DOI: 10.32517/2221-1993-2025-24-1-69-76

К. Ю. Поляков
Санкт-Петербургский государственный морской технический университет, Санкт-Петербург, Россия;
Школа № 174, Санкт-Петербург, Россия

КЛАСТЕРИЗАЦИЯ В ЗАДАЧАХ ЕГЭ ПО ИНФОРМАТИКЕ

Аннотация

В статье рассмотрены методические подходы к решению задачи на анализ данных, включенной в единый государственный экзамен (ЕГЭ) по информатике с 2025 года. Показано, что при любой форме кластеров возможно разделить их набором прямых, параллельных осям координат, или наклонными прямыми. Предлагается ускоренный алгоритм поиска центров кластеров, позволяющий значительно уменьшить время вычислений и таким образом решить задачу для наборов данных, содержащих более 10 000 точек, за приемлемое время. Рассмотрены усложненные, в сравнении с демонстрационным вариантом контрольно-измерительных материалов, варианты задачи: наличие аномалий, которые нужно игнорировать при поиске центра кластера, и кластеры сложной формы. Проведено сравнение различных подходов к разделению данных на кластеры, включая алгоритм DBSCAN и метод *k*-средних.

Ключевые слова: информатика, ЕГЭ, КИМ, анализ данных, кластеризация, аномалии, алгоритм DBSCAN, метод *k*-средних.

Информация об авторе

Поляков Константин Юрьевич, доктор тех. наук, профессор кафедры судовой автоматики и измерений, факультет корабельной энергетики и автоматики, Санкт-Петербургский государственный морской технический университет, Санкт-Петербург, Россия; учитель информатики, школа № 174, Санкт-Петербург, Россия; *e-mail:* kpolyakov@mail.ru

1. Введение

В демонстрационный вариант контрольно-измерительных материалов (КИМ) ЕГЭ по информатике 2025 года впервые включена задача на анализ данных [3]. В этой задаче требуется выполнить кластеризацию данных, заданных координатами на плоскости, и найти центры кластеров. В настоящей статье рассматриваются возможные подходы к решению этой задачи в условиях ЕГЭ и формулируются рекомендации, которые могут быть полезны при подготовке школьников к экзамену. Исследуется не только простейший вариант задачи, приведенный в [3], но и ее возможные вариации: наличие аномалий и сложные формы кластеров.

Задачи кластеризации являются одними из наиболее популярных задач анализа данных — см., например, [1, 8–10]. Для практических вычислений можно использовать как языки программирования [9], так и табличные процессоры [6].

Особенность ситуации на ЕГЭ состоит в том, что специализированные пакеты для решения задач анализа данных и машинного обучения, такие как scikit-learn*, недоступны. Поэтому можно использовать только возможности стандартной библиотеки языка программирования.

* Пакет scikit-learn можно загрузить с сайта: https://scikit-learn.org

* Материалы к статье можно скачать на сайте ИНФО: http://infojournal.ru/journals/school/school_01-2025/

K. Yu. Polyakov
State Marine Technical University, Saint Petersburg, Russia;
School 174, Saint Petersburg, Russia

CLUSTERING IN TASKS OF THE UNIFIED STATE EXAM IN INFORMATICS

Abstract

The article presents methodological approaches to solving the data analysis problem included in the Unified State Exam (USE) in informatics since 2025. It is shown that for any shape of clusters, it is possible to separate them by a set of straight lines parallel to the coordinate axes or by inclined lines. An accelerated algorithm for finding cluster centers is proposed, which allows to significantly reduce the computation time and thus to solve the problem for data sets containing more than 10,000 points in an acceptable time. Complex versions of the problem, as compared with the demonstration version of the control and measuring materials, are considered: the presence of anomalies that should be ignored when searching for the cluster center, and clusters of complex shape. A comparison of various approaches to dividing data into clusters, including the DBSCAN algorithm and the *k*-means method, is carried out.

Keywords: informatics, USE, test materials, data analysis, clustering, anomaly, DBSCAN algorithm, *k*-means method.

Information about the author

Konstantin Yu. Polyakov, Doctor of Sciences (Engineering), Professor at the Department of Ship Automation and Measurements, Faculty of Ship Power Engineering and Automation, State Marine Technical University, Saint Petersburg, Russia; informatics teacher, School 174, Saint Petersburg, Russia; *e-mail:* kpolyakov@mail.ru

ISSN 2221-1993 • **Informatics in School** • 2025 • Volume 24 No 1

Далее все фрагменты программ приведены на языке Python, хотя не составит труда переписать их на любом языке, который поддерживается на ЕГЭ.

2. Формулировка задачи

Рассмотрим сокращенную формулировку задачи из демонстрационного варианта КИМ 2025 года.

Задача 1 [3].

Фрагмент звездного неба спроецирован на плоскость с декартовой системой координат. Ученый решил провести кластеризацию полученных точек, являющихся изображениями звезд, т. е. разбить их множество на N непересекающихся непустых подмножеств (кластеров). Будем называть ***центром кластера*** точку этого кластера, сумма расстояний от которой до всех остальных точек кластера минимальна. Для каждого кластера гарантируется единственность его центра. Расстояние между двумя точками на плоскости $A(x_1, y_1)$ и $B(x_2, y_2)$ вычисляется по формуле:

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

В файле *input.txt* хранятся координаты точек трех кластеров. Известно, что количество точек не превышает 10 000. 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 два числа: сначала целую часть произведения $P_x \times 10\,000$, затем целую часть произведения $P_y \times 10\,000$.

