for each pont:
    """

    #Build the k-d tree
    tree = KDTree(self.data)

    # Calculate the k nearest neighbors for each point
    _, indices = tree.query(self.data, k=k + 1)  # +1 to include the point itself

    # Initialize an empty dictionary to store the reverse nearest neighbors
    reverse_neighbors = [0] * self.n_id

    # Iterate over each point
    for i in range(self.n_id):
        for j in indices[i]:
            if j != i:
                reverse_neighbors[j] += 1

    return reverse_neighbors
```

```
1 usage
def upward_distance(self):
    """
            Calculate upward distance .

            :return: list
    """

    ud = [0] * self.n_id
    maxDensity = np.max(self.density)

    for begin in range(self.n_id):
        ud[begin] = np.inf
        if self.density[begin] < maxDensity:
            for end in range(self.n_id):
                if self.density[end] > self.density[begin]:
                    ud[begin] = min(ud[begin], self.distances[begin][end])
        else:
            ud[begin] = np.max(self.distances[begin])

    return ud
```

کد زیر برای مشخص کردن نقاط نویز، در الگوریتم اول پیاده سازی شده است.

C یک آرایه به تعداد داده ها می‌باشد که وضعیت هر نقطه را مشخص میکند، مقدار مثبت نشان دهنده مرکز خوشه، ۰ نشان دهنده داده نویز و -۱ بقیه نقاط را مشخص میکند.

```
# determine the noise points
for i in range(self.n_id):
    if self.ud[i] > (mean_ud + std_ud) and self.density[i] < (
            mean_density - std_density) \
            and self.rnn[i] < (mean_rnn - std_rnn):
        C[i] = 0
```

۳-۳ تولید خوشه های اولیه

برای جلوگیری از تعیین تعداد خوشه ها و خوشه بندی خودکار روی مجموعه داده های نامتعادل، پس از حذف نقاط نویز، تعداد زیادی مراکز خوشه به عنوان خوشه بندی اولیه انتخاب میشوند. تکه کد زیر برای تعیین مراکز خوشه اولیه کاربرد دارد:

ICC تعداد خوشه ها را مشخص میکند.

```
# determine the initial sub-cluster centers
for i in range(self.n_id):
    if self.ud[i] > (mean_ud + std_ud) and C[i] != 0:
        ICC += 1
        C[i] = ICC
```

نقاط باقیمانده به ترتیب نزولی چگالی آنها مرتب می شوند. برای اولین نقطه باقی مانده y1، نزدیکترین نقطه مرکزی i با چگالی بیشتر از y1 مشخص می شود. سپس، y1 به زیر خوشه اولیه Ci اختصاص داده می شود. برای بقیه نقاط باقیمانده، نزدیکترین نقطه x با چگالی بیشتر مشخص می شود. هر نقطه به زیر خوشه اولیه ای که x به آن تعلق دارد، اختصاص داده می شود. به این ترتیب، تمام نقاط باقی مانده را می توان به زیر خوشه های اولیه آنها اختصاص داد.

```
# assign the remaining points to the initial sub-clusters
clusters = {i: set() for i, e in enumerate(C) if e > 0}

point_density = {i: self.density[i] for i, e in enumerate(C) if e == -1}
sorted_density = sorted(point_density.items(), key=lambda x: x[1], reverse=True)

for y1, p1 in enumerate(sorted_density):
    nearest_distance = np.inf
    nearest_cluster = None
    i, val = p1
    for j in clusters.keys():
        if self.density[j] > self.density[i] and self.distances[i, j] < nearest_distance:
            nearest_distance = self.distances[i, j]
            nearest_cluster = j
    for y2 in range(y1):
        j, val2 = sorted_density[y2]
        if self.distances[i, j] < nearest_distance:
            nearest_distance = self.distances[i, j]
            for c in clusters.keys():
                if j in clusters[c]:
                    nearest_cluster = c
    clusters[nearest_cluster].add(i)
```

۴-۳ به روز رسانی زیر خوشه‌ها

در این بخش مراکز خوشه که به اشتباه انتخاب شده بودند، شناسایی میشوند. برای انجام این کار فرض کنید که i مین زیر خوشه اولیه با مرکز Ci حاوی Ni نقطه است و در همسایگی آن با چگالی dc، تعداد نقاط Ndc است. اگر Ni<0:5*Ndc، به این معنی است که بیش از نیمی از نقاط این همسایگی توسط سایر زیر خوشه ها جذب می شوند و چنین خوشه ای به درستی انتخاب نشده و باید حذف شود. سپس، با تخصیص نقاط موجود در زیرخوشه‌های نادرست به نزدیک‌ترین زیر خوشه‌های همسایه، زیرخوشه‌ها به روز می‌شوند.

کد زیر برای محاسبه Ni , Ndc برای هر خوشه میباشد:

