# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №7**

**по дисциплине «Статистические методы обработки экспериментальных данных»**

**Тема: Кластерный анализ. Метод поиска сгущений.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6381 |  | Фиалковский М.С. |
| Преподаватель |  | Середа В.И. |

Санкт-Петербург 2020

# Цель работы.

Освоение основных понятий и некоторых методов кластерного анализа.

# Задание.

Дано конечное множество из объектов, представленных двумя признаками (в качестве этого множества принимаем исходную двумерную выборку, сформированную ранее в лабораторной работе №4).

1. Выполнить разбиение исходного множества объектов на конечное число подмножеств (кластеров) с использованием метода поиска сгущений. Добиться устойчивого разбиения.
2. На каждом шаге процедуры разбиения методом k-средних вычислять функционалы качества полученного разбиения:

𝐹1 – сумма по всем кластерам квадратов расстояний элементов кластеров до центров соответствующих кластеров;

𝐹2 – сумма по всем кластерам внутрикластерных расстояний между элементами кластеров;

𝐹3 – сумма по всем кластерам внутрикластерных дисперсий (относительно центров кластеров).

1. Промежуточные и итоговый результаты разбиения по возможности представлять в табличном виде, включая значения функционалов качества разбиения.
2. Разбиение на кластеры проиллюстрировать графически.
3. Полученные результаты содержательно проинтерпретировать, в том числе в сопоставлении с результатами работы метода *k*-средних.

# Выполнение работы.

Для начала работы алгоритма следует выбрать способ выбора точек для центров окружностей. Введём параметр дельта, который будет равен радиусу окрестности вокруг точки, в который считается количество соседей. После этого каждый раз из ещё необработанных точек выбирается такая, у которой в этой самой окрестности самое большое количество других точек из текущего множества. Если ни у одной точки в соседях никого не осталось, то дельта увеличивается на 1. Если дельта достигла размера радиуса, то автоматически выбирается первая из последовательности точка.

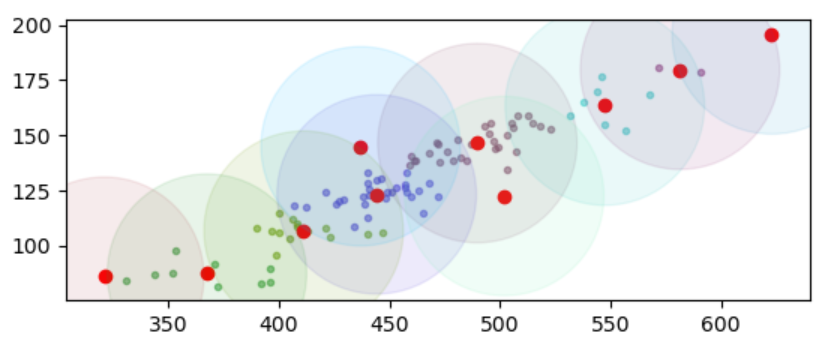
После нахождения центра окружности в работу вступает основная часть алгоритма сгущения. В итоге получаем кластеры с центрами. После этого нужно найти устойчивое разбиение, то есть такое, которые при увеличении радиуса не изменится. При постепенном увеличении радиуса на 0.7% получим разбиение из 10 кластеров с помощью окружностей радиусом 42.29. Картинка этого разбиения представлена на рисунке 1.

Рисунок 1 – Устойчивое разбиение

На каждом шаге с увеличением радиуса будет считать текущие функционалы и разместим результаты в таблицу:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **R** | **Число кластеров** | **𝐹1** | **𝐹2** | **𝐹3** |
| 3.23 | 89 | 14.39 | 14.39 | 8.12 |
| 6.02 | 75 | 77.28 | 114.78 | 76.63 |
| 8.81 | 59 | 212.74 | 385.61 | 319.76 |
| 11.60 | 48 | 363.55 | 1228.83 | 509.78 |
| 14.39 | 38 | 548.94 | 2810.27 | 800.27 |
| 17.18 | 32 | 681.15 | 3714.34 | 1132.63 |
| 19.97 | 28 | 819.56 | 5422 | 1276.25 |
| 22.76 | 22 | 984.86 | 7181.00 | 1687.08 |
| 25.55 | 23 | 1071.87 | 9200.04 | 1832.36 |
| 28.34 | 17 | 1292.65 | 13328.81 | 2259.59 |
| 31.13 | 14 | 1438.29 | 14722.12 | 2568.62 |
| 33.92 | 12 | 1629.96 | 20350.12 | 2712.77 |
| 36.71 | 12 | 1677.84 | 22405.97 | 2777.10 |
| 39.50 | 10 | 1820.48 | 26324.19 | 2859.14 |
| 42.29 | 10 | 1924.79 | 30209.38 | 2844.59 |

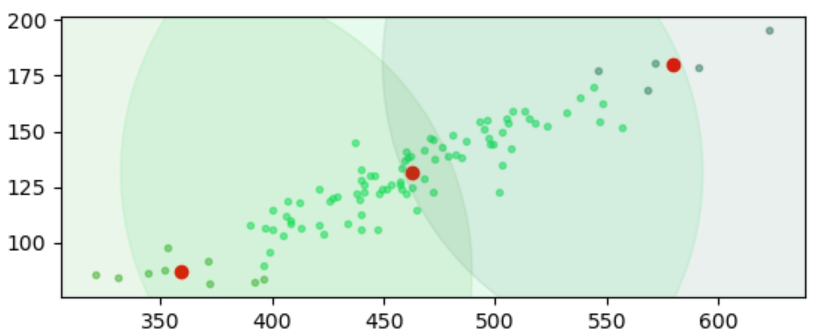
Если увеличить шаг интервала до 2%, получим устойчивое разбиение при радиусе окружности в 130.48. Картинка такого разбиения показана на рис. 2.

Рисунок 2 – Устойчивое разбиение

Функционалы ведут себя ожидаемо также – непреклонно увеличиваются с каждой итерацией. Зафиксируем их на конечном шаге: R1 = 4868, R2 = 253660, R3 = 5950.

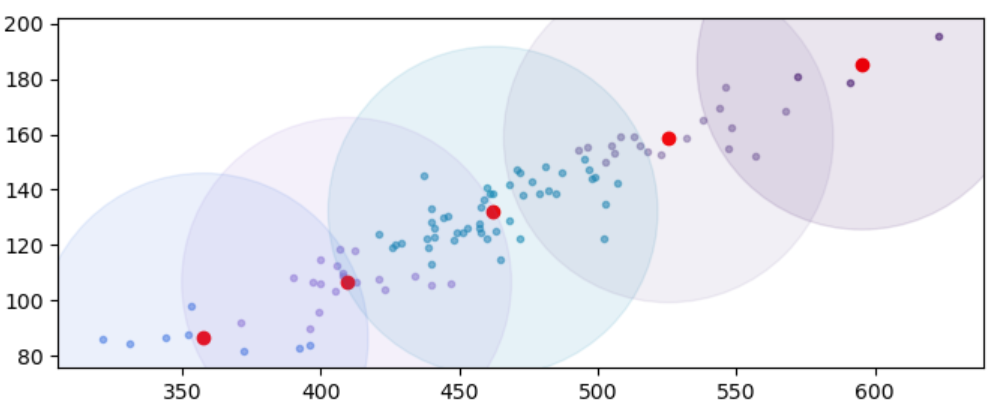
Теперь подберём такой минимальный радиус (хотя бы для неустойчивый), чтобы количество кластеров было равно количеству кластеров из прошлой работы, то есть пяти. После недолгого подбора коэффициентов получаем радиус размером приблизительно в 59,5. Изобразим разбиение на рис. 3.