Возможные данные проиллюстрированы графиком на рисунке 1.

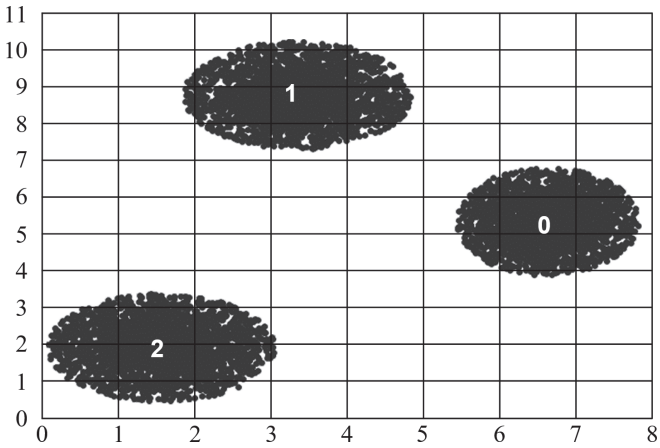


Рис. 1. График к задаче 1

Решение задачи включает два этапа:
1) разбиение данных на кластеры;
2) вычисление координат центров кластеров.
Разберем их последовательно.

3. Разбиение данных на кластеры

Сначала необходимо выполнить *визуализацию*, т. е. представить данные в виде, удобном для наблюдения. По условию задачи данные — это декартовы координаты точек на плоскости, поэтому можно просто отметить эти точки на графике каким-нибудь маркером, не соединяя их, как это сделано на рисунке 1. Для этой цели удобно использовать табличный процессор, например, LibreOffice Calc или Microsoft Excel.

Кластеры на рисунке 1 пронумерованы, начиная с нуля. В данном случае их легко отделить друг от друга прямыми, параллельными осям координат. По диаграмме видно, что все точки, для которых $x > 5$, относятся к кластеру 0. Оставшиеся точки разделяются на два кластера прямой $y = 6$: при $y > 6$ точка относится к кластеру 1, а в противном случае — к кластеру 2. Следующая функция принимает координаты точки и возвращает номер кластера, к которому она относится:

```
def clusterNo( x, y ):
  return 0 if x > 5 else \
         1 if y > 6 else \
         2
```

Удобно сразу распределить точки по кластерам при чтении данных. Для этого создаем список *clusters*:

```
K = 3       # количество кластеров
clusters = [ [] for i in range(K) ]
```

В случае трех кластеров (при $K = 3$) список *clusters* сначала содержит три пустых списка. В каждый из них мы будет добавлять координаты точек соответствующего кластера. Цикл, в котором данные читаются из файла и точки распределяются по кластерам, можно записать так:

```
for s in open("input.txt"):
  x, y = s.replace(',', '.').split()  # 1
  x, y = float(x), float(y)           # 2
  k = clusterNo( x, y )               # 3
  clusters[k].append( (x, y) )        # 4
```

Цикл for перебирает все строки файла *input.txt*, так что каждая из них на очередной итерации цикла оказывается в переменной *s*. В исходном файле целые части чисел отделяются от дробных запятой, а не точкой, как в языках программирования. Поэтому в строке 1 (см. нумерацию в комментариях) все запятые заменяются на точки с помощью метода *replace*. Строка сразу разбивается методом *split* на две части: в первой записана x-координата очередной точки, а во второй — ее y-координата. В строке 2 эти координаты с помощью функции *float* преобразуются из символьной записи в вещественные числа. Номер кластера k, к которому относится эта точка, определяем с помощью функции *clusterNo* (строка 3). Затем в строке 4 координаты новой точки, объединенные в кортеж, добавляются к списку *clusters*[k], который содержит точки кластера k.

4. Вычисление координат центра кластера

Заметим, что по условию задачи центр кластера — это *одна из точек кластера*, координаты которой присутствуют в файле, а не произвольная точка плоскости. Для определения центра можно перебрать все точки, принадлежащие кластеру, вычисляя для каждой из них сумму расстояний до остальных точек этого же кластера.

ISSN 2221-1993 • **Информатика в школе** • 2025 • Том 24 № 1

В качестве центра выбирается точка, для которой эта сумма расстояний минимальна.

Если в кластере N точек, то нам нужно для каждой из них вычислить $N-1$ расстояний до других точек. Поэтому сложность алгоритма — квадратичная, $O(N^2)$. Для 10 000 точек время работы программы должно составить несколько секунд, поэтому такой алгоритм вполне можно использовать на экзамене.

Для определения расстояния между двумя точками удобно написать отдельную функцию:

```
def dist( p1, p2 ):
  return ((p1[0]-p2[0])**2 +
          (p1[1]-p2[1])**2)**0.5
```

Параметры функции *p*1 и *p*2 — это пары координат (кортежи), которые описывают две точки. Первые элементы пар, *p*1[0] и *p*2[0], — это *x*-координаты точек, а вторые элементы, *p*1[1] и *p*2[1], — это их *y*-координаты. Функция возвращает квадратный корень из суммы квадратов разностей координат точек по каждой оси, т. е. евклидово расстояние между ними.

Для ускорения вычислений можно вместо нашей функции *dist* использовать одноименную функцию из модуля *math*. Благодаря эффективной реализации функции из стандартной библиотеки время поиска центров трех кластеров для данных из [3] сокращается в два-три раза.

