Информатика. Методическое пособие по задаче №27.

Содержание

1 Теоретическое введение

1.1 Основные сведения о задаче

Задание №27 является новым и направлено на решение задач анализа данных. В качестве ключевых навыков, которыми нужно обладать для решения №27, можно выделить:

- Сбор первичных данных;
- Очистка и оценка качества данных;
- Выбор и построение модели;
- Преобразование данных;
- Визуализация данных;
- Интерпретация результатов.

В демоверсии-2025 условие задачи было направлено на кластеризацию данных. Именно поэтому в основной волне экзамена ожитается похожий уровень задания.

Кластеризация – это метод машинного обучения, который группирует объекты в кластеры на основе их схожести. В данной задаче:

- **Объекты** – это звёзды с координатами (x, y);
- **Кластер** – это множество звёзд, лежащих внутри заданной области;
- **Центроид** и **Антицентр** – звезда, максимизирующая или минимизирующая сумму расстояний до других звёзд кластера (в зависимости от заданного условия).

Ключевые свойства:

- Каждый объект принадлежит ровно одному кластеру;
- Кластеры не пересекаются.

Помимо центроида в задачах, составленных методистами курса, может встречаться формулировка **периферoида** – звезды, максимизирующей сумму расстояний до других звёзд кластера.

При решении задания №27 на курсе были использованы следующие способы:

- Базовый – разделение на кластеры с помощью координат.
  В данном способе необходимо разделить кластеры по координатам x и y, никаких дополнительных действий не требуется.
  Минус способа – может плохо работать в случае, если даны мусорные данные или кластеры имеют произвольную форму.
- Разделение на кластеры прямыми.
  В данном способе перед кластеризации строятся прямые линии, которые отделяют кластеры, расположенные близко друг к другу.
  Аналогично к минусам можно отнести проблемы в разделении, если даны мусорные данные или кластеры имеют произвольную форму.
- Разделение на кластеры окружностями.
  В данном способе перед кластеризации строятся уравнения окружностей.
- Метод DBSCAN.
  Способ предполагает создания функции, которая по алгоритму DBSCAN будет разделять точки по кластерам.
  Способ очень хорошо справляется с произвольной формой кластеров и мусорными точками при подборе правильного расстояния между точками.

Перед реализацией каждого способа решения необходимо выполнить стандартный алгоритм: открыть Excel файл и построить диаграмму. Построение диаграммы выполняется следующим способом:

Вставка → **Диаграммы** → **Точечная диаграмма**

После построения диаграммы необходимо ее проанализировать и на основе таких факторов как: расположение кластеров, наличие мусорных точек, форма кластеров, выбрать оптимальный способ обработки данных.

Рассмотрим подробнее каждый из способов обработки данных в разделе **«Разбор прототипов задач»**.

1.2 Алгоритм DBSCAN

Алгоритм ***DBSCAN*** – это метод кластеризации, который позволяет выделять кластеры различной формы и размера. Принцип работы алгоритма следующий:

- Перед началом работы алгоритма необходимо определить радиус R, в пределах которого будут рассматриваться соседние точки.
- Алгоритм начинается с выбора произвольной точки из набора данных, от которой будут искаться все остальные точки кластера.
- Для выбранной точки ищутся все точки, которые находятся на расстоянии R.
- Процесс продолжается до тех пор, пока все точки на заданном расстоянии не будут обработаны. Как только все возможные точки добавлены в кластер, алгоритм переходит к следующей непроверенной точке в наборе данных (то есть начинает формировать следующий кластер).
- Алгоритм повторяет шаги для всех непроверенных точек в наборе данных, пока не будут обработаны все точки.

2 Разбор прототипов задач

Примечание: первые три прототипа задач будут показаны без мусорных точек и имеют одинаковое условие.

Более сложный по своей сути алгоритм DBSCAN будет включать в себя мусорные точки.

Учёный решил провести кластеризацию некоторого множества звёзд по их расположению на карте звёздного неба. Кластер звёзд – это набор звёзд (точек) на графике, лежащий внутри круга радиусом R. Каждая звезда обязательно принадлежит только одному из кластеров.

Истинный центр кластера, или центроид, – это одна из звёзд на графике, сумма расстояний от которой до всех остальных звёзд кластера минимальна.

Под расстоянием понимается расстояние Евклида между двумя точками $A(x_1, y_1)$ и $B(x_2, y_2)$ на плоскости, которое вычисляется по формуле:

$$d(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

В файле А хранятся данные о звёздах **трёх** кластеров, где $R = 3$. В каждой строке записана информация о расположении на карте одной звезды: сначала координата x, затем координата y. Значения даны в условных единицах, которые представлены вещественными числами. Известно, что количество звёзд не превышает 4119.