Рисунок 3 – Случай кластеров

Получили интересную картинку. Для сравнения с прошлой лабораторной зафиксируем центры окружностей и центры функционалов:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лаб | Центр 1 | Центр 2 | Центр 3 | Центр 4 | Центр 5 | F1 | F2 | F3 |
| 7 | (357.63; 86.43) | (409.32; 106.54) | (462.19; 132.21) | (525.55; 158.74) | (595.33; 185.03) | 2717.18 | 61631 | 4364.41 |
| 6 | (366.09; 88.11) | (421.48; 122.91) | (462.34; 132.70) | (512.09; 153.00) | (574.00; 178.37) | 2369.39 | 36764 | 4141.25 |

# Выводы.

В процессе выполнения лабораторной работы рассматриваемая выборочная совокупность была разбита на 10, 3 и 5 кластеров с помощью метода поиска сгущений.

По таблице видно, что значения функционалов увеличиваются с каждым увеличением радиуса окружности. Это легко объясняется тем, что при каждом таком увеличении в кластеры будут всё дальние от центра точки и общая «кучность» будет ухудшаться. В другом устойчивом разбиении наблюдаем аналогичную ситуацию, только теперь получилось всего 3 кластера.

Для сравнения с методом k-средних нашли такой минимальный радиус окружности, при котором также получаем 5 кластеров, как и в прошлой работе. Сравнивая картинки можно сделать вывод, что центры окружностей получились также лежащими примерно на одной прямой. Действительно, сравнимая центры с 6 предыдущей работой можно отметить их большое сходство (за исключением последнего кластера, но точки в нем отстоят довольно далеко, не мешая общей тенденции). Это может говорить о наличии некой зависимости между параметрами, гипотеза о которой частично подтвердилась в 5 работе.

Для сравнения качества разбиения двух алгоритмов сравним их функциональные значения в случае 5 кластеров. При использованных параметрах алгоритмов можно сказать, что метод k-средних в обоих вариациях справляется лучше, чем метод сгущения, причём по всем параметрам.

# 

# ПРИЛОЖЕНИЕ А КОД ПРОГРАММЫ

import util

from math import sqrt

import matplotlib.pyplot as plt

from matplotlib.lines import Line2D

import copy

import numpy as np

import lab1

def getDistance(point\_1, point\_2):

    # Евклидово расстояние с поправочным на масштаб коэффициентом

    return sqrt((pow(point\_2[0] - point\_1[0], 2) + pow(ncf\*(point\_2[1] - point\_1[1]),2)))

def compareClusters(list\_1, list\_2):

    if len(list\_1) != len(list\_2):

        return False

    for i in range(len(list\_1)):

        if len(list\_1[i]) != len(list\_2[i]):

            return False

        for j in range(len(list\_2[i])):

            if list\_1[i][j] != list\_2[i][j]:

                return False

    return True

def recalcClusterCenter(cluster):

    mid\_x = sum([elem[0] for elem in cluster]) / len(cluster)

    mid\_y = sum([elem[1] for elem in cluster]) / len(cluster)

    return [mid\_x, mid\_y]

def findNewPopularPoint(points, delta):

    if len(points) == 1:

        return 0

    index = -1

    maxNeig = 1

    for i, cur\_point in enumerate(points):

        num = len([point for point in points if getDistance(cur\_point, point) <= delta])

        if num > maxNeig:

            maxNeig = num

            index = i

    return index

def F\_1(clusters, centers):

    result = 0

    for i, cluster in enumerate(clusters):

        for elem in cluster:

            result += getDistance(elem, centers[i])

    return result

def F\_2(clusters):

    result = 0

    for cluster in clusters:

        for i in range(len(cluster)):

            for j in range(i, len(cluster)):

                result += getDistance(cluster[i], cluster[j])

    return result

def F\_3(clusters, centers):

    result = 0

    for i, cluster in enumerate(clusters):

        sigma = 0

        for elem in cluster:

            sigma += pow(getDistance(elem, centers[i]), 2)

        sigma /= len(cluster)

        result += sigma

    return result

def makeClusters(data1, data2, R):

    size = len(data1)

    history = []

    points = list(zip(data1, data2))

    clusters = []

    iters = 0

    delta = 1

    points\_to\_process = points[:]

    while len(points\_to\_process) > 0:

        iters += 1

        index = np.random.randint(0, len(points\_to\_process))

        while True:

            index = findNewPopularPoint(points\_to\_process, delta)

            if index == -1:

                delta += 1

                if (delta > R1):

                    index = 0

                    break

            else:

                break

        center = points\_to\_process[index]

        points\_in\_cluster = [point for point in points\_to\_process if getDistance(point, center) < R]

        iter = 0

        while True:

            iter += 1

            center\_new = recalcClusterCenter(points\_in\_cluster)

            if (abs(center[0] - center\_new[0]) + abs(center[1] - center\_new[1])) < 0.001:

                break

            points\_in\_cluster = [point for point in points\_to\_process if getDistance(point, center\_new) < R]

            center = center\_new

            if iter == 100:

                break

        for cl\_point in points\_in\_cluster:

            for point in points\_to\_process:

                if (abs(cl\_point[0] - point[0]) + abs(cl\_point[1] - point[1])) < 0.001:

                    points\_to\_process.remove(cl\_point)

        clusters.append(points\_in\_cluster)

    return clusters

def plotClusters(clusters, clusterMeans, R):

    ax = plt.axes(aspect='equal')

    for i, cluster in enumerate(clusters):

        x, y = list(zip(\*cluster))

        color = np.random.rand(3,)

        color[0] = (color[0]\*255 % 192) / 255

        plt.scatter(x, y, marker=".", color=color, alpha=0.5)

        ax.add\_artist(plt.Circle(clusterMeans[i], R, color=color, alpha=0.1))

        xs = [mean[0] for mean in clusterMeans]

        ys = [mean[1] for mean in clusterMeans]

    plt.scatter(xs, ys, color = 'r', marker='o')

    plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

    n = lab1.selection\_size

    general\_population = lab1.read\_data(filename=lab1.data\_file\_name)

    sample\_density = lab1.get\_sample\_first(general\_population, n)

    sample\_elastic = lab1.get\_sample\_second(general\_population, n)

    sample\_2D = list(zip(sample\_density, sample\_elastic))

    min\_elem\_x, max\_elem\_x = min(sample\_density), max(sample\_density)

    min\_elem\_y, max\_elem\_y = min(sample\_elastic), max(sample\_elastic)

    ncf = (max\_elem\_x - min\_elem\_x) / (max\_elem\_y - min\_elem\_y)

    minR = np.Inf

    maxR = -np.Inf

    for p1 in sample\_2D:

        for p2 in sample\_2D:

            d = getDistance(p1, p2)

            if 0 < d < minR:

                minR = d

            if d > maxR:

                maxR = d

    print("Минимальный радиус:", minR)

    print("Максимальный радиус:", maxR)

    R1 = 59.5

    # clusters = makeClusters(sample\_density, sample\_elastic, R1, maxIters=10)

    # centers = [recalcClusterCenter(cluster) for cluster in clusters]

    # print("Clusters num: {0}".format(len(clusters)))

    # print(F\_1(clusters, centers))

    # print(F\_2(clusters))

    # print(F\_3(clusters, centers))

    # print(sorted(centers))

    # plotClusters(clusters, centers, R1)

    # exit(0)

    old\_clusters = []

    for R in np.linspace(minR, maxR, 1000): # 68 400 200

        clusters = makeClusters(sample\_density, sample\_elastic, R, maxIters=10)

        centers = [recalcClusterCenter(cluster) for cluster in clusters]

        print("Clusters num: {0}".format(len(clusters)))

        print("F1: {:.2f}".format(F\_1(clusters, centers)))

        print("F2: {:.2f}".format(F\_2(clusters)))

        print("F3: {:.2f}".format(F\_3(clusters, centers)))

        print("\n")

        # if compareClusters(clusters, old\_clusters):

        if len(clusters) == 5:

            # Устойчивое!

            print("success! ", R, "\nSteps:", steps)

            print(sorted(centers))

            print("R: {:.2f}".format(R))

            plotClusters(clusters, centers, R)

            break

        old\_clusters = copy.deepcopy(clusters)

    plt.show()