**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по лабораторной работе №6**

**по дисциплине «Статистические методы обработки**

**экспериментальных данных»**

**Тема: Кластерный анализ. Метод k-средних.**

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

Санкт-Петербург

2020

**Цель работы.**

Освоение основных понятий и некоторых методов кластерного анализа.

**Постановка задачи.**

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 k-средних.
2. На каждом шаге процедуры разбиения методом k-средних вычислять функционалы качества полученного разбиения:

* сумма по всем кластерам квадратов расстояний элементов кластеров до центров соответствующих кластеров;
* сумма по всем кластерам внутрикластерных расстояний между элементами кластеров;
* сумма по всем кластерам внутрикластерных дисперсий (относительно центров кластеров).

1. Реализовать метод k-средних в двух вариантах:

3.1. пересчет центра кластера осуществляется после каждого изменения его состава;

3.2. пересчет центра кластера осуществляется лишь после того, как будет завершен просмотр всех данных (шаг процедуры).

1. Промежуточные расчеты (по результатам каждой итерации) по возможности представлять в табличном виде, включая текущие значения функционалов качества разбиенияэ
2. Разбиение на кластеры проиллюстрировать графически
3. Полученные результаты содержательно проинтерпретировать.

**Ход работы.**

Для начала работы нужно было найти первые приближения возможных центров кластеров. Оно было выполнено с помощью нахождения средних значений в равных по размаху диапазонах для каждой переменной.

Далее при сравнении размахов выборок по каждому признаку можно заметить, что они заметно отличаются. Из-за этого при подсчёте евклидового расстояния вклад по первому признаку будет заметно выше, чем по второму. Поэтому при расчёте расстояния между точками значения второго признака будет умножаться на поправочный коэффициент, равный отношению размахов первой выборки ко второй. В абсолютной величине он будет равняться ≈ 2.65

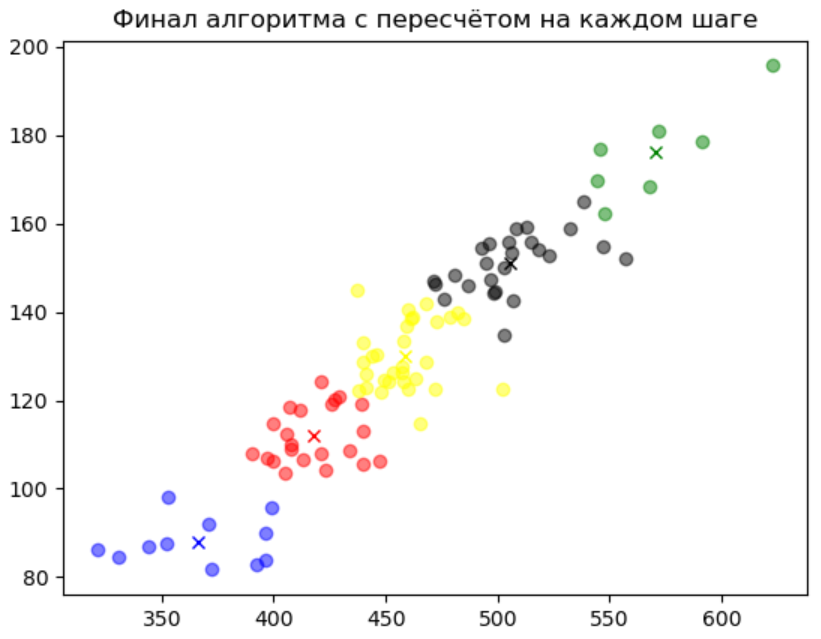
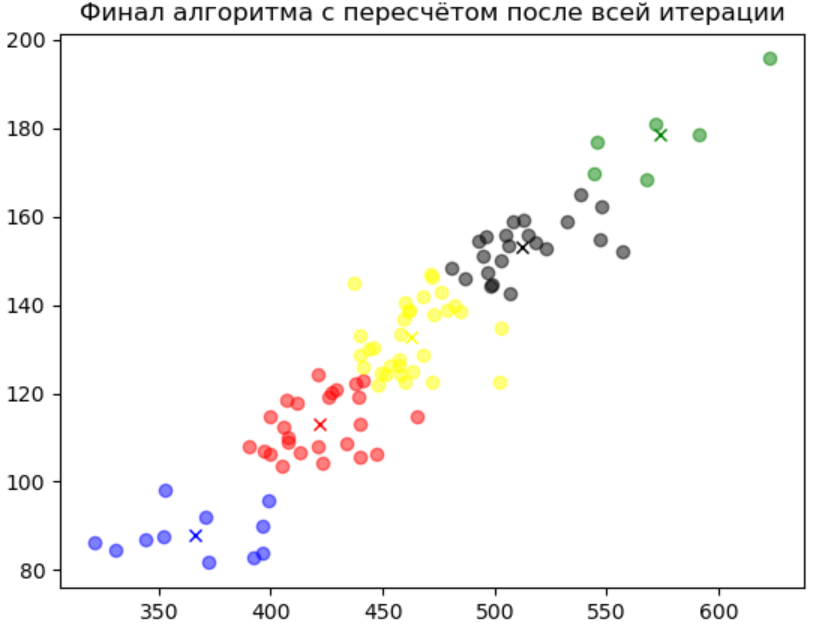
После выполнения подготовительных действий, используя реализованный метод *k*-средних было выполнено разбиение на 5 кластеров, причём двумя способами: с пересчётом центров кластеров сразу после изменения их наполнения и с пересчётом только после полной итерации. Графические изображения результатов представлены на рис. 1 и на рис. 2 соответственно.