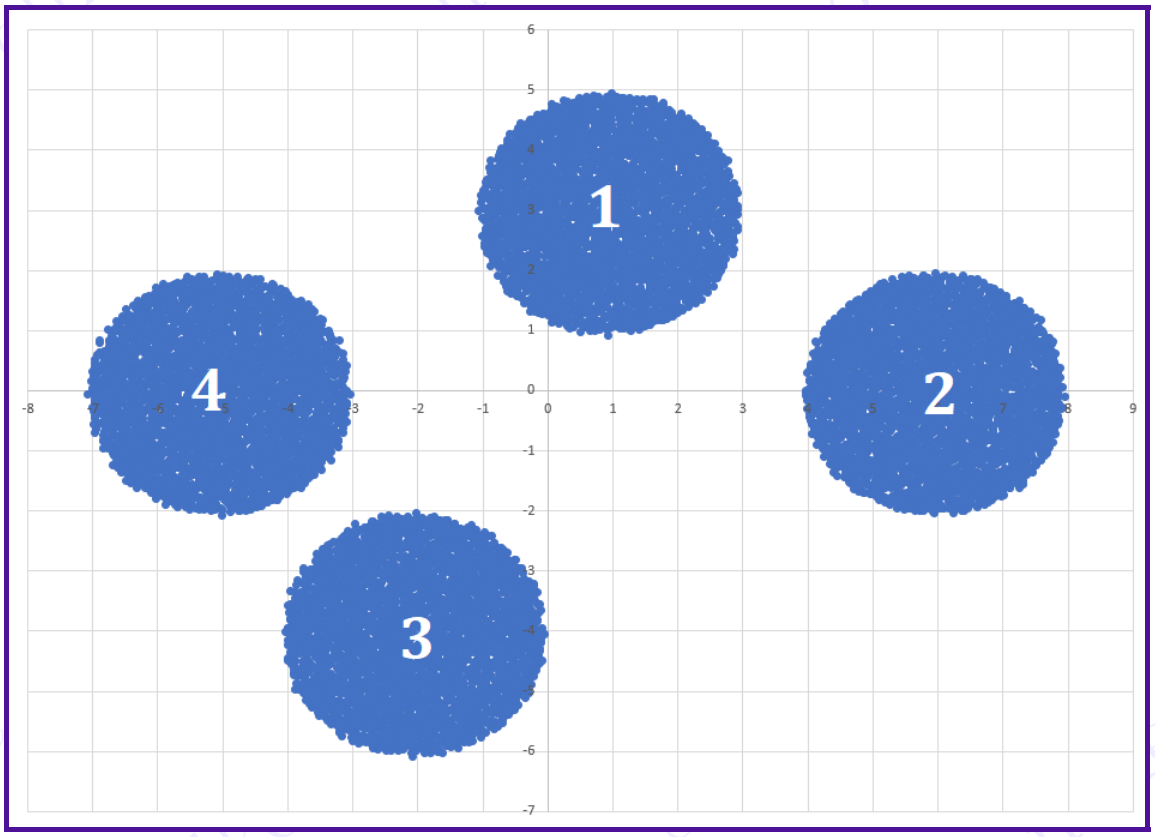
В файле Б хранятся данные о звёздах **четырёх** кластеров, где $R = 2$. Известно, что количество звёзд не превышает 11268. Структура хранения информации о звездах в файле Б аналогична файлу А.

Для каждого файла определите координаты центра каждого кластера, затем вычислите два числа: P_x — среднее арифметическое абсцисс центров кластеров, и P_y – среднее арифметическое ординат центров кластеров.

В ответе запишите четыре числа через пробел: сначала целую часть произведения $P_x \cdot 250$ для файла А и $P_y \cdot 250$ для файла А, далее целую часть произведения $P_x \cdot 300$ для файла Б и $P_y \cdot 300$ для файла Б.

2.1 Кластеризация по координатам, прямой и окружности

Для начала визуализируем данные нам кластеры и пронумеруем их:



Как видно из диаграммы, кластера под номерами 1 и 2 легко разделяются по координатам, но кластеры 3 и 4 находятся слишком близко друг к другу.

Разделим кластер 1 базовым способом – с помощью координат, кластер 2 – с помощью уравнения окружности, кластеры 3 и 4 – с помощью уравнения прямой, проходящей между ними.

1. Для начала отделим границами кластер 1: из диаграммы видно, что все точки, которые находятся в промежутках

$$-2 < x < 3.5 \text{ и } 0.5 < y < 6$$

принадлежат первому кластеру. При реализации программы будем пользоваться этими неравенствами.

2. Для составления уравнения окружности для кластера 2 воспользуемся форму-

лой:

$$(x - x_0)^2 + (y - y_9)^2 = R^2,$$

где (x_0, y_0) – центр окружности, а R – ее радиус. В нашем случае удобно взять $R \sim 2.5$, а центр – $(6; 0)$.