```
nd = [0] * self.ICC
nc = [0] * self.ICC

centers = np.array(list(self.clusters.keys()))

for i, c in enumerate(centers):
    nc[i] = len(self.clusters[c])
    for j in range(self.n_id):
        if j != c and self.distances[c, j] < self.ds:
            nd[i] += 1
```

در کد زیر مراکز خوشه اشتباه شناسایی می شوند:

```
# delete the false sub-cluster centers from the initial sub-cluster centers
points = set()
for i, c in enumerate(centers):
    if nc[i] < (0.5 * nd[i]):
        self.ICC = self.ICC - 1
        F_ICC += 1
        self.C[c] = -1
        points.add(c)
        val = self.clusters.pop(c)
        points.update(val)
```

در این قسمت از مقاله برای تخصیص نقاط موجود در زیر خوشه نادرست به بقیه خوشه ها تنها گفته شده نزدیک ترین همسایه و گفته نشده منظور از این نزدیکی فاصله با مرکز خوشه ها یا نقاط حاشیه ای است.

در این پیاده سازی من فاصله نقاط باقی مانده با همه نقاط را بررسی کردم و به خوشه نزدیک ترین نقطه، هر نقطه باقی مانده را اختصاص دادم.

```
# assigning the points in the false sub-clusters to their nearest neighboring sub-clusters.
for p in points:
    nearest_distance = np.inf
    nearest_cluster = None

    for j in self.clusters.keys():
        if self.distances[p, j] < nearest_distance:
            nearest_distance = self.distances[p, j]
            nearest_cluster = j

        for n in self.clusters[j]:
            if self.distances[p, n] < nearest_distance:
                nearest_distance = self.distances[p, n]
                nearest_cluster = j
    self.clusters[nearest_cluster].add(p)
```

با این حال، هنگامی که یک خوشه حاوی چندین نقطه اوج چگالی باشد، پس از فرآیند به‌روزرسانی زیرخوشه ممکن است تعداد خوشه‌های فرعی حاصله، تعداد خوشه‌های واقعی نباشد. بنابراین، برای به دست آوردن خوشه های واقعی، یک استراتژی ادغام زیر خوشه ها در این الگوریتم پیشنهاد شده است.

۳-۵ ادغام خوشه ها

در این بخش ابتدا نقاط مرزی هر زیر خوشه مشخص می شود. نقاط مرزی به‌عنوان نقاطی تعریف می‌شوند که چگالی آن‌ها کمتر از میانگین چگالی هر خوشه است. کد سمت چپ زیر برای محاسبه نقاط مرکزی در هر خوشه استفاده میشود. تابع سمت راست برای محاسبه شعاع، طبق رابطه داده شده در مقاله پیاده سازی شده است.

```
def boundary_points(self, cluster):
    """

    :param cluster:
    :return boundary points in a cluster:
    """
    # calculate average density in cluster
    sum_density = 0
    num = 0
    for i in self.clusters[cluster]:
        sum_density += self.density[i]
        num += 1
    avg_density = sum_density / num

    boundary_points = []
    for i in self.clusters[cluster]:
        if self.density[i] < avg_density:
            boundary_points.append(i)

    return boundary_points
```

```
def radius(self):

    if 0 not in self.C:
        r = self.ds
    else:
        r = 0
        for i, c in enumerate(self.C):
            if c != 0 and self.Di[i] > r:
                r = self.Di[i]
    return r
```

برای دو خوشه m و n فاصله آنها به صورت زیر محاسبه میشود:

$$d(m,n) = \min\{d_{ij} | x_i \in m, x_j \in n\}$$

این تابع برای محاسبه فاصله میان دو خوشه کاربرد دارد و فاصله و نقاط نزدیک هر خوشه را در خروجی دارد.

```
def cluster_distance(self, center1, center2):

    """

    :param center1:
    :param center2:
    :return: distance between cluster1 and cluster2
    """

    distance = np.inf
    x1 = None
    x2 = None
    for point1 in self.clusters[center1]:
        for point2 in self.clusters[center2]:
            if distance > self.distances[point1, point2]:
                distance = self.distances[point1, point2]
                x1, x2 = point1, point2
    return distance, x1, x2
```

برای ادغام دو خوشه m و n باید شرایط زیر بررسی شود:

1. اگر فاصله آنها از r بزرگتر باشند، این دو زیر خوشه از یکدیگر دور هستند و نباید ادغام شوند.
2. در غیر اینصورت:
   a. اگر نقاط نزدیک دو خوشه جزو نقاط مرزی آنها نباشند، آن دو خوشه با یکدیگر ادغام میشوند.
   b. در غیر اینصورت:
      i. اگر جمع چگالی نقاط نزدیک دو خوشه از میانگین چگالی مرکز های آنها بیشتر باشد، آن دو زیر خوشه با یکدیگر ادغام میشوند.
      ii. در غیر اینصورت این دو زیر خوشه با یکدیگر ادغام نمیشوند.

```
centers = np.array(list(self.clusters.keys()))
merge_list = [[i] for i in self.clusters.keys()]
for m in range(self.ICC - 1):
    for n in range(m + 1, self.ICC):
        c1 = centers[m]
        c2 = centers[n]
        d, xi, xj = self.cluster_distance(c1, c2)
        if d > self.r:
            continue
        else:
            if xi not in self.boundary_points(c1) and xj not in self.boundary_points(c2):
                merge_list.append([c1, c2])
            else:
                if self.density[xi] + self.density[xj] > ((self.density[c1] + self.density[c2]) / 2):
                    merge_list.append([c1, c2])
                else:
                    continue
```

این فرآیند تا زمانی تکرار می شود که هیچ جفتی از خوشه ها شرایط ادغام را برآورده نکنند. پس از فرآیند ادغام، اگر نقاط نویز باقی نماند، خوشه بندی کامل می شود. در غیر این صورت، هر نقطه نویز به نزدیکترین خوشه اختصاص داده می شود.

```
# Assign the noise points to the clusters
noise_points = [i for i, e in enumerate(self.C) if e == 0]
for n in noise_points:
    min_dis = np.inf
    num_cluster = None
    for i, c in enumerate(final_cluster):
        for p in c:
            if self.distances[n, p] < min_dis:
                min_dis = self.distances[n, p]
                num_cluster = i
    final_cluster[num_cluster].add(n)
```

۶-۳ چارچوب و پیچیدگی زمانی

پیچیدگی زمانی الگوریتم پیاده سازی شده از مرتبه $O(n^2)$ میباشد.

برای هر دیتاست ورودی ابتدا با استفاده از تابع زیر مقدار هر بعد را بین ۰-۱ نرمالایز میکند.