Рисунок 1 – Результат первого алгоритма

Рисунок 2 – Результат второго алгоритма

На этих рисунках точки, относящиеся к одному кластеру закрашены одним цветом. Центры этих кластеров показаны крестиками.

Составим таблицы с промежуточными данными для каждой итерации алгоритмов разбиения. Таблица для первого алгоритма с пересчётом на каждом шаге:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Шаг | Центр 1 | Центр 2 | Центр 3 | Центр 4 | Центр 5 | F1 | F2 | F3 |
| 0 | (353.00; 98.00) | (412.00; 117.90) | (473.00; 137.90) | (532.00; 158.70) | (591.00; 178.50) | - | - | - |
| 1 | (366.09; 88.11) | (416.48; 112.22) | (462.12; 131.13) | (509.64; 152.72) | (576.43; 178.39) | 2334.36 | 36199 | 3951.53 |
| 2 | (366.09; 88.11) | (417.86; 111.95) | (458.61; 130.16) | (505.60; 150.99) | (570.29; 176.07) | 2324.37 | 35339 | 4195.10 |

Здесь алгоритм закончил работу на третьем шаге, т.к. полученные на этом шаге кластеры совпали с кластерами на предыдущем. Функциональные качества разбиения F1 иF2 стали лучше, а в то время как F3 стало хуже.

Таблица для второго алгоритма с пересчётом после всей итерации:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Шаг | Центр 1 | Центр 2 | Центр 3 | Центр 4 | Центр 5 | F1 | F2 | F3 |
| 0 | (353.00; 98.00) | (412.00; 117.90) | (473.00; 137.90) | (532.00; 158.70) | (591.00; 178.50) | - | - | - |
| 1 | (366.09; 88.11) | (422.58; 113.96) | (470.94; 136.31) | (522.24; 157.09) | (581.83; 179.85) | 2422.33 | 41117 | 3957.54 |
| 2 | (366.09; 88.11) | (423.19; 113.73) | (467.94; 134.97) | (518.78; 156.12) | (580.00; 180.12) | 2461.18 | 41208 | 4204.10 |
| 3 | (366.09; 88.11) | (422.5; 113.25) | (464.09; 133.90) | (516.29; 154.35) | (580.00; 180.12) | 2413.35 | 38800 | 4173.75 |
| 4 | (366.09; 88.11) | (422.5; 113.25) | (462.81; 133.05) | (512.09; 153.00) | (574.00; 178.37) | 2375.06 | 36620 | 4152.49 |
| 5 | (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 |

Здесь алгоритм закончил работу на шестом шаге, т.к. полученные на этом шаге кластеры совпали с кластерами на предыдущем. Видно, что все функциональные качества разбиения деградировали на втором шаге, но затем стали постепенно улучшаться. Но F3 так и не вернулся на прежние позиции.

**Выводы.**

В ходе выполнения работы проведены разбиения двумерной выборки на кластеры с помощью алгоритма k-средних в двух его вариациях. Результаты визуализированы в виде графиков с закрашиванием точек, относящихся с к разным кластерам, в разный цвет. В качестве метрики определения разности двух точек было выбрано расстояние евклидово. На каждом шаге каждого из алгоритмов были посчитаны значения функциональные качества разбиения. Из динамики их изменения можно сделать вывод, что каждая итерация алгоритма улучшает суммарные показатели «кучности» в итоговых кластерах, но не по каждой метрике в отдельности. Повторные тестирования для разного количества кластеров также не помогли установить чёткую ясность в этом вопросе, только лишь показали тенденцию к более быстрой сходимости первого алгоритма.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД**

from math import sqrt

import matplotlib.pyplot as plt

import copy

import lab1

K\_num = 3

def calcFirstCenters(sameple):

# Прикидываем центры для начала работы

min\_elem, max\_elem = min(sameple), max(sameple)

step = (max\_elem - min\_elem) / (K\_num)

borders = [min\_elem]

for i in range(K\_num):

borders.append(borders[i] + step)

centers = []

for i in range(K\_num):

centers.append((borders[i] + borders[i+1]) / 2)

return centers

def findRealClosestPointIndex(im\_point, sample):