Таким образом, неравенство для второго кластера:

$$(x - 6)^2 + y^2 < 2.5^2$$

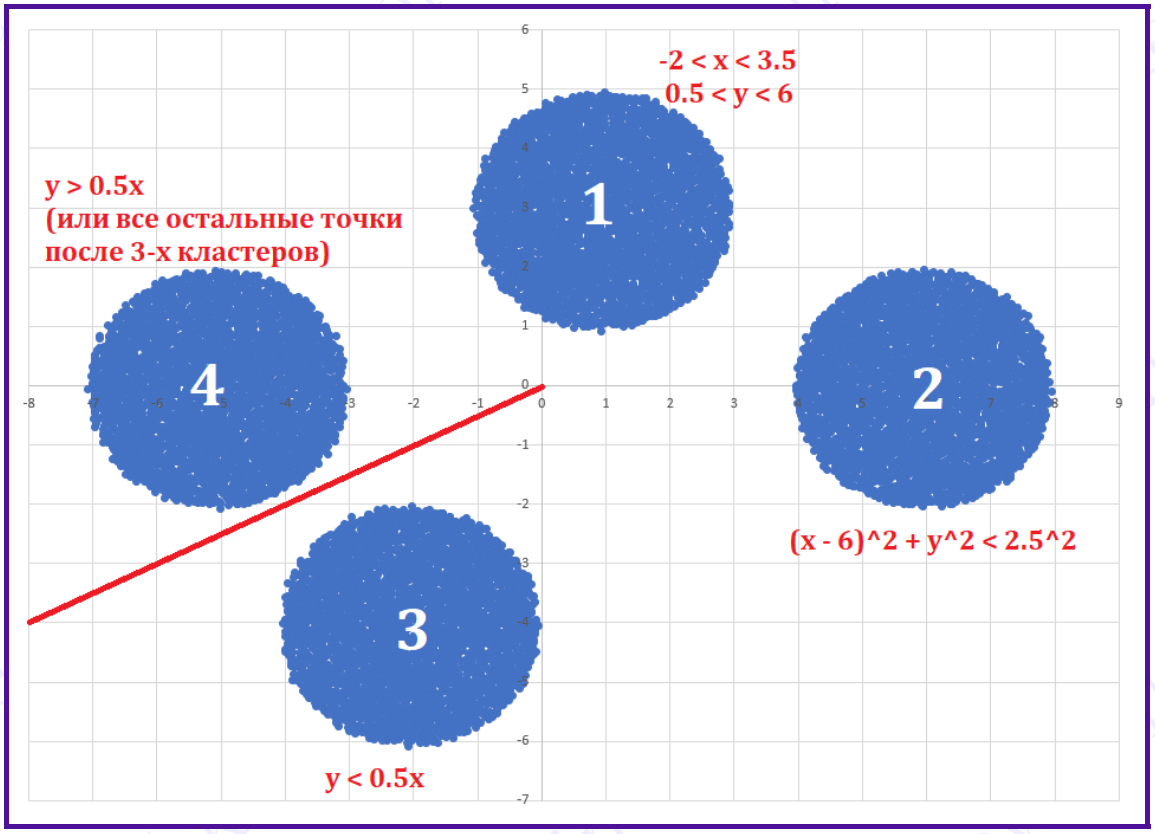
3. Для разделения 3 и 4 кластеров построим прямую по точкам $(0; 0)$ и $(-4; -2)$:

$$\frac{x}{0+4} = \frac{y}{0+2} \to y = \frac{x}{2}$$

Получаем, что третьему кластеру принадлежат все точки, удовлетворяющие неравенству:

$$y < \frac{x}{2}$$

Четвертому кластеру – все оставшиеся точки.



Важно!!!

При таком распределении важно, чтобы точки к 4 кластеру добавлялись в самый последний момент – после распределения точек по другим кластерам. Иначе может возникнуть ошибка.

Программная реализация задачи:

Решим задачу последовательно. Прежде всего считаем все данные из файла и создадим список для хранения кластеров после разделения:

```
f = open('27.txt')
n = f.readline() # читаем первую строку из файла и сохраняем в переменную n
clusters = [[] for i in range(4)] # списки для будущих кластеров
```

Следующим шагом последовательно будем проходится по строкам файла, читать две координаты – x и y, а затем проверять условия на соответствие кластерам:

```
for i in range(11268): # цикл для обработки 11268 звезд
# чтение строки, замена запятых на точки, разбиение строки на отдельные числа
    star = list(map(float, f.readline().replace(',', '.').split()))
    x, y = star[0], star[1] # star[0] - координата x, star[1] - координата y
    if -2 < x < 3.5 and 0.5 < y < 6: # принадлежность звезды к кластеру 1
        clusters[0].append(star) # добавление звезды в первый кластер
# проверка принадлежности звезды к кластеру 2
    elif (x - 6)**2 + y**2 < 2.5**2: # уравнение окружности
        clusters[1].append(star) # добавление звезды во второй кластер
# проверка принадлежности звезды к кластеру 3
    elif y < 0.5*x: # условие нахождения ниже прямой y = 0.5x
        clusters[2].append(star) # добавление звезды в третий кластер
    else: # все остальные звезды попадают в кластер 4
        clusters[3].append(star)  # добавление звезды в четвертый кластер
```