```
def normalize_array(arr):
    min_vals = np.min(arr, axis=0)
    max_vals = np.max(arr, axis=0)
    normalized_arr = (arr - min_vals) / (max_vals - min_vals)
    return normalized_arr
```

در پیاده سازی این الگوریتم، گام های زیر پیاده سازی شده:

1، تابع initial_sub_cluster_construction نقاط نویز را تعیین کنید، مراکز زیر خوشه اولیه و زیر خوشه اولیه مربوطه را ایجاد کنید.

2: تابع sub_cluster_updating مراکز زیر خوشه اولیه و زیرخوشه های مربوطه را بروز میکند.

3: تابع sub_cluster_merging خوشه های فرعی را ادغام کنید تا خوشه های نهایی را با استفاده از الگوریتم 3 بدست آورید.

بخش چهارم - ارزیابی

در این بخش معیار های ارزیابی که برای ارزیابی عملکرد الگوریتم به کار گرفته شده معرفی می شود و سپس نتایج الگوریتم پیاده سازی شده برای سه دیتاست آزمایش میشود

۱-۴ معیارهای ارزیابی

Accuracy:

دقت به این معناست که مدل تا چه اندازه خروجی را درست پیش‌بینی می‌کند. با نگاه کردن به دقت ، بلافاصله می‌توان دریافت که آیا مدل درست آموزش دیده است یا خیر و کارآیی آن به طور کلی چگونه است. اما این معیار اطلاعات جزئی در مورد کارایی مدل ارائه نمی‌دهد.

$$Accuracy = \frac{true\ positives + true\ negatives}{total\ examples}$$

در زمینه خوشه بندی، دقت معمولاً به عنوان معیار ارزیابی اولیه استفاده نمی شود زیرا خوشه بندی جزو یادگیری بدون نظارت است. برخلاف وظایف یادگیری نظارت شده که در آن داده ها را برای مقایسه با آنها برچسب گذاری کرده ایم، الگوریتم های خوشه بندی هدفشان این است که نقاط داده مشابه را بر اساس الگوها یا شباهت های ذاتی آنها بدون دسترسی به برچسب های حقیقی گروه بندی کنند.
دقت یک الگوریتم عددی میان ۰-۱ است و هر چه بزرگتر باشد نشان دهنده این است، الگوریتم عملکرد بهتری دارد.

Recall:

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

زمانی که ارزش false negatives بالا باشد، معیار Recall، معیار مناسبی خواهد بود.
معیار Recall معمولاً در زمینه یادگیری نظارت شده و وظایف طبقه بندی استفاده می شود. در خوشه بندی، که یک کار یادگیری بدون نظارت است، یادآوری معمولاً به عنوان معیار ارزیابی اولیه استفاده نمی شود. این به این دلیل است که الگوریتم های خوشه بندی در طول فرآیند خوشه بندی به برچسب های حقیقت زمینی دسترسی ندارند.
Recall توانایی یک مدل برای شناسایی صحیح همه موارد مثبت از کل نمونه های مثبت موجود در داده ها را اندازه گیری می کند. به عنوان نسبت نمونه های مثبت واقعی به مجموعه نمونه های مثبت واقعی و موارد منفی کاذب تعریف

می شود. هدف الگوریتم‌های خوشه‌بندی کشف گروه‌بندی‌ها یا الگوهای طبیعی درون داده‌ها بدون دانستن انتساب‌های خوشه‌ای واقعی است. این معیار در بازه ۰-۱ قرار دارد و هر چقدر بزرگتر باشد نشان دهنده این است که الگوریتم عملکرد بهتری دارد.

NMI:

NMI مخفف Normalized Mutual Information یک معیار ارزیابی رایج در کارهای خوشه بندی است. اطلاعات متقابل بین خوشه‌های پیش‌بینی‌شده و برچسب‌های واقعی (در صورت وجود) را اندازه‌گیری می‌کند و در عین حال آنتروپی هر دو خوشه را در نظر می‌گیرد.
NMI اندازه‌گیری شباهت یا توافق بین نتایج خوشه‌بندی و برچسب‌های حقیقت پایه را ارائه می‌دهد. اطلاعات متقابل بین دو خوشه، میزان اطلاعاتی را که آنها به اشتراک می گذارند اندازه گیری می کند. اطلاعات متقابل بالاتر نشان دهنده شباهت یا توافق بیشتر بین خوشه بندی ها است. NMI اطلاعات متقابل را با تقسیم آن بر میانگین هندسی آنتروپی دو خوشه نرمال می کند. این نرمال سازی مقداری بین 0 و 1 ارائه می دهد، که در آن 0 نشان دهنده عدم وجود اطلاعات متقابل و 1 نشان دهنده توافق کامل بین خوشه بندی ها است. NMI به ویژه هنگام ارزیابی الگوریتم‌های خوشه‌بندی در مواردی که برچسب‌های حقیقت پایه برای مقایسه در دسترس هستند، مفید است و یک معیار محبوب برای اندازه‌گیری کیفیت نتایج خوشه‌بندی بدون نظارت در برابر برچسب‌های واقعی شناخته شده است. مقادیر بالاتر NMI نشان‌دهنده تطابق خوشه‌بندی بهتر با برچسب‌های واقعی است.

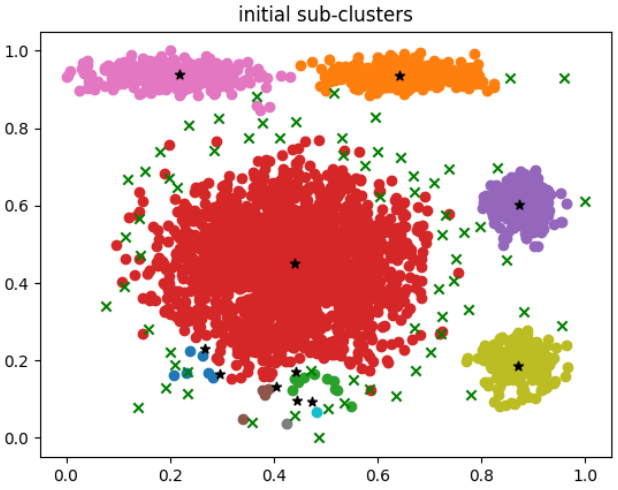
۴-۲ تست و ارزیابی روی دیتاست

برای تست الگوریتم پیاده سازی شده روی دیتاست ها، مقدار k به صورت تصادفی قرار گرفته است، برای همین ممکن است نقاط نویز و خوشه بندی اولیه با نتیجه مقاله متفاوت باشد ولی در نهایت معیار های ارزیابی روی آن نتایج مشابهی با مقاله دارد.
برای پیاده سازی معیار های ارزیابی جایگشت های مختلف از لیبل ها در نظر گرفته شده و نتایج براساس لیبلی با بالاترین دقت گزارش شده

۴-۲-۱ دیتاست ids2

نتیجه اجرای تابع initial_sub_cluster_construction :

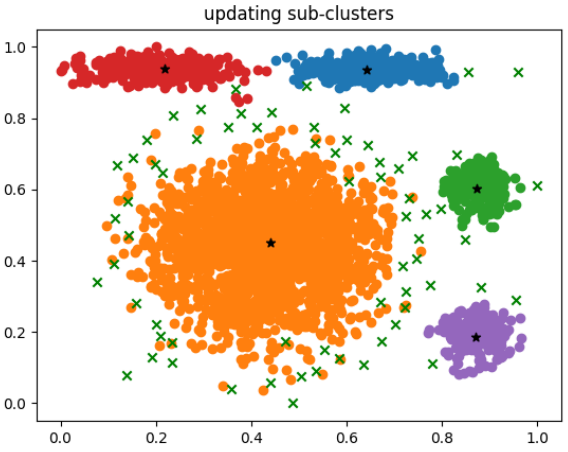
67 noise points and 11 initial sub-cluster centers



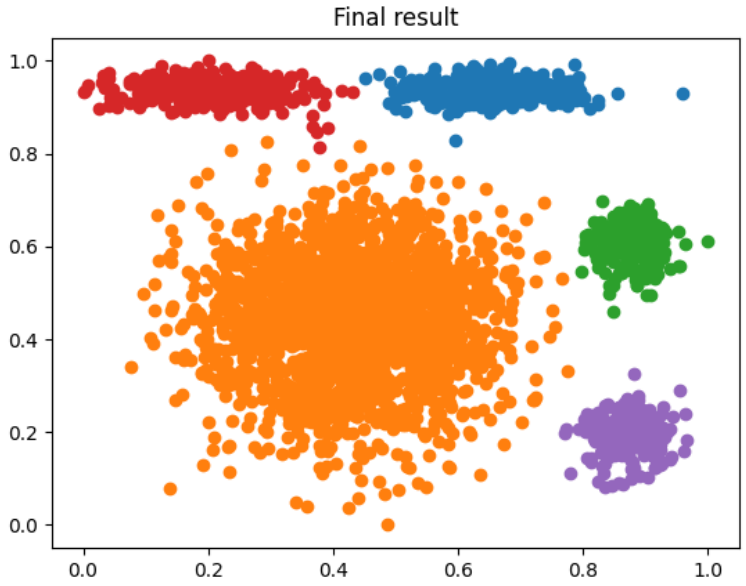
در شکل بالا، ستاره ها مرکز خوشه و نقاط با علامت ضربدر نشان دهنده نقاط نویزی هستند.(برای اجرا k=20 در نظر گرفته شده)

نتیجه اجرای تابع sub_cluster_updating :

5 true sub-cluster centers indicated as blue stars with 6 false sub-cluster



نتیجه اجزا تابع sub_cluster_merging :



معیار های ارزیابی:

```
67 noise points and 11 initial sub-cluster centers
5 true sub-cluster centers indicated as blue stars with 6 false sub-cluster
Accuracy: 0.996875
Recall: 0.9986
Normalized Mutual Information: 0.9820093895654568
Number of Clusters: 5
```

۲-۲-۴ دیتاست gaussian

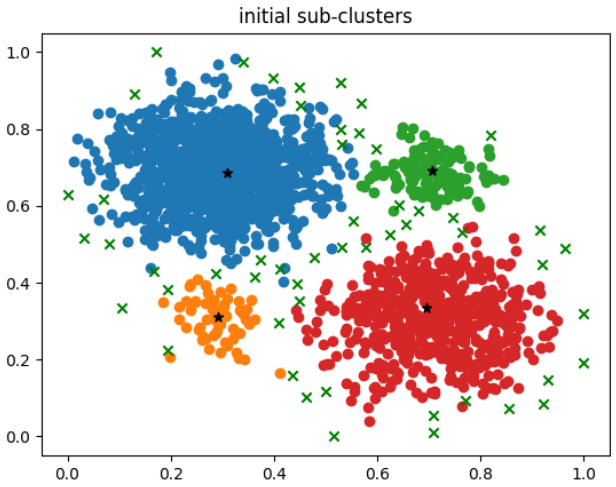
نتیجه اجرای تابع initial_sub_cluster_construction :

44 noise points and 13 initial sub-cluster centers

به علت زیاد بودن تعداد کلاس ها امکان نمایش آن وجود نداشت. (k=400)

در صورتی که k=40 باشد نتایج به صورت زیر است:

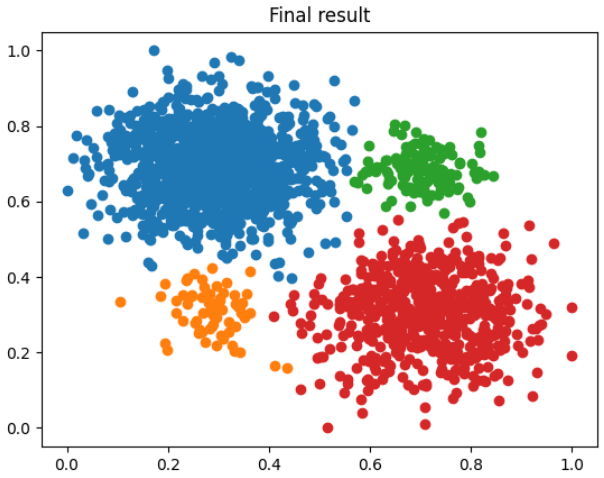
53 noise points and 4 initial sub-cluster centers



نتیجه اجرای تابع (sub_cluster_updating(k=400 :

4 true sub-cluster centers indicated as blue stars with 9 false sub-cluster

نتیجه اجزا تابع sub_cluster_merging :




```
44 noise points and 13 initial sub-cluster centers
4 true sub-cluster centers indicated as blue stars with 9 false sub-cluster
Accuracy: 0.9895
Recall: 0.9880599460840271
Normalized Mutual Information: 0.9383480351896172
Number of Clusters: 4
```

۳-۲-۴ دیتاست Thyroid

برای تست کردن این دیتاست k=10 در نظر گرفته شده و امکان نمایش کلاستر ها به دلیل ابعاد نمیباشد

```
10 noise points and 8 initial sub-cluster centers
2 true sub-cluster centers indicated as blue stars with 6 false sub-cluster
Accuracy: 0.6976744186046512
Recall: 0.3333333333333333
Normalized Mutual Information: 0.21541456567072342
Number of Clusters: 2
```

بخش پنجم - چالش ها

- در پیاده سازی این مقاله برای پارامتر k از یک مقاله دیگر استفاده شده که من آنرا پیاده سازی نکردم و پارامتر k را رندوم انتخاب کردم بنابراین تعداد نقاط نویز و کلاستر های اولیه با مقاله متفاوت است.
- در زمینه خوشه بندی، معیار accuracy , recall معمولاً به عنوان معیار ارزیابی اولیه استفاده نمی شود زیرا خوشه بندی جزو یادگیری بدون نظارت است، برای حل این چالش من به تعداد خوشه های به دست آمده جایگشت ایجاد کردم و نتایج براساس جایگشتی که دارای بیشترین دقت است میباشد.
- در این مقاله برای اختصاص دادن نقاط نویز و یا باقی مانده به خوشه ها تنها گفته شده نزدیکی و مشخص نشده منظور نزدیکی به مرکز خوشه و یا نقاط حاشیه اس در خوشه است و پیاده سازی من بر اساس نزدیکی به نقاط حاشیه است

به نظر من همه نتایج ارائه شده در مقاله درست نیست، با اینکه من از درستی الگوریتم پیاده سازی شده اطمینان دارم ولی برای دیتاست banana , Lithuanian عملکرد درستی ندارد و به دلیلی شرط نقاط داخلی در مرج کردن، دو خوشه نمیتوانند با هم ادغام شوند و نتیجه نهایی شامل سه خوشه است