Задача 3 (Кластерный анализ на плоскости)

Стельмах Татьяна

Январь 2021

Содержание

1 Введение

Задача: На плоскости провести кластерный анализ точек: на малой выборке пользуемся иерархической кластеризацией (для определения местоположения центров), проверяем возможна ли впоследствии `K-means` с помощью вычисления расстояния между центрами, проверяем работу программы на трёх сгенерированных тестах (допущение: в кластерах примерно одинаковое количество точек), визуализируем их, объясняем результат программы и делаем краткие выводы.

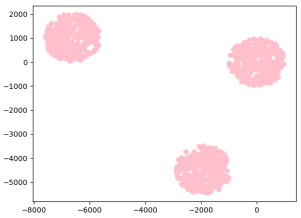
2 Описание метода

Иерархическая кластеризация и `K-means` написаны самостоятельно.
На малой выборке реализуем иерархичсекую кластеризацию несколько раз (так как мы смотрим на малой выборке), находим массив получившихся центров (во всех этих испытаниях). Перед этим обязательно на каждом шаге проверяем расстояние между центрами, если же в 60% от всех случаях центры довольно близки, либо в 80% случаях в одном из кластеров намного больше точек, то делаем вывод о невозможности `K-means` (кластеры либо накладываются, либо один из них окружает другой). Затем берём 3 рандомных центра (разных кластеров!), с помощью них проводим `K-means` для центров

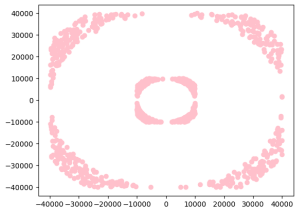
до тех пор, пока центры центров не будут находится на нужном расстоянии. Затем с помощью найденных центров проводим `K-means` уже для всего набора точек.

3 Описание данных

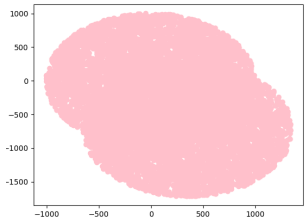
Данные генерируются самостоятельно: в 1 тесте генерируем 3 кластера на заданном расстоянии, во 2 тесте генерируем 2 кластера: один кластер окружает другой, в 3 тесте также генерируем 2 кластера: они накладываются друг на друга. Ниже можно увидеть их визуализацию (визуалзировали лишь случайную выборку из полученного csv-файла):



Test 1



Test 2



Test 3

В итоге на вход получаем csv-файлы, которые считываем в теле программы.

4 Описание постановки эксперимента

Приведём код иерархической кластеризации:

```
def distance(x, y, x1, y1):
    dist = sqrt((y - y1) ** 2 + (x - x1) ** 2)
    return dist

def get(v, st):
    while st[v] != -1:
        v = st[v]
    return v


def merge(v1, v2, st):
    v1 = get(v1, st)
    v2 = get(v2, st)
    if v1 != v2:
        st[v1] = v2
        return True
    return False


def simplify(v, st):
    vertices = []
    while st[v] != -1 and st[v] != v:
        vertices.append(v)
        v = st[v]
    for vertex in vertices:
        st[vertex] = v
    st[v] = v
```

```
def hierarchy_cluster(data, n_clusters):
    distances = []
    for j in range(len(data)):
        for k in range(j + 1, len(data)):
            distances.append([distance(data[j][0], data[j][1], data[k][0], data[k][1]), j, k])
    distances.sort()

    n = len(data)
    clusters = [-1 for i in range(n)]
    hist = []
    for element in distances:
        root_1 = get(element[1], clusters)
        root_2 = get(element[2], clusters)
        if merge(root_1, root_2, clusters):
            hist.append((root_1, root_2))

    for i in range(n_clusters - 1):
        v1, v2 = hist[-1 - i]
        clusters[v1] = -1

    for i in range(n):
        simplify(i, clusters)
    indixes = dict()
    i = 0
    for el in set(clusters):
        indixes[el] = i
        i += 1
    for i in range(len(clusters)):
        clusters[i] = indixes[clusters[i]]

    return clusters
```

Приведём код K-means:

```
def kmeans_cluster(data, centers, epochs=15):
    clusters = np.zeros(len(data))
    cnt_of_element = np.zeros(n_clusters)
    sum_of_element = [[0, 0] for i in range(n_clusters)]
    for m in range(epochs):
        for i in range(len(data)):
            for j in range(len(centers)):
                if j == 0:
                    min_dist = distance(data[i][0], data[i][1], centers[j][0], centers[j][1])
                    index_min_dist = 0

                if distance(data[i][0], data[i][1], centers[j][0], centers[j][1]) < min_dist:
                    index_min_dist = j
                    min_dist = distance(data[i][0], data[i][1], centers[j][0], centers[j][1])

            clusters[i] = index_min_dist
            sum_of_element[index_min_dist][0] += data[i][0]
            sum_of_element[index_min_dist][1] += data[i][1]
            cnt_of_element[index_min_dist] += 1
            if i % 100000 == 0:
                print("K-Means epoch", m, ":", i+1)
        for t in range(n_clusters):
            centers[t][0] = sum_of_element[t][0] / cnt_of_element[t]
            centers[t][1] = sum_of_element[t][1] / cnt_of_element[t]

    return clusters, centers
```

Основную часть программы можно увидеть непосредственно в самом коде, идея описана выше.