После разделения точек по кластерам остается дело за малым – найти центроид каждого:

```
# инициализация переменных для суммирования координат центроидов
sum_x = sum_y = 0 # суммы x и y координат всех центроидов
tx = ty = 0 # временные переменные для хранения текущего центроида
for i in clusters: # перебираем каждый кластер в списке clusters
# инициализируем минимальную сумму расстояний большим числом
    mn = 100000050000 # начальное значение для поиска минимума
# перебираем каждую звезду в текущем кластере как потенциальный центроид
    for j in i:
        x1, y1 = j # координаты текущей звезды
        sm = 0 # сумма расстояний от текущей звезды до всех других
#сСчитаем сумму расстояний от текущей звезды до всех остальных в кластере
        for k in i:
            x2, y2 = k
            sm += ((x2 - x1)**2 + (y2 - y1)**2)**0.5 # Еклидово расстояние
 # если текущая сумма расстояний меньше минимальной - обновляем центроид
        if sm < mn:
            mn = sm
            tx, ty = x1, y1 # запоминаем координаты нового центроида
# добавляем координаты найденного центроида к общим суммам
    sum_x += tx
    sum_y += ty
# вычисляем средние координаты центроидов и умножаем на 300
# int() используется для получения целой части результата
print(int(sum_x / len(clusters) * 300))  # среднее по x
print(int(sum_y / len(clusters) * 300))  # среднее по y
```

Итоговый полный вариант реализованной программы:

```
f = open('27.txt')
n = f.readline()
clusters = [[] for i in range(4)]
for i in range(11268):
    star = list(map(float, f.readline().replace(',', '.').split()))
    x,y = star[0],star[1]
    if -2 < x < 3.5 and 0.5 < y < 6:
        clusters[0].append(star)
    elif (x - 6)**2 + y**2 < 2.5**2:
        clusters[1].append(star)
    elif y < 0.5*x:
        clusters[2].append(star)
    else:
        clusters[3].append(star)
sum_x = sum_y = tx = ty = 0
for i in clusters:
    mn = 100000050000
    for j in i:
        x1, y1 = j
        sm = 0
        for k in i:
            x2, y2 = k
            sm += ((x2-x1)**2 + (y2-y1)**2)**0.5
        if sm < mn:
            mn = sm
            tx, ty = x1, y1
    sum_x += tx
    sum_y += ty
print(int(sum_x / len(clusters) * 300))
print(int(sum_y / len(clusters) * 300))
```

Важно!!!

В ФИПИ появились задачи, в которых вместо уловия:

*«**Истинный центр кластера**, или **центроид**, – это одна из звёзд на графике, сумма расстояний от которой до всех остальных звёзд кластера **минимальна**.»*

задано:

*«Будем называть **антицентром** кластера точку этого кластера, сумма расстояний от которой до всех остальных точек кластера **максимальна**.»*

Для такого условия суть решения никак не меняется. Единственное отличие, которое существует для решения задач подобного типа – поиск суммы расстояний.

Кроме этого в базе Школково уже существуют подобные задачи, но вместо слова **«антицентр»** используется слово **«периферoид»**.

Приведем пример поиска максимальной суммы расстояний если бы в задаче выше необходимо было найти **антицентр**:

```
# разделение на кластеры остается прежним
sum_x = sum_y = tx = ty = 0
for i in clusters:
    mx = -100000050000 # вместо минимума ищем максимум
    for j in i:
        x1, y1 = j
        sm = 0
        for k in i:
            x2, y2 = k
            sm += ((x2-x1)**2 + (y2-y1)**2)**0.5
        if sm > mx: # если сумма больше максимальной,
            mx = sm # обновляем максимальную сумму
            tx, ty = x1, y1
    sum_x += tx
    sum_y += ty
print(int(sum_x / len(clusters) * 300))
print(int(sum_y / len(clusters) * 300))
```

2.2 Разделение на кластеры с помощью метода DBSCAN

Рассмотрим задачу, содержащую кластеры с аномальными точками:

Финес и Ферб, вдохновлённые летними каникулами, решили построить трёхмерную модель звёздного неба и провести кластеризацию звёзд. Кластер звёзд — это набор точек, где каждая звезда находится от хотя бы одной другой на расстоянии не более R условных единиц. Каждая звезда принадлежит только одному кластеру.

Истинный центр кластера, или центроид, — это звезда, сумма расстояний от которой до всех остальных звёзд кластера минимальна.

Расстояние между точками $A(x_1, y_1)$ и $B(x_2, y_2)$ вычисляется по формуле:

$$d(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Аномалиями называются точки, расположенные более чем на одну условную единицу от всех точек кластеров. Аномалии в расчётах не учитываются.