Удобно оформить алгоритм определения центра кластера как отдельную функцию *getCenter*, которая принимает один параметр — список точек кластера:

```
def getCenter( cluster ):
  minSumDist = float("inf")
  for p0 in cluster:
    sumDist = sum( dist(p0,p)
                   for p in cluster )
    if sumDist < minSumDist:
      minSumDist = sumDist
      center = p0
  return center
```

Сумму расстояний от очередной точки *p*0 до остальных точек кластера вычисляем с помощью функции *sum* и затем сравниваем эту сумму с минимальной, которая записана в переменной *minSumDist*. Если найдена новая минимальная сумма, запоминаем точку *p*0 как новый центр кластера в переменной *center*.

После распределения данных по кластерам координаты всех точек кластера *k* находятся в списке *clusters*[*k*]. Используя функцию *getCenter*, центры всех кластеров можно найти с помощью генератора списка:

```
centers = [ getCenter(cluster)
            for cluster in clusters ]
```

Теперь *centers*[*k*] — это кортеж, содержащий координаты центра кластера с номером *k*. Первый элемент кортежа, *centers*[*k*][0], — это *x*-координата центра, а *centers*[*k*][1] — его *y*-координата. Остается вычислить среднее арифметическое *x*-координат и *y*-координат центров кластеров и вывести результат:

```
Px = sum( centers[k][0] for k in range(K) ) / K
Py = sum( centers[k][1] for k in range(K) ) / K
print( int(Px*10000), int(Py*10000) )
```

5. Ускорение вычислений

Попробуем ускорить вычисление координат центра кластера. Очевидно, что центр находится где-то в середине кластера, а не у его края. Определим координаты ***средней точки (центроида)*** кластера как средние арифметические координат по каждой оси:

```
N = len(cluster)
cx = sum( p[0] for p in cluster ) / N
cy = sum( p[1] for p in cluster ) / N
pMid = (cx, cy)
```

Теперь в переменной *cx* записана *x*-координата средней точки, а в переменной *cy* — ее *y*-координата; эти координаты объединены в кортеж *pMid*.

Естественно предположить, что центр кластера, который мы ищем, расположен вблизи средней точки. Чтобы сократить перебор, будем рассматривать только те точки, которые находятся на расстоянии не более некоторого δ от средней точки *pMid*. Из этих точек составим список *pointsNearCenter*:

```
delta = 0.2
pointsNearCenter = [p for p in cluster
                    if dist(pMid,p) < delta ]
```

и будем выбирать центр только среди точек из этого списка. Их будет намного меньше, чем всех точек кластера, так что алгоритм закончит работу быстрее.

Приведем ускоренный вариант функции *getCenter*:

```
def getCenter( cluster ):
  N = len(cluster)
  pMid = ( sum( p[0] for p in cluster ) / N,
           sum( p[1] for p in cluster ) / N )
  delta = 0.2
  pointsNearCenter = [p for p in cluster
                if dist(pMid,p) < delta ]
  minSumDist = float("inf")
  for p0 in pointsNearCenter:
    sumDist = sum( dist(p0,p)
                   for p in cluster )
    if sumDist < minSumDist:
      minSumDist = sumDist
      center = p0
  return center
```

Для набора данных из [3] при δ = 0,2 улучшенная версия алгоритма работает за счет сокращения перебора примерно в 6 раз быстрее, чем первоначальная.

Значение δ выбирается с запасом в зависимости от размеров кластера и плотности точек в нем. Как правило, можно выбирать δ равным 10–15 % от радиуса кластера; для кластеров сложной формы иногда приходится использовать бóльшие значения δ. Если выбрано слишком малое δ, то точка, которая является центром кластера, может не попасть в δ-окрестность средней точки, и тогда получится неверный ответ. Чем больше δ, тем медленнее работает алгоритм, но меньше вероятность ошибки. Радиусы кластеров для данных из [3] примерно равны 1,5 единицы, при этом правильное решение получается для любого $\delta \geq 0{,}02$.

ISSN 2221-1993 • **Informatics in School** • 2025 • Volume 24 No 1

6. Аномалии

Аномалии — это значения в наборе данных, которые сильно отличаются от всех остальных. В задачах кластеризации, которые могут быть предложены на ЕГЭ, аномалии — это точки, не принадлежащие ни одному из кластеров. Перед обработкой данных аномалии нужно удалить — это часть процедуры, которая называется ***очисткой данных***. Для этого необходимо точно определить, что в данной задаче мы считаем аномалиями.

Задача 2.

[Текст, совпадающий с условием задачи 1].

Аномалиями назовем группы не более чем из 10 точек, находящиеся на расстоянии более одной условной единицы от точек всех кластеров. При расчетах аномалии учитывать не нужно.

Возможные данные проиллюстрированы графиком на рисунке 2. Аномалии выделены штриховыми линиями.

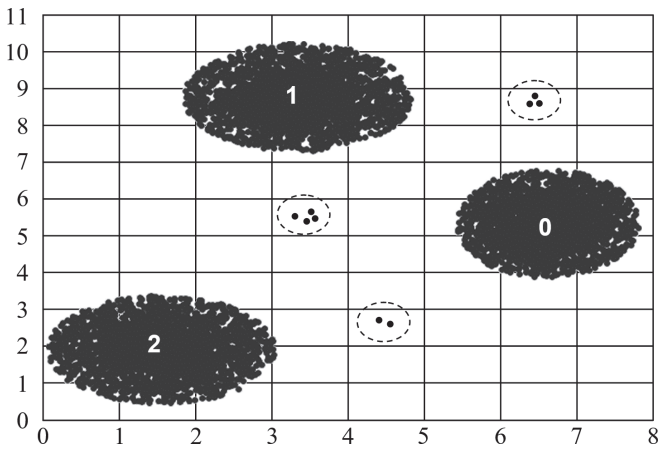


Рис. 2. Иллюстрация данных задачи 2

В этой задаче нам не удастся так просто разделить точки на кластеры, как мы это сделали при решении задачи 1. Дело в том, что условию $x > 5$, например, соответствуют не только точки кластера 0, но и точки аномалии в полосе $6 < x < 7$. Поэтому для выделения кластера 0 необходимо применить сложное условие $x > 5$ *and* $y < 7$. Точки кластера 1 определяются условием $x < 5$ *and* $y > 6$, а точки кластера 2 — условием $x < 4$ *and* $y < 4$. Каждое из трех условий задает ***квадрант*** — четверть плоскости, ограниченную прямым углом, причем ни в одну из этих областей аномалии не попадают.

Таким образом, с помощью условий, которые выделяют только интересующие нас кластеры, мы исключили все аномалии, так что дальнейший алгоритм решения задачи изменять не нужно.

Изменим функцию *clusterNo* так, чтобы она возвращала значение –1 для точек аномалий, не принадлежащих ни одному из кластеров:

```
def clusterNo( x, y ):
  return 0 if x > 5 and y < 7 else \
         1 if x < 5 and y > 6 else \
         2 if x < 4 and y < 4 else \
        -1
```

Аномалии не должны добавляться в список *clusters*, поэтому нужна дополнительная проверка в цикле ввода данных (эта строка программы выделена фоном):

```
for s in open("input.txt"):
  x, y = s.replace(',', '.').split()
  x, y = float(x), float(y)
  k = clusterNo( x, y )
  if k >= 0:
    clusters[k].append( (x, y) )
```

Заметим, что у нас получилась универсальная программа: при изменении количества и расположения кластеров и аномалий требуется изменить только значение переменной *k* в начале программы и функцию *clusterNo*, которая определяет номер кластера по координатам точки.

7. Наклонные разделительные прямые

Рассмотрим три кластера на рисунке 3. Ограничить каждый кластер квадрантом здесь не получится, однако можно отделить кластеры друг от друга наклонными прямыми I и II, которые показаны на рисунке штриховыми линиями.

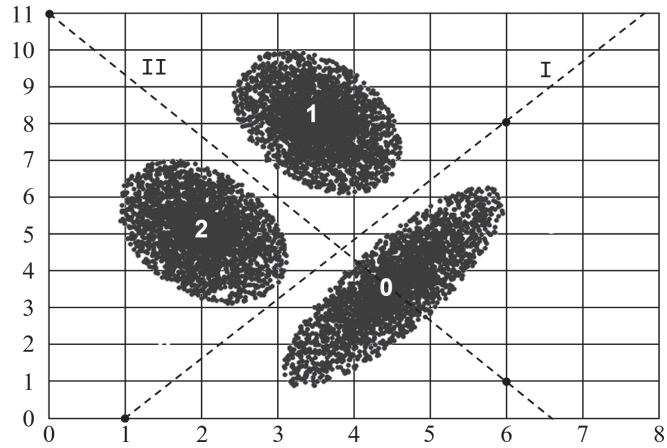


Рис. 3. Отделение кластеров друг от друга наклонными прямыми

Уравнения этих прямых можно построить аналитически, используя сведения из курса школьной математики. В данном случае получаем следующие уравнения:

прямая I: $y = \frac{8}{5}(x-1)$;

прямая II: $y = -\frac{5}{3}x + 11$.

Для проверки можно построить эти прямые на диаграмме, добавив к данным из файла дополнительные столбцы и расширив на них область данных диаграммы. Например, на рисунке 4 на диаграмму добавлена прямая I.

В ячейку C1 (рис. 4) записана формула: =$D$2*A1+$E$2, которая потом была скопирована на весь столбец C. Изменяя значения ячеек D2 и E2, можно подбирать коэффициенты *k* и *b* экспериментально.

Используя полученные выше уравнения разделяющих прямых, запишем функцию *clusterNo* для рассмотренного случая:

ISSN 2221-1993 • **Информатика в школе** • 2025 • Том 24 № 1

	A	B	C	D	E	F	G	H	I	J	K	L
1	3,713	2,942	4,341	*k*	*b*							
2	4,798	3,602	6,076	1,6	-1,6							
3	4,896	3,239	6,233									
4	3,773	6,559	4,437									
5	3,274	9,672	3,639									
6	4,778	3,363	6,045									
7	3,664	7,875	4,262									
8	2,792	9,586	2,868									
9	3,189	6,555	3,503									
10	3,748	9,587	4,397									
11	1,980	5,153	1,569									
12	4,283	4,679	5,253									
13	4,469	3,521	5,550									
14	2,419	4,707	2,270									
15	3,498	8,008	3,996									
16	3,577	2,028	4,123									
17	2,691	4,428	2,706									

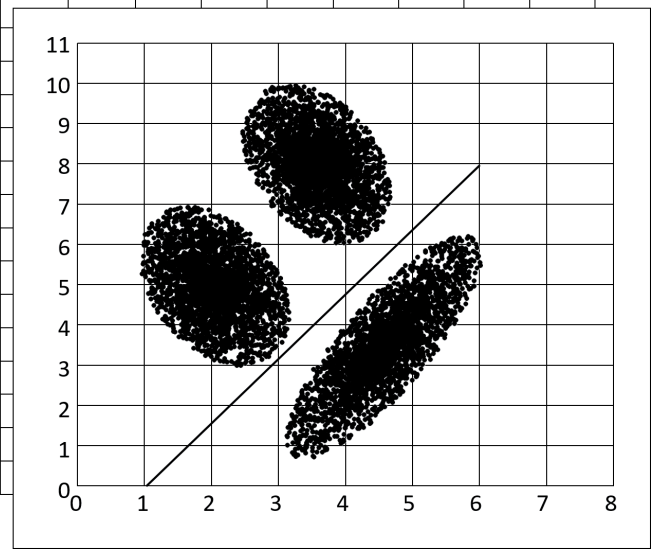


Рис. 4. Добавление наклонной прямой на диаграмму

```
def clusterNo( x, y ):
  return 0 if y < (8/5)*(x-1) else \
         1 if y > -5/3*x+11 else \
         2
```

Если в данных есть аномалии, их можно учесть так же, как мы делали ранее. Отметим, что остальная часть программы не изменяется.

8. Метод покрытия прямоугольниками

Рассмотрим диаграмму на рисунке 5.

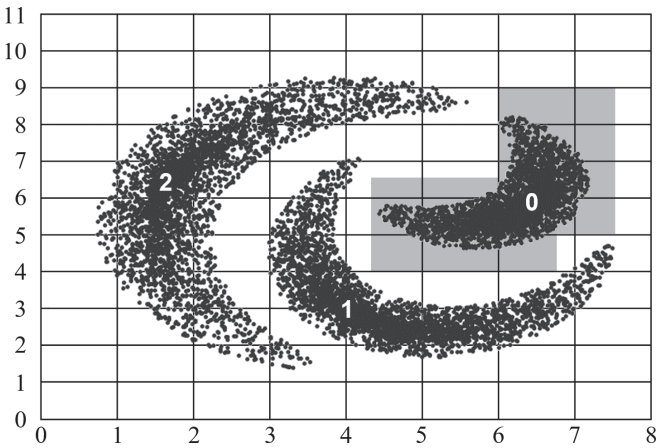


Рис. 5. Пример усложненной диаграммы. Покрытие прямоугольниками кластера 0

Чтобы разделить кластеры в этой задаче, можно провести между ними какие-нибудь сложные границы или собирать кластеры из частей.

Покажем, как можно записать условие, определяющее кластер 0 на рисунке 5. Заметим, что этот кластер полностью перекрывается двумя прямоугольниками, которые выделены фоном на рисунке 5. Поэтому условие принадлежности точки с координатами (x, y) кластеру 0 можно записать в виде:

$$(4{,}3 < x < 6{,}8 \textit{ and } 4 < y < 6{,}5) \textit{ or } (6 < x < 7{,}5 \textit{ and } 5 < y < 9).$$

В данном случае правую и верхнюю границы второго прямоугольника можно вообще не указывать, потому что правее и выше этого прямоугольника никаких «лишних» точек нет:

$$(4{,}3 < x < 6{,}8 \textit{ and } 4 < y < 6{,}5) \textit{ or } (x > 6 \textit{ and } y > 5).$$

На рисунке 6 показано покрытие прямоугольниками кластера 1 (кластер 0 удален с диаграммы, поскольку мы его уже обработали).

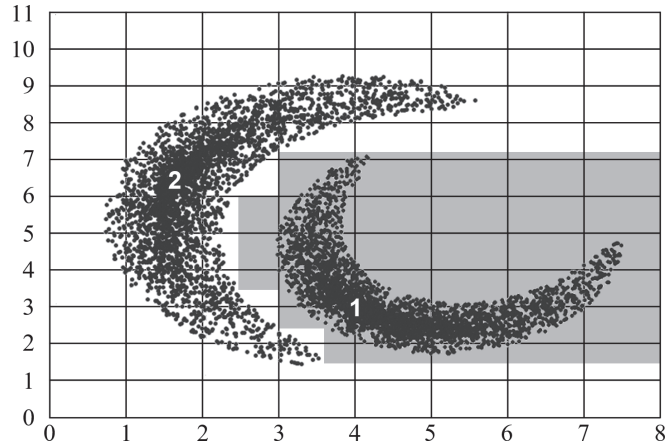


Рис. 6. Покрытие прямоугольниками кластера 1

Чтобы выделить кластер 1, нужно использовать три полосы:

$$(x > 3 \textit{ and } 2{,}4 < y < 7{,}2)$$

$$\textit{or } (x > 2{,}5 \textit{ and } 3{,}5 < y < 6)$$

$$\textit{or } (x > 3{,}6 \textit{ and } 1{,}5 < y < 3)$$

ISSN 2221-1993 • **Informatics in School** • 2025 • Volume 24 No 1

Аномалий в этой задаче нет, поэтому все точки, которые не попали в кластеры 0 и 1, относятся к кластеру 2. Функция *clusterNo* запишется следующим образом:

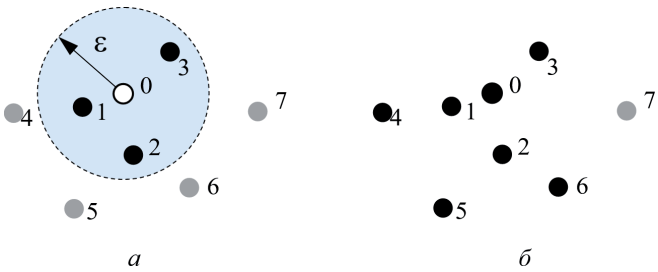
```
def clusterNo( x, y ):
  return 0 if \
            (4.3 < x < 6.8 and 4 < y < 6.5) \
             or (x > 6 and y > 5) else \
          1 if (x > 3 and 2.4 < y < 7.2) or \
               (x > 2.5 and 3.5 < y < 6) or \
               (x > 3.6 and 1.5 < y < 3) else \
          2
```

Этот простой метод применим для кластеров любой формы. Поэтому в задачах ЕГЭ нет необходимости использовать разделительные линии, которые задаются сложными уравнениями.

9. Алгоритм DBSCAN

Во многих случаях возможно автоматическое разбиение точек на кластеры, выполняемое программой без участия человека. В случае, когда точки кластера расположены достаточно плотно, а именно это характерно для задач ЕГЭ, хорошо работает алгоритм DBSCAN (*англ.* Density-based spatial clustering of applications with noise) [8, 10]. Его основная идея состоит в том, что точки, расстояние между которыми меньше некоторого ε, считаются соседями, т. е. относятся к одному и тому же кластеру*.

Мы начинаем строить кластер, взяв произвольную точку из набора данных. Пусть это будет точка 0, обозначенная белым цветом на рисунке 7, *а*. Среди оставшихся точек выбираем и добавляем в тот же кластер всех ее соседей, т. е. все точки ее ε-окрестности, расстояние от которых до точки 0 меньше ε. Эти точки (с номерами 1, 2 и 3) обозначены на рисунке 7, *а* черным цветом, а все еще не выбранные точки — серым.



а *б*

Рис. 7. Построение кластера

Затем в кластер добавляются все точки в ε-окрестностях вновь добавленных точек. В ε-окрестности точки 1 находим одного нового соседа — точку 4. В ε-окрестность точки 2 попадают два новых соседа: точки 5 и 6. Просмотр ε-окрестностей точек 3, 4, 5 и 6 не дает результатов: оставшаяся точка 7 находится от любой точки кластера на расстоянии, большем, чем ε. Окончательный результат показан на рисунке 7, *б*: в кластер включены точки 0, 1, 2, 3, 4, 5 и 6, а точка 7 этому кластеру не принадлежит.

Алгоритм DBSCAN позволяет автоматически разбить все точки на кластеры и выявить аномалии. Например, если обнаружен кластер, в котором менее 10 точек, он может считаться аномалией. В отличие от многих алгоритмов кластеризации, количество кластеров задавать заранее не нужно.

Рассмотренный вариант алгоритма DBSCAN имеет единственный параметр — величину ε. Для каждого конкретного набора данных подходящее значение ε определяется визуально по диаграмме.

Напишем программу, которая реализует алгоритм DBSCAN. Координаты всех точек читаем из файла и записываем в список *data* как кортежи, состоящие из двух координат:

```
data = []
for s in open("input.txt"):
  x, y = s.replace(',', '.').split()
  data.append( (float(x), float(y)) )
```

Основную идею алгоритма реализует функция *addNeighbors*, которая добавляет к кластеру *cluster* всех соседей заданной точки *p*0:

```
def addNeighbors( cluster, p0, eps ):
  neighbors = [ p for p in data
                  if dist(p, p0) < eps ]
  cluster += neighbors
  for p in neighbors:
    data.remove( p )
  for p in neighbors:
    addNeighbors( cluster, p, eps )
```

Функция принимает три параметра: *cluster* — список точек кластера, который мы хотим расширить; *p*0 — кортеж, содержащий координаты точки, соседей которой мы добавляем; *eps* — значение параметра ε. Функция ничего не возвращает (в ней нет оператора *return*), в результате ее работы изменяется список *cluster*, переданный ей как параметр.

Сначала генератор списка находит все точки, находящиеся от точки *p*0 на расстоянии, меньшем, чем ε, и строит из них список *neighbors*. Затем все выбранные точки удаляются из глобального списка *data*. Второй цикл в теле функции добавляет в кластер все точки, соседние с уже выбранными, т. е. «соседей соседей». Рекурсия закончится тогда, когда для очередной точки не удастся найти новых (еще не выбранных) соседей в ее ε-окрестности.

Теперь можно написать функцию *getCluster*, которая собирает в список все точки кластера, к которому принадлежит заданная точка *p*0:

```
def getCluster( p0, eps ):
  cluster = [ p0 ]
  data.remove( p0 )
  addNeighbors( cluster, p0, eps )
  return cluster
```

Функция создает новый кластер, содержащий сначала только одну точку *p*0, и удаляет эту точку из списка *data*. Затем в кластер добавляются все соседи начальной точки, соседи соседей и т. д. Результат работы функции — список точек кластера, содержащего точку *p*0.

Если *p*0 — это изолированная точка, в ε-окрестности которой нет ни одного соседа, то функция вернет список, состоящий из одной этой точки. Если аномалия включа-

* Далее описывается упрощенный вариант алгоритма DBSCAN, применимый для решения задач ЕГЭ.

ISSN 2221-1993 • **Информатика в школе** • 2025 • Том 24 № 1

ет несколько точек рядом с точкой *p*0, только эта группа и будет выделена в новый кластер.

Разбиение на кластеры в основной программе выполняется с помощью цикла, который работает до тех пор, пока в списке *data* не останется невыбранных точек.

```
clusters = []
while data:
  cluster = getCluster( data[0], eps )
  if len(cluster) > 10:
    clusters.append( cluster )
```

Проверка условия, выделенная фоном, учитывает аномалии, состоящие не более чем из 10 точек. Аномалии не добавляются в список *clusters*.

Другие варианты реализации алгоритма DBSCAN при решении задач ЕГЭ, в том числе нерекурсивные, можно найти в [2, 4, 5, 7].

Для того чтобы убедиться, что алгоритм правильно разделил данные на кластеры, полезно построить картинку на экране так, чтобы точки одного кластера имели одинаковый цвет, а точки разных кластеров были раскрашены по-разному. Для этой цели в [2] было предложено использовать встроенный модуль «черепашьей графики» *turtle*. Отметим, что из-за особенностей графики модуля *turtle* картинка для большого количества точек строится довольно долго, особенно на слабых компьютерах.

Несомненное достоинство метода DBSCAN состоит в том, что он позволяет разделить точки на кластеры полностью автоматически. При этом количество кластеров не нужно знать заранее, необходимо только выбрать подходящее значение параметра ε. Если значение ε слишком велико, программа может ошибочно объединить несколько кластеров в один. Это значит, что в данных оказались точки, принадлежащие разным кластерам, но отстоящие друг от друга на расстояние, меньшее, чем ε. Если задать слишком маленькое значение ε, программа может воспринять один кластер как несколько отдельных кластеров. Обычно правильное значение ε несложно найти визуально по построенной диаграмме.

Основной недостаток алгоритма DBSCAN связан с тем, что он не позволяет автоматически разделить кластеры, точки которых находятся близко друг к другу (например, когда кластеры треугольной формы почти касаются вершинами).

10. Метод *k*-средних

Метод *k*-средних (*англ.* *k*-means) позволяет не только разделить данные на кластеры в автоматическом режиме, но и сразу найти их центры. Для этого требуется заранее задать количество кластеров (в задачах ЕГЭ оно известно из условия).

В классическом варианте методом *k*-средних определяют ***центроиды*** кластеров, которые могут располагаться в произвольных точках плоскости. В задачах ЕГЭ центр кластера, который нужно найти, должен обязательно совпадать *с одной из точек этого кластера*. Такая точка называется ***медоидом*** кластера, а соответствующий вариант решения задачи — методом *k*-медоидов (*англ.* *k*-medoids). Следуя формулировке задания 27 из [3], мы будем далее называть медоид центром кластера.

Пусть нужно разделить данные на *k* кластеров и для каждого кластера найти его центр. Эта задача строго решается только полным перебором вариантов. Однако есть несколько эвристических методов, которые позволяют быстро найти хорошее решение, но не гарантируют, что оно будет оптимальным. Один из алгоритмов выглядит так:

Шаг 1. Выбрать *k* точек — временные центры кластеров.

Шаг 2. Каждую точку данных связать с ближайшим центром; таким образом выполняется разделение на кластеры.

Шаг 3. Для каждого кластера найти новый центр — точку кластера, суммарное расстояние от которой до всех остальных точек этого кластера минимально.

Шаг 4. Если центры кластеров поменялись, перейти к шагу 2, иначе — завершить работу алгоритма.

К сожалению, этот алгоритм нельзя использовать «вслепую», без анализа конкретных данных. Он не всегда находит самое лучшее решение, результат его работы зависит от выбора начальных координат центров на шаге 1. Как их лучше выбирать — точно не известно. Обычно используют один из двух вариантов: случайный выбор или назначение вручную по результатам анализа диаграммы. Например, для данных, изображенных на рисунке 1, начальные центры кластеров 0, 1 и 2 можно выбрать в точках (6,5; 5), (3,5; 9) и (1,5; 2) соответственно. Неважно, что точки с такими координатами, вероятнее всего, не входят в набор данных. Это не влияет на окончательный результат работы алгоритма, так как уже на первой итерации после шага 3 центры будут размещены в точках, принадлежащих кластерам.

Так же, как и в программе для метода DBSCAN, загружаем данные из файла в список *data*. Затем задаем количество кластеров и примерные координаты их центров:

```
K = 3 # количество кластеров
centers = [(6.5, 5), (3.5, 9), (1.5, 2)]
```

Список *oldCenters*, где будем сохранять предыдущие координаты центров, оставляем пустым:

```
oldCenters = []
```

Основной цикл выполняется до тех пор, пока координаты центров кластеров не перестанут изменяться, т. е. пока списки *centers* и *oldCenters* не совпадут:

```
while centers != oldCenters:
  oldCenters = centers
  clusters = [ [] for i in range(K) ]
  for p in data:
    distToCenters = [ dist(p,center)
                      for center in centers ]
    k = distToCenters.index \
                    ( min(distToCenters) )
    clusters[k].append( p )
  centers = [ getCenter(cluster)
              for cluster in clusters ]
```

Список *clusters* сначала содержит *k* пустых списков, в которые будут добавляться точки. Вложенный цикл *for*

перебирает все точки из списка *data*, определяя, к какому центру ближе всего каждая из них, т. е. к какому кластеру она относится. Для этого в списке *distToCenters* вычисляются расстояния от выбранной точки до всех центров. Затем определяется индекс минимального элемента в этом списке — это и будет номер нужного кластера *k*, в который добавляется эта точка. После разделения на кластеры центр каждого кластера пересчитывается с помощью функции *getCenter*.

Увы, алгоритм *k*-медоидов может приводить к неверному ответу. Рассмотрим данные, представленные на рисунке 8.

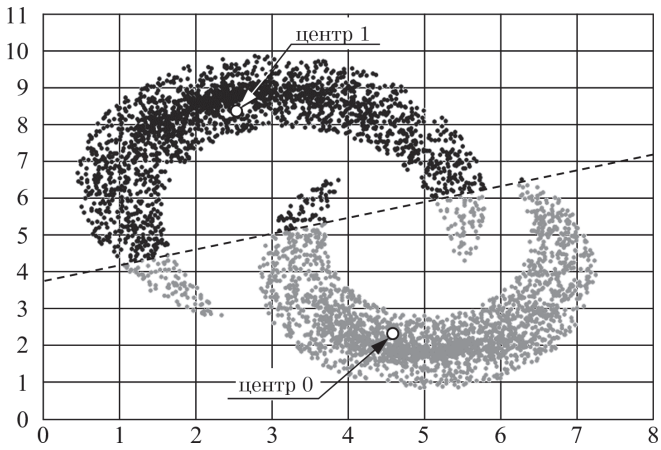


Рис. 8. Пример данных для иллюстрации неверного результата при использовании алгоритма k-медоидов

Приведенный выше алгоритм определяет центры кластеров как точки с координатами* (2,49; 8,29) и (4,93; 2,30). К сожалению, это ошибочный результат. Проблему легко увидеть, если построить диаграмму и раскрасить точки разных кластеров разными цветами. Штриховая линия на рисунке 8 показывает границу между кластерами — это точки, равноотстоящие от обоих центров, найденных с помощью нашего алгоритма. Понятно, что такое деление неверно: часть каждого кластера отнесена к другому кластеру. Границы между кластерами образуют ***диаграмму Вороного*** — разбиение плоскости, при котором точки в каждой области расположены ближе к одному из центров, чем к любому другому центру.

Итак, метод *k*-средних (и его вариант — метод *k*-медоидов) во многих случаях позволяет разбить данные на кластеры в автоматическом режиме. Он отлично работает для кластеров, форма которых похожа на круг с нормальным (гауссовым) распределением точек внутри круга. Если данные образуют кластеры сложной формы, как, например, на рисунке 8, этот метод дает неправильный результат.

Второй недостаток метода — он не учитывает аномалии. Если в задаче с аномалиями все же хочется использовать именно этот метод, можно считать каждую группу аномалий отдельным кластером и при вычислении ответов на вопрос задачи не учитывать эти «аномальные» кластеры. К сожалению, такой прием тоже не всегда приводит к правильному ответу.

* Здесь и далее все числовые значения округлены до двух знаков после запятой.

11. Выводы

В настоящей статье рассмотрена методика решения задач на кластеризацию, которые могут предлагаться на ЕГЭ по информатике. Рассмотрены три простых метода кластеризации данных:

1) ручное разделение точек на кластеры с помощью разделительных линий или покрытия кластеров прямоугольниками;
2) алгоритм DBSCAN;
3) метод *k*-средних.

Последние два метода позволяют выделить кластеры автоматически, но не всегда приводят к верному результату: алгоритм DBSCAN не может разделить кластеры, расположенные близко друг к другу; метод *k*-средних неверно выделяет кластеры сложной формы; при его использовании не всегда удается правильно обрабатывать аномалии.

В следующей таблице сравнивается время работы программы при решении задачи 27 из [3] разными методами. За единицу принято время при ручном выделении кластеров (на компьютере автора оно составило примерно одну секунду).

Метод решения	Время
Ручное разбиение на кластеры	1,0
DBSCAN	8,6
k-средних	2,2

Таким образом, *метод, использующий ручное разделение на кластеры, работает быстрее всего и надежнее всего*. Именно его рекомендуется использовать при решении задачи на кластеризацию на экзамене.

Список источников

1. *Баюк Д. А., Березин Д. В., Иванюк В. А.* Практическое применение методов кластеризации, классификации и аппроксимации. М.: Прометей, 2020. 448 с.

2. *Богданов А. А.* Солвер всех прототипов новых 27х на кластеризацию. https://vkvideo.ru/video-142628541_456239405

3. Демонстрационный вариант контрольных измерительных материалов единого государственного экзамена 2025 года по информатике // ФИПИ. https://doc.fipi.ru/ege/demoversii-specifikacii-kodifikatory/2025/inf_11_2025.zip

4. *Кабанов А. М.* Задание 27 | КЕГЭ по информатике 2025. https://vkvideo.ru/video-205865487_456240432

5. *Лашин В. А.* Самое простое решение 27 задания — кластеризация слиянием. https://rutube.ru/video/82c13ef41e32b3eabc844fd6d40bb3bd/

6. *Хун Чжоу.* Машинное обучение сквозь призму Excel: примеры и упражнения / пер. с англ. А. Ю. Гинько. М.: ДМК Пресс, 2025. 270 с.

7. *Шастин Л. Д.* Задание 27 — Метод DBSCAN. https://rutube.ru/video/5476cc0a61b984cb9b69d223cfca17ec/

8. *Scitovski R., Sabo K., Martínez-Álvarez F., Ungar Š.* Cluster Analysis and Applications. Cham: Springer International Publishing, 2021. 271 p. DOI: 10.1007/978-3-030-74552-3.

9. *Vasques X.* Machine learning theory and applications: hands-on use cases with Python on classical and quantum machines. Hoboken, New Jersey: Wiley, 2024. 512 p.

10. *Wierzchoń S., Kłopotek M.* Modern Algorithms of cluster analysis. Cham: Springer International Publishing, 2018. 421 p. DOI: 10.1007/978-3-319-69308-8.