# Ищем наиболее близкие точки в реальной выборке наивным поиском

indexOfMin = 0

currMinDistance = getDistance(im\_point, sample[0])

for i in range(1, len(sample)):

dist = getDistance(im\_point, sample[i])

if dist < currMinDistance:

currMinDistance = dist

indexOfMin = i

return indexOfMin

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 findClosestCluster(point, centers):

ds = [getDistance(point, center) for center in centers]

return ds.index(min(ds))

def recalcClusterCenter(cluster, cl\_index, centers):

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

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

centers[cl\_index] = [mid\_x, mid\_y]

def recalcAllCenters(clusters, centers):

for i, cluster in enumerate(clusters):

recalcClusterCenter(cluster, i, centers)

def initData(sample\_density, sample\_elastic):

# Инициализируем

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

im\_centers = list(zip(calcFirstCenters(sample\_density), calcFirstCenters(sample\_elastic)))

pointsPull = sample\_2D[:]

centers = []

for im\_center in im\_centers:

indexOfRealPoint = findRealClosestPointIndex(im\_center, pointsPull)

print(indexOfRealPoint)

centers.append(sample\_2D[indexOfRealPoint])

del pointsPull[indexOfRealPoint]

clusters = [[centers[i]] for i in range(K\_num)]

return clusters, centers, pointsPull

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

if len(clusters\_1) != len(clusters\_2):

return False

for i in range(K\_num):

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

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

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

return False

else:

return False

return True

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 k\_algo\_1(sample\_density, sample\_elastic):

# Первый вариант - пересчитываем центр после каждого обновления кластера

clusters, centers, pointsPull = initData(sample\_density, sample\_elastic)

print("Начальные центры:")

for elem in centers:

print("({0:.2f}; {1:.2f})".format(elem[0], elem[1]), end=" ")

print(end="\n")

old\_clusters = []

for i in range(10):

for point in pointsPull:

index = findClosestCluster(point, centers)

clusters[index].append(point)

recalcClusterCenter(clusters[index], index, centers)

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

if compareClusters(clusters, old\_clusters):

print("Найдено устойчивое состояние! {0} совпадает с {1}".format(i, i+1))

colors = ["blue", "red", "yellow", "black", "green", "cyan", "grey", "blue", "red", "yellow"]

fig, ax = plt.subplots()

for j, cluster in enumerate(clusters):

for point in cluster:

ax.plot(point[0], point[1], 'o', color=colors[j], alpha=0.5)

for j, center in enumerate(centers):

ax.plot(center[0], center[1], 'x', color=colors[j])

plt.title("Финал алгоритма с пересчётом на каждом шаге")

plt.show()

break

else:

print("{} шаг процедуры:".format(i+1))

for elem in centers:

print("({0:.2f}; {1:.2f})".format(elem[0], elem[1]), end=" ")

print(end="\n")

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

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

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

print(end="\n")

old\_clusters = copy.deepcopy(clusters)

clusters.clear()

for \_ in range(len(old\_clusters)):

clusters.append([])

def k\_algo\_2(sample\_density, sample\_elastic):

print("Начало алгоритма с пересчётом центра после просмотра всех данных")

clusters, centers, pointsPull = initData(sample\_density, sample\_elastic)

print("Начальные центры:")

for elem in centers:

print("({0:.2f}; {1:.2f})".format(elem[0], elem[1]), end=" ")

print(end="\n")

old\_clusters = []

for i in range(10):

for point in pointsPull:

index = findClosestCluster(point, centers)

clusters[index].append(point)

recalcAllCenters(clusters, centers) # Пересчитываем центры после подсчёт статистики

if compareClusters(clusters, old\_clusters):

print("Найдено устойчивое состояние! {0} совпадает с {1}".format(i, i+1))

colors = ["blue", "red", "yellow", "black", "green", "cyan", "grey", "blue", "red", "yellow"]

fig, ax = plt.subplots()

for j, cluster in enumerate(clusters):

for point in cluster:

ax.plot(point[0], point[1], 'o', color=colors[j], alpha=0.5)

for j, center in enumerate(centers):

ax.plot(center[0], center[1], 'x', color=colors[j])

plt.title("Финал алгоритма с пересчётом после всей итерации")

plt.show()

break

else:

print("{} шаг процедуры:".format(i+1))

for elem in centers:

print("({0:.2f}; {1:.2f})".format(elem[0], elem[1]), end=" ")

print(end="\n")

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

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

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

print(end="\n")

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

old\_clusters = copy.deepcopy(clusters)

clusters.clear()

for \_ in range(len(old\_clusters)):

clusters.append([])

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)

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)

print(ncf)

k\_algo\_1(sample\_density, sample\_elastic)

k\_algo\_2(sample\_density, sample\_elastic)

# import matplotlib.pyplot as plt

# fig, ax = plt.subplots()

# ax.plot(sample\_density, sample\_elastic, 'ok')

# for point in im\_centers:

# ax.plot(point[0], point[1], 'og')

# for point in centers:

# ax.plot(point[0], point[1], 'ob')

# plt.show()