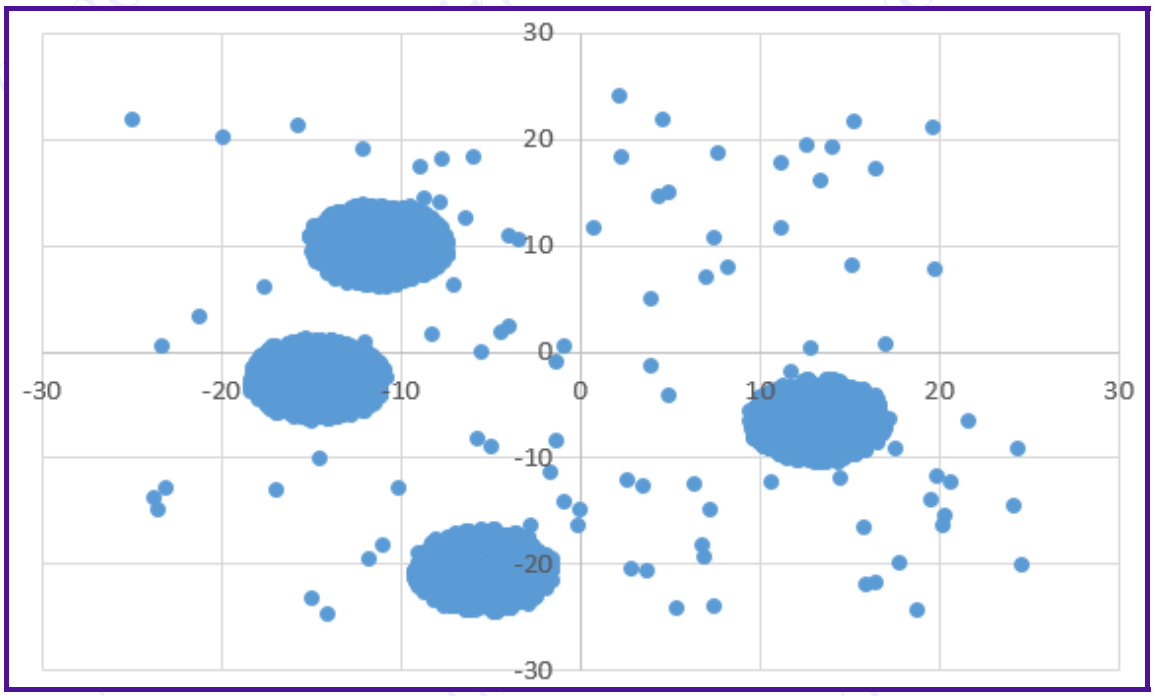
В файле A хранятся данные о звёздах **двух** кластеров, где $R = 1$ для каждого кластера. В каждой строке записаны координаты звезды: x, y. Значения — вещественные числа в условных единицах. Количество звёзд не превышает 1000.

В файле B хранятся данные о звёздах **четырёх** кластеров, где $R = 0.7$ для каждого кластера. Количество звёзд не превышает 5200. Структура данных аналогична файлу A.

Для каждого файла определите координаты центроида каждого кластера, затем вычислите P_{xy} — квадрат произведения средних арифметических абсцисс и ординат центроидов.

В ответе запишите два числа через пробел: сначала целую часть $\frac{P_{xy}}{4}$ для файла A, затем целую часть $P_{xy} \cdot 10$ для файла B.

Визуализируем данные, чтобы понять, возможно ли разделение базовыми методами:



Как видно из диаграммы, мусорных точек слишком много и некоторые из них расположены довольно близко к кластерам. Поэтому ни один способ из использованных ранее не сможет справиться с этой задачей.

Кроме того, в условии задачи сразу задан радиус, на котором находятся точки всех кластеров: $R = 0.7$ для файла B, который рассматривается. Используем это значение и реализуем программное решение задачи.

Выделим три этапа решения:

1. Реализация функции DBSCAN.
2. Разделение звезд по кластерам.
3. Поиск центроида.

Реализуем каждый из этапов последовательно.

Программное решение:

Прежде всего реализуем функцию **DBSCAN**, чтобы использовать ее в дальнейшем в любом месте кода.

```
from math import * # библиотека math для работы с функцией dist
# реализация алгоритма DBSCAN
def dbscan(a, r):
    cl = []  # инициализируем список для хранения кластеров
    while a:  # пока есть элементы в входном массиве 'a'
        cl.append([a.pop(0)])
        for i in cl[-1]:  # проходим по элементам последнего кластера
            for j in a[:]:
                if dist(i, j) <= r: # вычисляем расстояние между звездами
                    cl[-1].append(j)  # добавляем 'j' в текущий кластер
                    a.remove(j)  # удаляем 'j' из списка 'a'
    return cl #возвращаем список кластеров
```

Следующим шагом считаем все данные из файла и применим только что написанную функцию для кластеризации точек.

Дополнительно создадим список итоговых кластеров, число точек в которых превышает 10. Таким образом мы избавимся от тех кластеров, которые составлены из аномальных точек, то есть мусора.

```
f = open("27.txt")
a = [list(map(float, i.replace(",", ".").split())) for i in f]
cl = dbscan(a, 0.7) # алгоритм DBSCAN для кластеризации данных
cl_total = [] # создаем список для хранения только значимых кластеров
for i in cl: # фильтрация кластеров: оставляем только те, где больше 10 точек
    if len(i) > 10: # проверяем размер кластера
        cl_total.append(i) # добавляем в итоговый список
```