5 Результаты

Итак, давайте разберём каждый пример по порядку:

1. Тест Первый: Программа справилась с ним без всяких проблем, в конце мы получили csv-файл, в котором хранится информация о принадлежности точки к определённому кластеру. Результат можно увидеть ниже (для удобства мы визуализировали случайную выборку из этого файла):

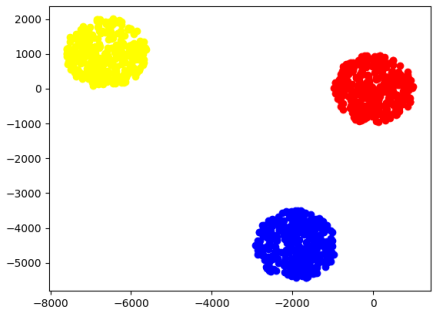


Рисунок 1. Результат 1 теста

2. Тест второй: Программа также справилась и с этим тестом, несмотря на то, что один кластер окружал другой. Она выявила тот факт, что K-means применять бесполезно, ниже мы визуализируем промежуточный результат: то есть то, как программа определяла центры с помощью Иерархической кластеризации. Ниже можно увидеть 2 случая: центры находятся очень близко; центры находятся чуть дальше. Это нормально, поскольку иерархическую кластеризацию мы проводим на малой выборке, поэтому и в теле программы хотя бы в 60% случаях центры должны располагаться довольно близко, чтобы сделать вывод о бессмысленности дальнейших действий.

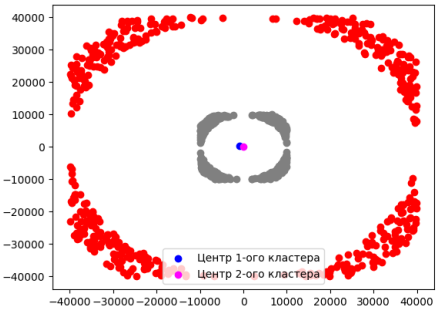


Рисунок 2. Случай 1

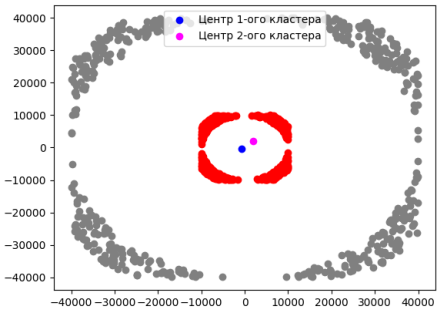


Рисунок 3. Случай 2

3. Тест третий: В этом случае уже возникли некоторые трудности, однако после некоторых соображений мы, кажется, смогли их преодолеть. Так как иерархическую кластеризацию мы рассматриваем на малой выборке, а кластеры накладываются друг друга, то есть находятся довольно близко, вероятнее всего во второй получившийся кластер попадёт крайне маленькое количество точек, и мы сможем распознать этот момент (в связи с малой выборкой будут некоторые точки, которые довольно сильно отдалены от остальных), тогда, как мы и предполагали, программа на промежуточном шаге будет иметь следующее расположение центров кластеров и самих кластеров (Рисунки 4, 5 и 6), что и будет свидетельствовать о невозможности K-means. На рисунках 4 и 5 есть кластеры, состоящие из одной точки, на рисунке 6 - из трёх, что в разы меньше кол-ва точек другого кластера.

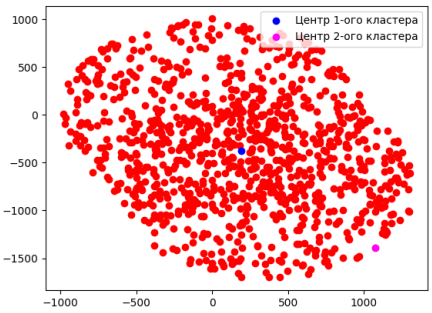


Рисунок 4. Случай 1

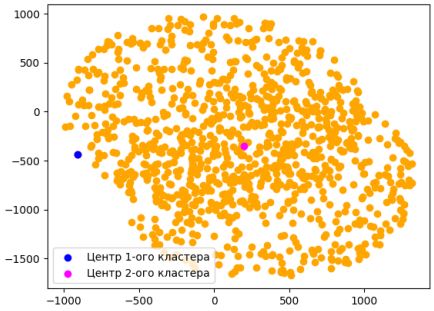


Рисунок 5. Случай 2

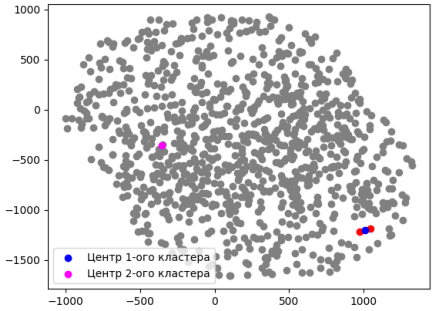


Рисунок 6. Случай 3

6 Краткие выводы

Мы произвели иерархическую кластеризацию на малой выборке, затем смогли различить ситуации, где K-means не применим (кластеры накладываются/один

кластер окружает другой), при этом нам понадобилось около 100 запусков иерархической кластеризации для того, чтобы быть уверенными в том, что один кластер действительно окружает другой (для теста 2), то есть на какой-то случайной малой выборке мы могли получить совершенно противоположные результаты.