После разделения точек по кластерам остается дело за малым – найти центроид каждого:

```
# инициализация переменных для суммирования координат центроидов
sum_x = sum_y = 0  # суммы x и y координат всех центроидов

# перебираем каждый значимый кластер (из cl_total)
for cluster in cl_total:
# временные переменные для хранения текущего центроида
    tx = ty = 0
# инициализация минимальной суммы расстояний (очень большое число)
    mn = 10 ** 20 # рактически "бесконечность" для поиска минимума
# поиск оптимального центроида в текущем кластере
    for centroid in cluster:
        sm = 0 # сумма расстояний от текущей точки до всех других
# вычисляем сумму расстояний до всех точек кластера
        for star in cluster:
            sm += dist(centroid, star) # Евклидово расстояние
# если нашли точку с меньшей суммой расстояний - обновляем центроид,
        if sm < mn:
            mn = sm
            tx, ty = centroid[0], centroid[1] # запоминаем координаты
# добавляем координаты найденного центроида к общим суммам
    sum_x += tx
    sum_y += ty

# вычисление и вывод результата:
print(int(((sum_x / 4 * sum_y / 4) ** 2) * 10))
```

Итоговый полный вариант реализованной программы:

```
from math import *
def dbscan(a, r):
    cl = []
    while a:
        cl.append([a.pop(0)])
        for i in cl[-1]:
            for j in a[:]:
                if dist(i, j) <= r:
                    cl[-1].append(j)
                    a.remove(j)
    return cl
f = open("27.txt")
a = [list(map(float, i.replace(",", ".").split())) for i in f]
cl = dbscan(a, 0.7)
cl_total = []
for i in cl:
    if len(i) > 10: cl_total.append(i)
sum_x = sum_y = 0
for cluster in cl_total:
    tx = ty = 0, mn = 10 ** 20
    for centroid in cluster:
        sm = 0
        for star in cluster:
            sm += dist(centroid, star)
        if sm < mn:
            mn = sm
            tx, ty = centroid[0], centroid[1]
    sum_x += tx
    sum_y += ty
print(int(((sum_x / 4 * sum_y / 4) ** 2) * 10))
```

2.3 Задачи, содержащие третий параметр

Примечание: нельзя исключать вариант добавления задачи, которая будет содержать не только входные координаты точек (x, y), а дополнительный третий параметр.

Методистами Школково были сделаны задачи подобного типа, которые содержат в качестве третьего параметра **блеск звезды**.

Учёный решил провести кластеризацию некоторого множества звёзд по их расположению на карте звёздного неба, а также по блеску звезды. Кластер звёзд – это набор звёзд (точек) на графике, лежащий внутри круга с радиусом R. Каждая звезда, подходящая под заданный уровень блеска, обязательно принадлежит только одному из кластеров. Остальные звёзды не относятся к рассматриваемым кластерам.

Истинный центр кластера, или центроид, – это одна из звёзд кластера, сумма расстояний от которой до всех остальных звёзд кластера минимальна.

В файле А хранятся данные о звёздах четырех кластеров, где $R = 4$ для каждого кластера, а звёзды обладают блеском более 20 и менее 40 условных единиц, целая часть которых кратна 9. В каждой строке записана информация об уровне блеска звезды, а также о расположении на карте одной звезды: сначала координата x, затем координата y и наконец уровень блеска m. Значения даны в условных единицах, которые представлены вещественными числами. Известно, что количество звёзд не превышает 3000.

В файле Б хранятся данные о звёздах пяти кластеров, где $R = 2$ для каждого кластера, а звёзды обладают блеском более 45 и менее 98 условных единиц, целая часть которых оканчивается на четную цифру. Известно, что количество звёзд не превышает 15000. Структура хранения информации о звездах в файле Б аналогична файлу А.

Для каждого файла определите координаты центра каждого кластера, затем вычислите два числа: P_x — среднее арифметическое абсцисс центров кластеров, и P_y – среднее арифметическое ординат центров кластеров.

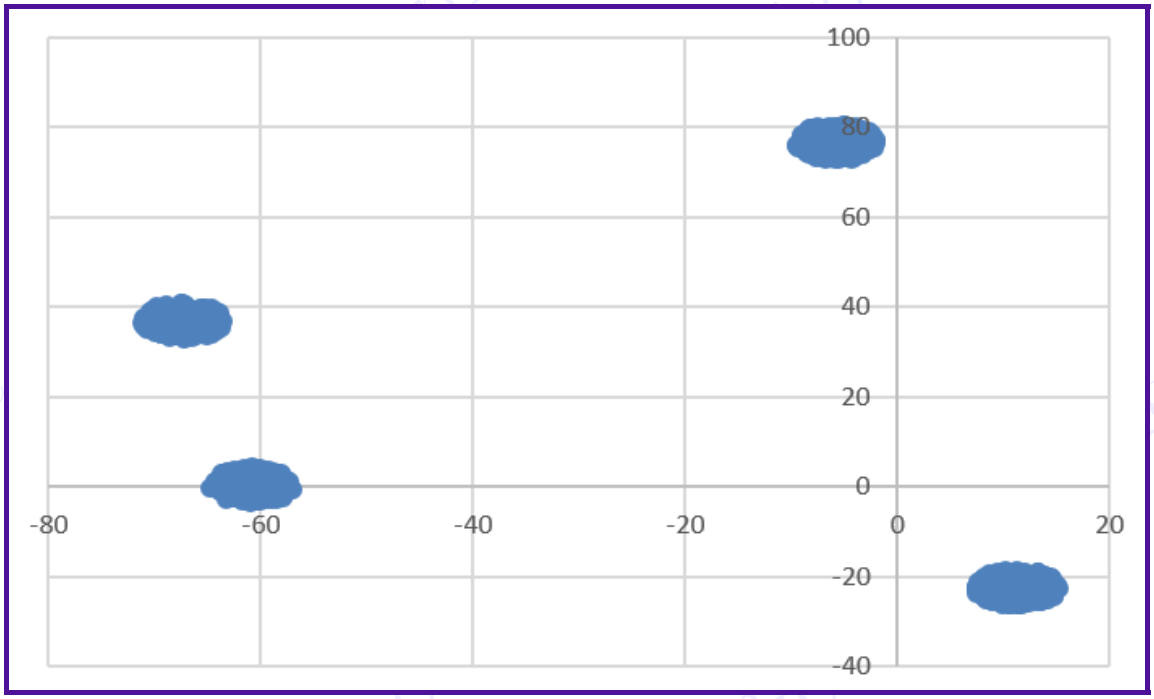
В ответе запишите четыре числа через пробел: сначала целую часть P_x для файла А и P_y для файла А, далее целую часть P_x для файла Б и P_y для файла Б.

Программное решение:

Поиск центроида в данной задаче аналогичен всем другим методам, самая главная задача – правильно разделить звезды на кластеры.

Самое главное отличие состоит в том, что при считывании данных из файла необходимо ввести дополнительную переменную, а затем согласно условию провести разделение на кластеры.

Рассмотрим часть программы, которая разделяет звезды на кластеры, на примере файла A. Перед написанием программы как обычно построим диаграмму для анализа расположения кластеров:



Как можно заметить по диаграмме, разделение на кластеры можно сделать следующим образом:

1) Для начала нужно оценить блеск звезды: проверить, чтобы он находится в диапазоне от 20 до 40 условных единиц и его целая часть была кратна 9;

2) Если ордината точки больше 60, сразу же относим точку к первому кластеру;

3) Иначе если ордината точки не больше 60, но больше 20, то точку можно отнести ко второму кластеру;

4) Иначе если абсцисса точки больше 0, относим её к третьему кластеру;

5) В другом случае точка точно принадлежит четвертому кластеру, так как на диаграмме нет аномальных (мусорных) точек.

```
f = open('27A.txt')
n = f.readline()  # считываем первую строку файла с названиями столбцов
a = [[] for i in range(4)]  # создаём список для кластеров
for line in f:
# считываем координату звезды и ее блеск
    x, y, m = list(map(float, line.replace(',', '.').split()))
# если блеск звезды находится в пределе от 20 до 40 условных единиц
# и целая часть значения блеска кратна 9, продолжаем обработку звезды
    if 20 < m < 40 and int(m) % 9 == 0:
        if y > 60: # если ордината >60, относим звезду в первый кластер
            a[0].append([x, y])
        elif y > 20: # иначе если ордината >20, относим звезду во второй
            a[1].append([x, y])
        elif x > 0: # иначе если абсцисса >0, относим звезду в третий
            a[2].append([x, y])
        else: # иначе звезда принадлежит четвертому (нет мусорных данных)
            a[3].append([x, y])
```

Важно!!!

Нужно предельно аккуратно решать задачу подобного типа с третьем параметром (за исключением случая, когда третий параметр – координата z).

Дело в том, что функция $DBSCAN()$, содержащая встроенную математическую функцию $dist()$ может работать некорректно. Это происходит из-за того, что функция $dist()$ из библиотеки $math()$ работает и в трехмерном пространстве. Поэтому при рассчете евклидового расстояния значение третьего параметра (в нашем случае – блеска звезды) будет учитываться как координата звезды.

Для того чтобы избежать подобной ошибки используйте срезы:

$$dist(i[:2], j[:2])$$

2.4 Задачи, содержащие двойную/тройную звездную систему

Примечание: нельзя исключать вариант добавления задачи, в которой необходимо будет произвести кластеризацию звезд дважды. То есть разделить начальный набор точек на N кластеров, а затем в каждом из этих кластеров выделить кластеры, которые содержат две/три звезды, находящиеся на на определенном расстоянии.

Методистами Школково были сделаны задачи подобного типа. В них сразу даны расстояния, по которым следует проводить кластеризацию. Поэтому такой тип задач оптимальнее всего решать только через метод **DBSCAN**.

Учёный решил провести кластеризацию некоторого множества звёзд по их расположению на карте звёздного неба. Кластер звёзд – это набор звёзд (точек) на графике, каждая из которых находится от хотя бы одной другой звезды на расстоянии не более R условных единиц. Каждая звезда обязательно принадлежит только одному из кластеров.

Двойная звездная система – это две звезды на расстоянии менее t. При этом других звезд на расстоянии менее t у этих двух звезд быть не должно.

Под расстоянием понимается расстояние Евклида между двумя точками $A(x_1, y_1)$ и $B(x_2, y_2)$ на плоскости, которое вычисляется по формуле:

$$d(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Аномалиями назовём точки, находящиеся на расстоянии более одной условной единицы от точек кластеров. При расчётах аномалии учитывать не нужно.

В файле A хранятся данные о звёздах **двух** кластеров, где $R = 0.5$, $t = 0.05$ для каждого кластера. В каждой строке записана информация о расположении на карте одной звезды, а также ее масса (в солнечных массах): сначала координата x, затем координата y, затем масса m. Значения даны в условных единицах, которые представлены вещественными числами. Известно, что количество звёзд не превышает 3000.

Для файла в каждом кластере найдите двойную звездную систему, состоящую из красного гиганта (масса от 5.3 до 10 солнечных масс) и желтого гиганта (масса от 2.5 до 5.0 солнечных масс) с максимальным расстоянием между ними. Затем

вычислите числа: P_x — среднее арифметическое абсцисс найденных звезд, и P_y – среднее арифметическое ординат найденных звезд.

Программное решение:

В данном типе задачи не обязательно строить диаграмму в Excel, так как все расстояния, на которых расположены звезды в каждом из кластеров, даны.

Реализуем функцию **DBSCAN()**:

```
from math import *
def dbscan(a, r):
    cl = [] # инициализируем список для хранения кластеров
    while a: # пока есть элементы в входном массиве 'a'
        cl.append([a.pop(0)]) # новый кластер с первым элементов из 'a'
        for i in cl[-1]: # проходим по элементам последнего кластера
            # проверяем каждый элемент 'j' в оставшихся элементах 'a'
            for j in a[:]:
                # если расстояние между 'i' и 'j' меньше радиуса 'r',
                x = [i[0], i[1]], y = [j[0], j[1]]
                if dist(x, y) < r:
                    cl[-1].append(j) # добавляем 'j' в текущий кластер
                    a.remove(j) # удаляем 'j' из списка 'a'
    return cl
```

На примере файла A для начала разделим все звезды на кластеры, находящиеся на расстоянии $R = 0.5$:

```
f = open("27.txt"), s = f.readline()
a = [list(map(float, i.replace(",", ".").split())) for i in f]
cl = dbscan(a, 0.5) # проводим кластеризацию
cl_total = [] # список с кластерами, в которых больше 10 звезд
for i in cl: if len(i) > 10: cl_total.append(i)
```

После кластеризации основного файла находим двойные системы в каждом кластере:

```
t = 0.05 # для файла A
ans = []
for i in cl_total: # проходим по каждому элементу в списке cl_total
    found_star = dbscan(i, t) # применяем алгоритм DBSCAN
    bin_stars = [] # список для бинарных звездных систем
    # список для хранения звездной системы с максимальным расстоянием
    mx_starsys = []
    # проходим по каждому кластеру, найденному алгоритмом DBSCAN
    for j in found_star:
        if len(j) == 2: # проверяем, состоит ли кластер из двух звезд
            # проверяем состоит ли кластер из белого и голубого карликов
            if ((5.3 <= j[0][2] <= 10 and 2.5 <= j[1][2] <= 5) or (
                    5.3 <= j[1][2] <= 10 and 2.5 <= j[0][2] <= 5)):
                bin_stars.append(j)
    mx_dist = 0 # переменная для хранения максимального расстояния
    for j in bin_stars: # проходим по всем найденным бинарным системам
        x = [j[0][0], j[0][1]]
        y = [j[1][0], j[1][1]]
        # вычисляем расстояние между звездами в бинарной системе
        if dist(x, y) > mx_dist:
            mx_dist = dist(x, y) # обновляем максимальное расстояние
            mx_starsys = j
    ans.append(mx_starsys)
# рассчитываем среднее значение
res_X, res_Y = 0, 0
for i in ans:
    res_X += (i[0][0] + i[1][0])
    res_Y += (i[0][1] + i[1][1])
print(int(res_X / 4 * 1000), int(res_Y / 4 * 1000))
```

Важно!!!

При решении задач на тройные звездые системы в условии может быть сказано: «Тройная звездная система – это система, в которой три звезды попарно находятся на расстоянии не более t.»

Здесь важно учитывать, что при второй кластеризации функция **DBSCAN()** не учтет, что звезды находятся на расстоянии не более t именно **попарно**. В таком случае важно проводить дополнительную проверку с помощью функции **dist()**.

Приведем пример: есть три звезды (a, b, c). Если рассматривать их одновременно, то расстояние между тремя звездами сразу будет меньше заданного t. Однако одно из расстояний между (a, b), (a, c) или (b, c) может превышать t и тогда такая звездная система не подойдет.

3 Заключение

Прорезюмируем, что Вы теперь знаете и умеете:

- Знаете, что такое кластеризация в рамках 27 задания
- Знаете как решить задачу методом координат
- Знаете как решить задачу разделением прямых
- Знаете как решить задачу через уравнение окружности
- Знаете алгоритм DBSCAN и умеете применять его в решении задач

Удачи на экзамене! Ваша команда Школково по Информатике.

Подробнее изучить работу с базами данных вы можете на вебинарах ***«Школково»***

А также подписавшись на наши социальные сети: