Федеральное агентство связи
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М. А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)

Факультет инфокоммуникационных сетей и систем
Кафедра программной инженерии и вычислительной техники

НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА

по дисциплине «Практика по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности»
на тему «Кластерный анализ»

Выполнил:
студент 3-го курса
дневного отделения
группы ИКПИ-81
Коваленко Леонид Александрович
Научный руководитель:
старший преподаватель
Петрова Ольга Борисовна

Санкт-Петербург
2020

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящем отчете о НИР применяют следующие термины с соответствующими определениями.

Множество X — совокупность объектов (элементов) x_i. Конечное множество: $X=\{x_1,\dots,x_n\}$. Бесконечное множество: $X=\{x_1,x_2,\dots\}$.

Упорядоченное множество X — совокупность объектов x_i с заданным на нем отношением порядка (расположенные в определенной последовательности). Конечное упорядоченное множество: $X=(x_1,\dots,x_n)$. Бесконечное упорядоченное множество: $X=(x_1,x_2,\dots)$. Пара упорядоченных элементов: (x_1,x_2).

Прямое (декартово) произведение $A\times B$ — множество всех упорядоченных пар, где первый элемент принадлежит множеству A, а второй множеству B: $A\times B=\{(a,b),a\in A,b\in B\}$.

Бинарное (двухместное) отношение R из множества X в множество Y — любое подмножество R прямого произведения $X\times Y$.

Функция f — бинарное отношение $f\subseteq X\times Y$, обладающее следующим свойством: если $(x,y_1)\in f$ и $(x,y_2)\in f$, то $y_1=y_2$ (свойство однозначности).

Кластерный анализ (иногда называемый численной таксономией, распознаванием образов или неконтролируемой классификацией) — многомерный статистический метод, применяемый для группирования множества объектов таким образом, чтобы объекты одной и той же группы, называемой кластером, были более похожи по каким-либо признакам, чем объекты других кластеров.

Кластеризация — автоматическое разбиение множества объектов на группы (кластеры) в зависимости от их схожести по каким-либо признакам.

Программное обеспечение (ПО) — программа или множество программ, используемых для управления компьютером.

k-means (k-средних) — популярный алгоритм кластеризации.

РЕФЕРАТ

Научно-исследовательская работа по теме «Кластерный анализ» содержит 105 страниц текста, 30 рисунков, 1 таблицы и 12 использованных источников.

КЛАСТЕРНЫЙ АНАЛИЗ, АЛГОРИТМЫ КЛАСТЕРИЗАЦИИ.

Цель работы — описать основы кластерного анализа и реализовать программу кластеризации алгоритмами k-means и k-means++.

В процессе работы были описаны основные этапы проведения кластерного анализа, приведены практические примеры использования теоретических знаний, реализовано ПО для кластеризации n-мерных данных алгоритмами k-means и k-means++.

В результате исследования был выявлен довольно широкий спектр возможностей кластерного анализа: разбиение объектов по целому набору параметров, отсутствие строгих ограничений на вид рассматриваемых объектов, возможность циклического приближения к необходимым исследователю результатам. Также были проанализированы достоинства, недостатки и общие характеристики алгоритмов k-means и k-means++.

ВВЕДЕНИЕ

Актуальность темы заключается в том, что кластерный анализ в последнее время широко используется в информатике, археологии, медицине, биологии, геологии и других областях, что обуславливает необходимость его дальнейшей теоретической и практической разработки.

Кластерный анализ как научное направление зародился в середине 60-х годов и с тех пор бурно развивается. Это вполне закономерно, так как речь идет практически о моделировании операции группирования, одной из важнейших как в статистике, так и вообще — в познании, в принятии решений.

Объектом исследования является кластерный анализ.

Предметом исследования является набор базовых положений и принципов работы кластерного анализа.

Целью исследования является описание основ кластерного анализа и реализация программы кластеризации алгоритмами k-means и k-means++.

Реализация цели исследования обусловила необходимость решения следующих задач:

1) Рассмотреть теоретическую основу кластерного анализа.

2) Изучить классификацию алгоритмов кластеризации.

3) Реализовать алгоритм k-means и проанализировать его характеристики, достоинства и недостатки.

4) Сделать тезисные выводы и предложения по результатам проведенного исследования.

Степень разработанности проблемы. Теоретической, методологической и практической основой исследования являются труды ведущих ученых-статистиков, чьи исследования тем или иным образом касаются вопросов современного кластерного анализа.

Кластерный анализ был основан в антропологии Харольдом Драйвером и Альфредом Кребером в 1932 году и введен в психологию Джозефом Зубиным в 1938 году и Робертом Трионом в 1939 году, широко

использовался Реймондом Кеттеллом с 1943 года для классификации теории признаков в психологии личности.

Теоретические исследования по кластерному анализу проводятся как отдельными зарубежными, так и отечественными учеными.

Практическая значимость исследования заключается в выводах и предложениях по результатам исследования поставленной в работе задачи, сделанных на основе исследования учебных пособий, книг и статей авторов, статистических и аналитических материалов. На основе публикаций проанализированы базовые положения и принципы кластерного анализа.

Научно-исследовательская работа на тему «Кластерный анализ» состоит из страницы с терминами и определениями, реферата, введения, трех глав, заключения, списка использованных источников и приложений.

ГЛАВА 1. КЛАСТЕРНЫЙ АНАЛИЗ

1.1 Место кластерного анализа в статистике

Статистика — 1) общеметодологическая наука, изучающая особенности массовых процессов и явлений; 2) отрасль практической деятельности, направленная на сбор, обработку, анализ и публикацию массовых данных об общественных и природных явлениях; 3) цифровой материал, который служит для характеристики какой-либо области общественных или природных явлений. [1]

Объект X — конечное упорядоченное множество n признаков: $X=(x_1,\dots,x_n)$, где x_i — i-й признак объекта. Примеры объектов: компьютер = (частота процессора, размер оперативной памяти, модель видеокарты), яблоко = (сладость, размер, аромат).

Иногда пишут слово «образ» («pattern») вместо «объект» («object»), подчеркивая, что для описания объекта используется только некоторая часть существенных признаков. Иначе говоря, образ — отображение объекта на множество его существенных признаков. Однако далее мы будем использовать слово «объект» и понимать под ним его образ.

Статистическая совокупность — конечное множество объектов, которые обладают одним и тем же набором признаков, но отличаются значениями этих признаков. [1] Примеры: персональные компьютеры = (Acer Aspire XC-330, Lenovo IdeaCentre 510S-07ICK, DEXP Atlas H203), яблоки = (Антоновка, Голден, Айдаред, Фуджи).

Признаки, которыми могут отличаться объекты статистической совокупности, носят разный характер. Кроме того, их может быть много. Они могут быть количественными: возраст, стаж работы, вес, рост и др. — тогда отдельные объекты отличаются друг от друга значениями этих признаков. Они могут быть качественными: образование, интересы, профессия и др. — тогда отдельные объекты отличаются друг от друга наличием или отсутствием того или иного качества, т. е. признаки не выражаются числовыми значениями. Количественные признаки могут быть дискретными и непрерывными. Качественные признаки могут быть атрибутивными (когда

есть несколько вариантов значений признака: цвет может быть белым, черным, синим и т. д.) и альтернативными (асимметричными: да/нет, есть/отсутствует, истина/ложь и др.; симметричными: мужчина/женщина, дети/родители и др.). [1]

Метод статистики — система приемов и способов, направленных на изучение признаков, их структуры, распределений, взаимосвязей и динамики. Метод статистики реализуется в три этапа: [1]

1. Статистическое наблюдение;

2. Сводка и группировка статистических данных;

3. Анализ, моделирование и прогнозирование изучаемых явлений.

Все эти этапы обязательно связаны между собой. Проведение статистического наблюдения не имеет смысла без дальнейшего анализа, а анализ невозможен без сводки и группировки статистических данных.

Второй этап — группировка статистических данных — является основным способом решения задачи классификации, а значит и основой всей дальнейшей работы с собранной информацией. Именно на этом этапе осуществляется кластерный анализ.

1.2 Введение в кластерный анализ

Кластерный анализ (иногда называемый численной таксономией, распознаванием образов или неконтролируемой классификацией) — многомерный статистический метод, применяемый для группирования множества объектов (той самой статистической совокупности) таким образом, чтобы объекты одной и той же группы, называемой кластером, были более похожи по каким-либо признакам, чем объекты других кластеров. [2]

Кластеризация — автоматическое разбиение множества объектов на группы (кластеры), в зависимости от их схожести по каким-либо признакам. От классификации отличается тем, что перечень групп изначально не задан и определяется в результате работы алгоритма.

В 1939 году впервые был определен предмет кластерного анализа и сделано его описание исследователем-психологом Робертом Трионом.

Однако активный интерес к данной теме пришёлся на период 60-80 гг. [3] Метод приобрел большую популярность за счет возможности разбиения объектов по нескольким признакам и отсутствии ограничений на тип рассматриваемых объектов.

Главное назначение методов кластерного анализа — определение внутренней структуры совокупности многомерных данных без предварительных предположений, в связи с чем их часто называют также классификацией без обучения.

Цели кластеризации: [4]

1. Понимание данных путем выявления кластерной структуры. Здесь число кластеров стараются делать поменьше и работать с каждым по отдельности.

2. Сжатие данных. Если исходная выборка большая, то можно сократить её, оставив по одному типичному представителю от каждого кластера. Здесь важно обеспечить высокую степень сходства объектов внутри каждого кластера, а самих кластеров может быть много.

3. Обнаружение новизны. Здесь подразумевается выделение нетипичных объектов, которые не удается присоединить ни к одному из кластеров. Эту задачу называют одноклассовой классификацией.

4. Другие специализированные цели.

Как правило, кластеризацию разделяют на две категории: жесткая кластеризация (или четкая кластеризация) и мягкая кластеризация (или нечеткая кластеризация). При жесткой кластеризации объект принадлежит одному и только одному кластеру, тогда как при мягкой кластеризации объект может принадлежать двум или более кластерам с некоторыми вероятностями. [5] Более подробная информация об этих категориях представлена далее (п. 1.2.2).

Если есть несколько признаков и нет обучающих выборок (отсутствует информация о распределении рассматриваемой совокупности объектов), то задача может быть решена методами кластерного анализа. Однако, как будет показано далее, решение задачи кластеризации принципиально

неоднозначно. Большая ответственность за качество кластеризации возлагается на эксперта-исследователя — специалиста предметной области. В частности, в кластерном анализе нет необходимости заранее делить признаки на зависимые и независимые. Зависимость признаков определяется экспертом по результатам кластеризации и толкового анализа, что отражает большую гибкость кластерного анализа по сравнению с другими методами.

Кластеризация данных — косвенный анализ данных, так как исследователь не совсем уверен, какие кластеры он ищет, что играет роль в формировании этих кластеров и как это происходит. Проблема кластеризации широко рассматривалась современными учеными, хотя единообразного определения кластеризации данных не существует и, возможно, никогда не будет существовать, так как в строгой постановке эта задача является неразрешимой. Действительно, чтобы определить конкретный объект, необходимо либо перечислить все то, что не является этим объектом, либо всю совокупность отличительных признаков, которые выделяют его из множества других объектов. В действительности, человек воспринимает и анализирует не весь объект, а лишь некоторую ограниченную совокупность его наиболее существенных признаков.

Кластерный анализ может применяться в: [6]

1. Биологии. Например, в количественном анализе экспрессии генов.

2. Психологии здоровья. В частности, в предупреждении болезней и инвалидности. В системах здравоохранения кластерный анализ используется для выявления групп людей, которым могут быть полезны определенные услуги. Кроме того, он используется для выявления групп людей с риском развития различных заболеваний.

3. Разных маркетинговых исследованиях. Например, для построения групп потребителей с целью получить максимально полное представление о том, как ведет себя клиент из какого-либо сегмента.

4. Сегментации изображений. При сегментации изображения кластерный анализ используется для обнаружения границ объектов.

5. Интеллектуальном анализе больших объемов данных с целью обнаружения полезной информации.

Как уже было сказано, объект X — конечное упорядоченное множество n признаков: $X=(x_1,\dots,x_n)$. Если речь идет о конкретном объекте, то пишут его наименование, в образце выше это X. Если речь идет конкретно о признаках какого-либо объекта, то пишут так называемый вектор признаков, в образце выше это $(x_1,\dots,x_n)$.

Будем обозначать буквой n число признаков, а m число объектов в кластере. Также i будет обозначать i-й признак, а j будет обозначать j-й объект кластера, $X_{j,i}$ — i-й признак j-ого объекта (т. е. объекта X_j).

Кластеризация данных включает в себя следующие этапы: [6]

1. Этап представления данных. Отбор выборки для кластеризации, выделение вектора признаков и (возможно) его стандартизация.

2. Этап моделирования и оптимизации. Определение меры сходства между объектами, разбиение объектов на группы по степени их схожести выбранным методом. Производится оценка качества, которая может быть либо оптимизирована, либо приближена во время поиска скрытых структур.

3. Этап проверки и представления результатов.

1.2.1 Этап представления данных

В самом начале нужно выделить признаки, которые будут описывать рассматриваемые объекты. Основная проблема в том, чтобы найти ту совокупность признаков, которая наилучшим образом отразит понятие сходства одних объектов и различие других.

Стоит попробовать выделить наиболее важные признаки, так как уменьшение количества признаков упрощает и ускоряет процесс кластеризации, иногда позволяет визуально оценивать результаты, однако не всегда приводит к наиболее верному выводу.

В качестве входных данных могут быть даны:

1. Векторы признаков. Можно представить в виде матрицы, где каждая строка будет представлять собой элементы вектора признаков каждого объекта (m объектов, n признаков):

$$Data=\begin{pmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,n} \\ X_{2,1} & X_{2,2} & \dots & X_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{m,1} & X_{m,2} & \dots & X_{m,n} \end{pmatrix}$$

2. Матрица расстояний между объектами. Элементы матрицы — значения $d(X_{j_1}, X_{j_2})$ в строке j_1 и столбце j_2. Матрица получается симметричной ($d(X_{j_1}, X_{j_2}) = d(X_{j_2}, X_{j_1})$ для всех j_1 и j_2). Очевидно, что на главной диагонали нули.

$$Data=\begin{pmatrix} 0 & d(X_1, X_2) & \dots & d(X_1, X_m) \\ d(X_2, X_1) & 0 & \dots & d(X_2, X_m) \\ \vdots & \vdots & \ddots & \vdots \\ d(X_m, X_1) & d(X_m, X_2) & \dots & 0 \end{pmatrix}$$

3. Матрица сходства между объектами. В данном случае каждый объект описывается степенями сходства с другими объектами. Например, степень сходства может определяться как расстояние (различие) между объектами от 0 до 1: $[0,1]$.

4. Специализированные виды входных данных.

Большинство алгоритмов кластеризации либо полностью исходит из матрицы расстояний, либо требует вычисления отдельных её элементов.

Как уже было сказано, признаки могут быть количественными и качественными. Количественные обладают свойством упорядоченности, поэтому в этом случае проблем с построением матрицы расстояний возникать не будет. Качественные не обладают свойством упорядоченности, поэтому вопрос не может быть решен тривиальным образом. В частности, качественным признакам можно поставить в соответствие некоторые числа (количественная оценка качества всевозможных предметов и процессов изучается научной дисциплиной под названием «квалиметрия»). Однако такие числа вряд ли будут отражать какую-либо реальную упорядоченность. Исключением являются альтернативные признаки, два значения которых можно считать упорядоченными. Например, если какой-то признак допускает варианты «есть/отсутствует», то можно принять «есть» как 1, а «отсутствует» как 0. Могут быть и другие числа.

Входные данные в таблице должны соответствовать одному типу признаков. Однако преобразование к одному типу является нетривиальной задачей. Существует три подхода: [6]

1) использовать коэффициент подобия, учитывающий информацию всех необходимых признаков;

2) нестрогое (интуитивное) преобразование к одному типу;

3) проведение отдельных анализов одного и того же набора объектов, причем каждый анализ включает рассматриваемые объекты с одним типом признаков (без других типов), а затем все результаты синтезируются.

Значения признаков объектов можно стандартизировать. [6]

Стандартизация (нормирование/нормировка) приводит значения всех признаков объектов к единому диапазону значений (например, $[0,1]$ или $[-1,1]$) для того, чтобы все компоненты давали одинаковый вклад при расчете расстояния. В широком смысле под стандартизацией можно понимать изменение дисперсии (или, что эквивалентно, диапазона) значений признаков. Стандартизировать можно как некоторые отдельные кластеры, так и все рассматриваемые (глобальная стандартизация).

Стандартизация может быть важна, если исследователь работает с данными, где признаки имеют разные единицы (например, дюймы, метры, тонны и килограммы), так как расстояние между объектами не может измеряться одновременно в разных единицах измерения, или где шкалы этих признаков сильно отличаются друг от друга (например, $[0,1]$ против $[0,1000]$), так как кластерный анализ сосредотачивается на множестве значений с большей дисперсией. Например, при кластеризации данных с использованием возраста, измеренного в годах, и оценки по 10-балльной шкале, данные по 10-балльной шкале будут играть гораздо меньшую роль в кластерном анализе, поскольку больший диапазон возрастных категорий преобразуется в большую дисперсию.

Стандартизация не характерна для опросных исследований. Во многих других областях статистического анализа она уместна. Например, при

кластеризации измерений веса и роста некоторой группы людей имеем две разные шкалы, т. е. здесь нужна стандартизация.

Стандартизация чаще всего необходима, если используются меры расстояния (п. 1.2.2). Но стоит иметь в виду, что в других случаях она может привести к увеличению схожести ранее различных объектов.

Два наиболее распространенных способа стандартизации: [6]

1. Среднеквадратичная стандартизация.

$$Z_j = \frac{X_j}{\sigma}, \quad \sigma = \sqrt{\frac{1}{m-1}\sum_{j=1}^{m}(X_j - \mu)^2}, \quad \mu = \sum_{j=1}^{m} X_j p_j, \quad \sum_{j=1}^{m} p_j = 1$$

где σ — среднеквадратичное отклонение выборки на основании несмещённой оценки дисперсии (n-мерная точка), μ — математическое ожидание выборки (n-мерная точка), в котором p_j чаще всего[1] можно считать равным $1/m$, X_j — j-й объект входных данных, Z_j — стандартизированный по отношению к X_j объект.

Все унарные (унарный минус, корень квадратный и др.) и бинарные (сложение, умножение и др.) операции с объектами выполняются напрямую со значениями признаков:

1. Любая унарная операция над объектом:

$$\circ(x_1, \ldots, x_n) = (\circ x_1, \ldots, \circ x_n)$$

2. Любая бинарная операция над двумя объектами с одинаковым числом признаков:

$$(x_1, \ldots, x_n) \circ (y_1, \ldots, y_n) = (x_1 \circ y_1, \ldots, x_n \circ y_n)$$

3. Любая бинарная операция объекта с числом:

$$NUMBER \circ (x_1, \ldots, x_n) = (NUMBER \circ x_1, \ldots, NUMBER \circ x_n)$$

$$(x_1, \ldots, x_n) \circ NUMBER = (x_1 \circ NUMBER, \ldots, x_n \circ NUMBER)$$

Поскольку оценки не были центрированы путем вычитания среднего, то информация о местоположении между объектами остается.

1 Среднее арифметическое равно математическому ожиданию только в том случае, если распределение равномерно (все точки одинаково вероятны). Так получилось, что среднее арифметическое часто оказывается достаточным средним для стандартизации.

Пример. $X_1=(2,7,9,10)$, $X_2=(390,400,416,454)$, $\mu_1=196$, $\mu_2=203.5$, $\mu_3=212.5$, $\mu_4=232$, $\sigma_1=274.4$, $\sigma_2=278$, $\sigma_3=288$, $\sigma_4=314$. $Z_1=(0.007,0.025,0.031,0.032)$, $Z_2=(1.421,1.439,1.444,1.446)$.

2. Вычисление Z-оценки или стандартизованной оценки. Используется только при глобальной стандартизации. Информация о местоположении между объектами теряется.

Стандартизованная случайная величина — это случайная величина, математическое ожидание которой равно нулю, а среднеквадратичное отклонение — единице. Любой вектор признаков с математическим ожиданием μ и среднеквадратичным отклонением σ может быть преобразован к стандартизованному вектору признаков по формуле:

$$Z_j=\frac{X_j-\mu}{\sigma}$$

Пример. $X_1=(2,7,9,10)$, $X_2=(390,400,416,454)$, $\mu_1=196$, $\mu_2=203.5$, $\mu_3=212.5$, $\mu_4=232$, $\sigma_1=274.4$, $\sigma_2=278$, $\sigma_3=288$, $\sigma_4=314$. $Z_1=(-0.707,-0.707,-0.707,-0.707)$, $Z_2=(0.707,0.707,0.707,0.707)$.

Также можно воспользоваться следующими формулами:

$$Z_j=\frac{X_j}{\mu}, Z_j=\frac{X_j}{X_{max}}, Z_j=\frac{X_j-X_{min}}{X_{max}-X_{min}}, Z_j=\frac{X_j-\mu}{X_{max}-X_{min}}, \ldots$$

где X_{max} — объект (возможно, несуществующий) с наибольшими значениями признаков, а X_{min} — с наименьшими. Например, если $X_1=(1,0), X_2=(0,1)$, то $X_{max}=(1,1), X_{min}=(0,0)$.

Следующую формулу будем называть MinMax-оценкой:

$$Z_j=\frac{X_j-X_{min}}{X_{max}-X_{min}}$$

Наряду со стандартизацией признаков, существует вариант придания каждой из них определенного коэффициента важности, или веса ($0\le w_i\le 1$, $w\in\mathbb{R}$), который бы отражал значимость соответствующего признака. В качестве весов могут выступать экспертные оценки, полученные в ходе опроса экспертов. Взвешивание следует применять, только если для этого имеется хорошее теоретическое обоснование и известна процедура

определения весов. Самая крайняя форма взвешивания — исключение некоторых признаков (значение 0). В ходе экспериментов возможно сравнение результатов, полученных с учетом экспертных оценок и без них, выбор лучшего. Возможны другие виды преобразования. [6]

В результате все объекты представляются в виде окончательных векторов признаков: $X_1=(x_{1,1}, x_{1,2}, \ldots, x_{1,n}), \ldots, X_m=(x_{m,1}, x_{m,2}, \ldots, x_{m,n})$.

1.2.2 Этап моделирования

Объекты, принадлежащие одному и тому же кластеру, чаще всего не являются полностью идентичными. Поэтому необходимо определить критерии, в соответствии с которыми будет определяться сходство (или несходство) между объектами.

На этом этапе возникает трудность, связанная с неоднозначностью определения меры сходства (несходства) между объектами. Наиболее популярная мера — мера расстояния (метрика), которая оценивает сходство количественно. При этом подходе объекты представляются точками координатного n-мерного пространства признаков, причем замеченные сходства и различия между точками находятся в соответствии с метрическими расстояниями между ними. [2]

Метрика выбирается в зависимости от пространства признаков, в котором расположены объекты (двумерное пространство для двух признаков, трехмерное для трех и т. д.), и от цели кластеризации.

Мера расстояния между двумя объектами U и V в пространстве признаков: $d(U,V)$ — удовлетворяет следующим аксиомам: [2]

1. $d(U,V)>0$ или $d(U,V)=0 \Leftrightarrow U=V$ ($d(U,U)=0$) (различимость нетождественных объектов и неразличимость идентичных объектов).

2. $d(U,V)=d(V,U)$ (симметрия).

3. $d(U,V)+d(V,W)\geq d(U,W)$ (неравенство треугольника).

Очевидно, что $d(U,V)$ определяет сходство по всем признакам двух объектов, в частности полное сходство между ними означает их одинаковое положение в пространстве признаков.

Очевидно, что мера расстояния — это n-мерная функция, которая ставит двум точкам (объектам) их расстояние в n-мерном пространстве.

Право выбора метрики полностью лежит на исследователе, поскольку результаты кластеризации могут существенно отличаться.

Примеры метрик приведены далее. [2]

1. Расстояние Евклида. Наиболее распространенная функция расстояния. Представляет собой геометрическое расстояние в n-мерном пространстве:

$$d_E(U,V)=\sqrt{\sum_{i=1}^{n}(U_i-V_i)^2}$$

где U_i — значение i-ого признака объекта U, V_i — значение i-ого признака объекта V.

Пример.

$$d_E((0,1),(2,3))=\sqrt{(0-2)^2+(1-3)^2}=\sqrt{4+4}=\sqrt{8}=2\sqrt{2}$$

Расстояние измеряется по прямой линии от U до V. Если рассматривать координатную плоскость, то совокупность точек, равноудаленных от начала координат по расстоянию Евклида, будет образовывать окружность (точки равноудалены от центра радиусом).

Использовать расстояние Евклида нужно, если выполнено хотя бы одно из следующих условий:

1) Признаки взаимно независимы и имеют одну и ту же дисперсию.

2) Признаки однородны по физическому смыслу и одинаково важны для классификации.

3) Пространство признаков совпадает с геометрическим пространством.

2. Квадрат расстояния Евклида. Применяется для придания большего веса более отдаленным друг от друга объектам.

$$d_E^2(U,V)=\sum_{i=1}^{n}(U_i-V_i)^2$$

Пример.

$$d_E^2((0,1),(2,3))=(0-2)^2+(1-3)^2=4+4=8$$

3. Расстояние городских кварталов (манхэттенское расстояние или расстояние L1). Для этой меры влияние отдельных выбросов меньше, чем при использовании расстояния Евклида, поскольку здесь координаты не возводятся в квадрат.

$$d_{L1}(U,V)=\sum_{i=1}^{n}|U_i-V_i|$$

Пример.

$$d_{L1}((0,1),(2,3))=|0-2|+|1-3|=2+2=4$$

Расстояние измеряется параллельными осям отрезками от U до V. Если рассматривать координатную плоскость, то совокупность точек, равноудаленных от начала координат по расстоянию городских кварталов, будет образовывать повернутый на 45 градусов квадрат.

4. Расстояние Чебышева. Может оказаться полезным, когда нужно определить два объекта как «различные», если они различаются по какой-либо одной координате.

$$d_{Ch}(U,V)=\max_{1\le i\le n}|U_i-V_i|$$

Пример.

$$d_{Ch}((0,1),(2,3))=\max(|0-2|,|1-3|)=2$$

Расстояние измеряется параллельными осям отрезками и диагоналями от U до V. Если рассматривать координатную плоскость, то совокупность точек, равноудаленных от начала координат по расстоянию городских кварталов, будет образовывать квадрат.

5. Степенное расстояние. Применяется в случае, когда необходимо увеличить или уменьшить вес схожести объектов.

$$d_{pow}(U,V)=\sqrt[r]{\sum_{i=1}^{n}(U_i-V_i)^p}$$

где r и p — фиксированные параметры, определяемые пользователем.

6. Расстояние Махаланобиса — мера несходства между двумя случайными объектами U и V из одного распределения вероятностей с матрицей ковариации COV:

$$d_M(U,V)=\sqrt{(U-V)*COV^{-1}*(U-V)^T}$$

Символ T означает операцию транспонирования, а под COV^{-1} подразумевается матрица, обратная ковариационной COV ($COV*COV^{-1}=$ $=COV^{-1}*COV=E$, где E — единичная матрица).

Метрика Махаланобиса позволяет значительно улучшить процедуру кластерного анализа в следующих случаях:

1) Признаки объектов выбраны неадекватно, кластеры плохо разделяются;

2) Признаки объектов сильно коррелируют между собой (зависят друг от друга);

3) Поверхности между кластерами имеют очень сложную форму.

Для того, чтобы найти расстояние, в метрике Махаланобиса используются такие величины как математическое ожидание (чаще всего совпадающее со средним арифметическим значением), дисперсия, среднеквадратичное отклонение и матрица ковариаций.

Ковариация — это численное выражение свойства ковариантности двух признаков объектов. Свойство ковариантности означает, что признаки имеют тенденцию изменяться совместно (ковариантно). В этом случае говорят, что признаки коррелируют.

Ковариационная матрица состоит из ковариаций между всеми парами признаков. Если количество признаков равно n, то ковариационная матрица — матрица размерности $n \times n$, имеющая вид:

$$COV=\begin{pmatrix} cov_{1,1} & cov_{1,2} & \dots & cov_{1,n} \\ cov_{2,1} & cov_{2,2} & \dots & cov_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ cov_{n,1} & cov_{n,2} & \dots & cov_{n,n} \end{pmatrix}$$

Элементы ковариационной матрицы — ковариации — для набора объектов вычисляются по формуле:

$$cov_{a,b}=\frac{1}{m-1}\sum_{j=1}^{m}(X_{j,a}-\mu_a)(X_{j,b}-\mu_b)$$

где $X_{j,a}$ — a-й признак j-ого объекта, $X_{j,b}$ — b-й признак j-ого объекта, μ_a и μ_b — средние значения a-ого и b-ого признаков по набору объектов соответственно.

Ковариации обладают следующими важными свойствами:

1. Если при переходе от одного объекта к другому a-й и b-й признаки увеличиваются вместе, то $cov_{a,b}>0$;

2. Если при переходе от одного объекта к другому a-й признак увеличивается, а b-й уменьшается (или наоборот), то $cov_{a,b}<0$;

3. Если при переходе от одного объекта к другому a-й и b-й изменяются независимо, то $cov_{a,b}=0$ (обратное утверждение в общем случае неверно);

4. Неравенство Коши — Буняковского: $cov_{a,b}\leq\sigma_a\sigma_b$, где σ_a — среднеквадратичное отклонение набора элементов по a-ому признаку и σ_b — по b-ому признаку соответственно. Т. е. $cov_{a,b}\in[-\sigma_a\sigma_b,+\sigma_a\sigma_b]$.

Так как для вычисления метрики Махаланобиса требуется найти обратную к COV матрицу, а матрица обратима тогда и только тогда, когда она является квадратной и невырожденной (определитель не равен нулю), то можно отметить, что ковариационная матрица COV будет вырожденной в следующих двух случаях:

1. Если количество объектов m в данном наборе меньше, чем количество признаков n плюс 1, т. е. $m<n+1$.

2. Если степень корреляции признаков максимальна ($cov_{a,b}=\sigma_a\sigma_b$).

В этих двух случаях нельзя обратить матрицу COV.

Если нельзя обратить матрицу COV, то следует вычислять расстояние по формуле стандартизированного расстояния:

$$d_{std}(U,V)=\sqrt{\sum_{i=1}^{n}\left(\frac{U_i-V_i}{\sigma_i}\right)^2}$$

где σ_i — среднеквадратичное отклонение выборки по i-ому признаку.

Алгоритм вычисления расстояния по Махаланобису.

Шаг 1. Вычислить математические ожидания (средние арифметические значения в случае равномерного распределения) признаков объектов.

$$\mu=\frac{X_1+X_2+...+X_m}{m}=\frac{1}{m}\sum_{j=1}^{m}X_j$$

Шаг 2. Вычислить дисперсии и среднеквадратичные отклонения.

Шаг 3. Вычислить ковариации между всеми парами признаков и составить ковариационную матрицу.

Шаг 4. Если матрица обратима, то вычислить расстояние по Махаланобису. Если матрица необратима, то вычислить расстояние по формуле стандартизированного расстояния.

Рассмотрим подробно процедуру вычисления расстояний в метрике Махалонобиса на следующем примере. Пусть в результате предварительной кластеризации 14 объектов, имеющих два признака, были распределены на три кластера (рис. 1).

$$C_{(1)}=\{X_1,X_2,X_3,X_4,X_5\},X_1=(1,1),X_2=(2,2),X_3=(3,3),X_4=(4,4),X_5=(5,5)$$

$$C_{(2)}=\{X_6,X_7,X_8,X_9,X_{10}\},X_6=(3,1),X_7=(4,0),X_8=(6,0),X_9=(6,2),X_{10}=(5,3)$$

$$C_{(3)}=\{X_{11},X_{12},X_{13},X_{14}\},X_{11}=(1,5),X_{12}=(2,4),X_{13}=(2,5),X_{14}=(1,4)$$

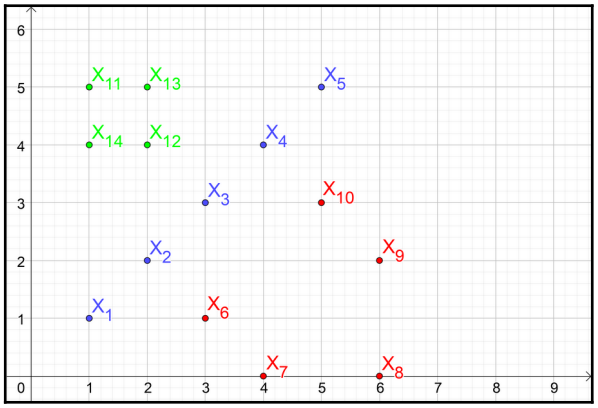

Рисунок 1 — Объекты-точки, разбитые на три кластера

Найдем математические ожидания.

$$\mu_{(1)}=\left(\frac{1+2+3+4+5}{5},\frac{1+2+3+4+5}{5}\right)=\left(\frac{15}{5},\frac{15}{5}\right)=(3,3)$$

$$\mu_{(2)}=\left(\frac{3+4+6+6+5}{5},\frac{1+0+0+2+3}{5}\right)=\left(\frac{24}{5},\frac{6}{5}\right)=(4.8,1.2)$$

$$\mu_{(3)}=\left(\frac{1+2+2+1}{4},\frac{5+4+5+4}{4}\right)=\left(\frac{6}{4},\frac{18}{4}\right)=(1.5,4.5)$$

Далее, необходимо вычислить дисперсии и среднеквадратичные отклонения признаков.

По первому признаку объектов первого кластера:

$$\sigma^2_{(1)1}=\frac{(1-3)^2+(2-3)^2+(3-3)^2+(4-3)^2+(5-3)^2}{5-1}=\frac{10}{4}=2.5$$

По второму признаку объектов первого кластера:

$$\sigma^2_{(1)2}=\frac{(1-3)^2+(2-3)^2+(3-3)^2+(4-3)^2+(5-3)^2}{5-1}=\frac{10}{4}=2.5$$

По первому признаку объектов второго кластера:

$$\sigma^2_{(2)1}=\frac{(3-4.8)^2+(4-4.8)^2+(6-4.8)^2+(6-4.8)^2+(5-4.8)^2}{5-1}=\frac{6.8}{4}=1.7$$

По второму признаку объектов второго кластера:

$$\sigma^2_{(2)2}=\frac{(1-1.2)^2+(0-1.2)^2+(0-1.2)^2+(2-1.2)^2+(3-1.2)^2}{5-1}=\frac{6.8}{4}=1.7$$

По первому признаку объектов третьего кластера:

$$\sigma^2_{(3)1}=\frac{(1-1.5)^2+(2-1.5)^2+(2-1.5)^2+(1-1.5)^2}{4-1}=\frac{1}{3}=0.333$$

По второму признаку объектов третьего кластера:

$$\sigma^2_{(3)2}=\frac{(5-4.5)^2+(4-4.5)^2+(5-4.5)^2+(4-4.5)^2}{4-1}=\frac{1}{3}=0.333$$

Среднеквадратичные отклонения — квадратные корни из дисперсии:

$$\sigma_{(1)1}=\sqrt{2.5}=1.581\,,\sigma_{(1)2}=\sqrt{2.5}=1.581$$

$$\sigma_{(2)1}=\sqrt{1.7}=1.3\,,\sigma_{(2)2}=\sqrt{1.7}=1.3$$

$$\sigma_{(3)1}=\sqrt{0.333}=0.577\,,\sigma_{(3)2}=\sqrt{0.333}=0.577$$

Теперь нужно вычислить матрицы ковариаций для обоих кластеров.

Для первого кластера:

$$cov_{(1)1,1}=\sigma^2_{(1)1}=2.5\,,cov_{(1)2,2}=\sigma^2_{(1)2}=2.5\,,cov_{(1)2,1}=cov_{(1)1,2}$$ [2]

$$cov_{(1)1,2}=\frac{1}{5-1}\left|(X_{(1)1,1}-\mu_{(1)1})(X_{(1)1,2}-\mu_{(1)2})+\ldots+(X_{(1)5,1}-\mu_{(1)1})(X_{(1)5,2}-\mu_{(1)2})\right|=$$

$$=\frac{1}{4}\left|(1-3)(1-3)+(2-3)(2-3)+(3-3)(3-3)+(4-3)(4-3)+(5-3)(5-3)\right|=\frac{10}{4}=2.5$$

2 Данные равенства легко проверить вручную.

Получаем следующую матрицу ковариаций:

$$COV_{(1)}=\begin{pmatrix} cov_{(1)1,1} & cov_{(1)1,2} \\ cov_{(1)2,1} & cov_{(1)2,2} \end{pmatrix}=\begin{pmatrix} 2.5 & 2.5 \\ 2.5 & 2.5 \end{pmatrix}$$

Для второго кластера:

$$cov_{(2)1,1}=\sigma^2_{(2)1}=1.7\,,cov_{(2)2,2}=\sigma^2_{(2)2}=1.7\,,cov_{(2)2,1}=cov_{(2)1,2}$$

$$cov_{(2)1,2}=\frac{1}{5-1}\left|(X_{(2)1,1}-\mu_{(2)1})(X_{(2)1,2}-\mu_{(2)2})+...+(X_{(2)5,1}-\mu_{(2)1})(X_{(2)5,2}-\mu_{(2)2})\right|=$$

$$=\frac{1}{4}\left|(3-4.8)(1-1.2)+(4-4.8)(0-1.2)+(6-4.8)(0-1.2)+(6-4.8)(2-1.2)+(5-4.8)(3-1.2)\right|=$$

$$=\frac{1.2}{4}=0.3$$

Получаем следующую матрицу ковариаций:

$$COV_{(2)}=\begin{pmatrix} cov_{(2)1,1} & cov_{(2)1,2} \\ cov_{(2)2,1} & cov_{(2)2,2} \end{pmatrix}=\begin{pmatrix} 1.7 & 0.3 \\ 0.3 & 1.7 \end{pmatrix}$$

Для третьего кластера:

$$cov_{(3)1,1}=\sigma^2_{(3)1}=0.333\,,cov_{(3)2,2}=\sigma^2_{(3)2}=0.333\,,cov_{(3)2,1}=cov_{(3)1,2}$$

$$cov_{(3)1,2}=\frac{1}{4-1}\left|(X_{(3)1,1}-\mu_{(3)1})(X_{(3)1,2}-\mu_{(3)2})+...+(X_{(3)4,1}-\mu_{(3)1})(X_{(3)4,2}-\mu_{(3)2})\right|=$$

$$=\frac{1}{3}\left|(1-1.5)(5-4.5)+(2-1.5)(4-4.5)+(2-1.5)(5-4.5)+(1-1.5)(4-4.5)\right|=0$$

Получаем следующую матрицу ковариаций:

$$COV_{(3)}=\begin{pmatrix} cov_{(3)1,1} & cov_{(3)1,2} \\ cov_{(3)2,1} & cov_{(3)2,2} \end{pmatrix}=\begin{pmatrix} 0.333 & 0 \\ 0 & 0.333 \end{pmatrix}$$

Пусть теперь предлагается отнести объект $(4,2)$ к одному из этих трех кластеров. Для этого необходимо вычислить расстояния между этим объектом и центрами этих кластеров.

Для первого кластера вычисляем определитель матрицы ковариаций. Он равен 0. Значит, для матрицы ковариаций первого кластера нельзя найти обратную матрицу, поэтому расстояние между новым объектом и первым кластером вычислим по формуле стандартизированного расстояния:

$$d_{std}((4,2),(3,3))=\sqrt{\left(\frac{4-3}{1.581}\right)^2+\left(\frac{2-3}{1.581}\right)^2}=\sqrt{0.8}=0.8945$$

Теперь найдем расстояние от этого объекта до центра второго кластера. Вычисляем определитель матрицы ковариаций. Он равен $2.8 \neq 0$. Значит, находим обратную матрицу с помощью матрицы алгебраических дополнений:

$$COV_{(2)}^{-1} = \frac{1}{\Delta_{(2)}} A_{(2)}^{T}$$

где $\Delta_{(2)}$ — определитель матрицы $COV_{(2)}$, $A_{(2)}^{T}$ — транспонированная матрица алгебраических дополнений для матрицы ковариаций второго кластера.

$$A_{(2)}^{T} = \begin{pmatrix} A_{(2)1,1} & A_{(2)2,1} \\ A_{(2)1,2} & A_{(2)2,2} \end{pmatrix} = \begin{pmatrix} 1.7 & -0.3 \\ -0.3 & 1.7 \end{pmatrix}$$

$$COV_{(2)}^{-1} = \frac{1}{2.8} \begin{pmatrix} 1.7 & -0.3 \\ -0.3 & 1.7 \end{pmatrix} = \begin{pmatrix} 0.607 & -0.107 \\ -0.107 & 0.607 \end{pmatrix}$$

Найдем расстояние между объектом и вторым кластером:

$$d_M((4,2),(4.8,1.2)) = \sqrt{((4,2)-(4.8,1.2)) * COV^{-1} * ((4,2)-(4.8,1.2))^T} =$$

$$= \sqrt{(4-4.8,2-1.2) * COV^{-1} * (4-4.8,2-1.2)^T} =$$

$$= \sqrt{(-0.8,0.8) * \begin{pmatrix} 0.607 & -0.107 \\ -0.107 & 0.607 \end{pmatrix} * (-0.8,0.8)^T} =$$

$$= \sqrt{(-0.5712,0.5712) \begin{pmatrix} -0.8 \\ 0.8 \end{pmatrix}} = \sqrt{0.91392} = 0.95599$$

Теперь найдем расстояние от этого объекта до центра третьего кластера. Вычисляем определитель матрицы ковариаций. Он равен $0.111 \neq 0$. Значит, находим обратную матрицу по аналогии:

$$COV_{(3)}^{-1} = \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}$$

Найдем расстояние между объектом и третьим кластером:

$$d_M((4,2),(1.5,4.5)) = \sqrt{((4,2)-(1.5,4.5)) * COV^{-1} * ((4,2)-(1.5,4.5))^T} =$$

$$= \sqrt{(2.5,-2.5) * \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix} * (2.5,-2.5)^T} =$$

$$= \sqrt{(7.5,-7.5) \begin{pmatrix} 2.5 \\ -2.5 \end{pmatrix}} = \sqrt{37.5} = 6.124$$

Итак, расстояние от объекта до первого кластера равно 0.8945, до второго кластера 0.95599, до третьего кластера 6.124. Теперь если посмотреть на рис. 1 и мысленно добавить точку $(4,2)$, то можно понять, что все найденные расстояния соответствуют действительности. Таким образом, объект следует отнести к первому кластеру.

Расширением понятия расстояния Махаланобиса является обобщенное (взвешенное) расстояние Махаланобиса, формула которой отличается от последней наличием дополнительного сомножителя — симметрической неотрицательно-определенной матрицы весовых коэффициентов Λ, недиагональные элементы которой чаще всего равны нулю: [2]

$$d_M(U,V)=\sqrt{(U-V)*\Lambda*COV^{-1}*\Lambda^T*(U-V)^T}$$

Пример матрицы весовых коэффициентов (является единичной, поэтому на результат не повлияет) для объектов с двумя признаками:

$$\Lambda=\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$$

7. Процент несогласия. Используется в тех случаях, когда признаки объекта являются качественными:

$$d_{disagree}(U,V)=\frac{1}{n}\sum_{i=1}^{n}(U_i\neq V_i)$$

где $(U_i\neq V_i)$ — возвращает 1, если неравны, или 0, если равны.

Пример. Первый признак объекта — пол, второй — возраст, третий — место работы. Пусть первый объект — (мужской, 20 лет, учитель), второй объект — (мужской, 28 лет, менеджер). Процент несогласия равен $\frac{2}{3}$. Т. е. объекты различаются на 66.6%.

Несмотря на очевидную важность метрик, они — отнюдь не единственный способ описания сходства объектов. [2] Имеет право существовать асимметрия сходства. Например, объект A является квадратом, а объект B является прямоугольником, при определении сходства получим, что объект A схож с объектом B (всякий квадрат является прямоугольником), но объект B необязательно схож с объектом A

(не всякий прямоугольник является квадратом). В таких случаях используются другие меры сходства: коэффициенты корреляции, коэффициенты ассоциативности, вероятностные коэффициенты сходства.

Коэффициент корреляции между объектами может быть вычислен для признаков, измеренных по шкалам отношений или шкалам интервалов. Значение коэффициента корреляции изменяется в пределах от -1 до $+1$; значение 0 указывает, что между объектами нет связи. [2]

Коэффициент корреляции нечувствителен к различиям в величине признаков. Это свойство особенно важно для многих приложений кластерного анализа в социальных науках (психология, социология). [2]

Классический пример — выборочный коэффициент корреляции Пирсона. Этой мере соответствует корреляционная матрица сходства объектов, диагональные элементы которой — единицы — отвечают максимальной степени близости объектов, а близость недиагональных элементов к нулю означает минимальную степень их близости (если исследователя интересует только сила взаимосвязи между признаками, используются абсолютные величины коэффициентов корреляции). [2]

К недостаткам коэффициентов корреляции между объектами следует отнести отсутствие статистического смысла (среднее значение определяется не по совокупности объектов, а по совокупности признаков, что вообще возможно только для безразмерных и стандартизированных признаков и не имеет точного смысла), а также несоблюдение требований метричности. [2]

Коэффициенты ассоциативности применяются в качестве меры сходства объектов, описываемых бинарными переменными. Предложено довольно много таких коэффициентов; особенно широко они используются в биологии. Наиболее распространены три меры: простой коэффициент совстречаемости, коэффициент Жаккара и коэффициент Гауэра. Первые два основаны на таблице ассоциативности 2×2, в которой 1 указывает на наличие признака, 0 — на его отсутствие (табл. 1). [2]

Таблица 1 — Таблица с матрицей ассоциативности

Объект 1	Объект 2	
	1	0
1	a	b
0	c	d

где a — число случаев, когда у обоих объектов присутствует признак, d — число случаев, когда у обоих объектов отсутствует признак, b и c — когда признак есть у одного объекта и его нет у другого.

Простой коэффициент совстречаемости имеет вид:

$$S = \frac{a+d}{a+b+c+d}$$

где S — свойство между двумя объектами, величина которого измеряется в пределах от 0 до 1.

Из этой формулы следует, что простой коэффициент совстречаемости учитывает также и отсутствие признака у обоих объектов (клетка матрицы ассоциативности d). Это не всегда удобно, поскольку некоторые объекты оказываются в значительной степени схожими не за счет наличия некоторых признаков, а за счет их отсутствия. В коэффициенте Жаккара этот недостаток устранен путем исключения числа d:

$$S = \frac{a}{a+b+c}$$

Видно, что коэффициент Жаккара принимает в расчет только те признаки, которые характерны хотя бы для одного из объектов.

Коэффициент Гауэра — обобщение коэффициента Жаккара на случаи, когда необходимо оценить сходство объектов, характеризуемых признаками, измеренными по разным шкалам. Этот коэффициент обладает рядом преимуществ, но редко встречается. В области социальных наук коэффициент Гауэра практически не используется. [2]

Вероятностные коэффициенты сходства существенно отличаются от всех предыдущих тем, что сходство между двумя объектами не вычисляется вообще. Мера сходства определяется по всему массиву исходных данных как

информационный выигрыш от объединения двух объектов и/или кластеров. Коэффициенты этого типа применимы только к бинарным данным и широко используются специалистами по численной таксономии и экологии. В социальных науках они практически не применяются. [2]

Также есть другие более специализированные меры.

Если нет меры сходства или несходства между парами точек данных, то значимый кластерный анализ невозможен.

Кластер C (от англ. «cluster» — гроздь, скопление) — множество схожих по некоторым критериям оптимальности объектов X_j: $C=\{X_1,\dots,X_m\}$, где m — количество объектов в кластере C. Критерий оптимальности — целевая функция, которая определяет степень принадлежности объекта кластеру C.

Кластеризация отличается от классификации тем, что кластеры изначально не заданы, и даже может быть неизвестно их число.

Смысл кластера объединяет различные правдоподобные критерии и требует, чтобы все объекты в кластере: [5]

1. Обладали одинаковыми или тесно связанными признаками.

2. Предполагали небольшие взаимные расстояния или несходства.

3. Имели отношения хотя бы с одним другим объектом в группе.

4. Четко отличались от всех остальных рассматриваемых объектов.

Кроме того, интуитивно предполагается, что:

1. Существуют относительно густонаселенные кластерами области пространства.

2. Кластеры окружены относительно пустыми областями пространства.

Целевая функция — функция, которая (в широком смысле) выражает уровень желательности различных разбиений (группировок).

Решение задачи кластерного анализа — совокупность разбиений (группировок), которые удовлетворяют целевой функции.

У кластера можно выделить следующие свойства: центр, плотность, дисперсия, размеры, форма, отделимость.

Центр кластера — среднее точек (объектов) кластера в пространстве.

$$ArithMean(C)=\frac{X_1+X_2+...+X_m}{m}=\frac{1}{m}\sum_{j=1}^{m}X_j$$

$$GeoMean(C)=\sqrt[m]{X_1*X_2*...*X_m}=\sqrt[m]{(\prod_{j=1}^{m}X_j)}$$

где $ArithMean(C)$ — среднее арифметическое объектов кластера C, $GeoMean(C)$ — среднее геометрическое объектов кластера C.

Пример (рис. 2).

$$\frac{1}{3}(\underset{A}{(3,1)}+\underset{B}{(1,4)}+\underset{C}{(3,4)})=\frac{1}{3}(7,9)=\underset{ArithMean}{(2.33,3)}$$

$$\sqrt[3]{\underset{A}{(3,1)}*\underset{B}{(1,4)}*\underset{C}{(3,4)}}=\sqrt[3]{(9,16)}=\underset{GeoMean}{(2.08,2.52)}$$

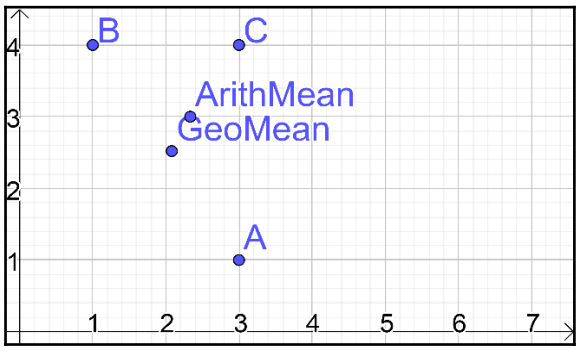

Рисунок 2 — Центры кластера *ArithMean* и *GeoMean*

Плотность — это свойство, которое позволяет определить кластер как относительно плотное скопление точек в пространстве данных по сравнению с другими областями пространства, содержащими либо мало точек, либо не содержащих их вообще.

Дисперсия — это степень рассеяния точек в пространстве данных относительно центра кластера. Дисперсия характеризует, насколько близко друг к другу расположены в пространстве данных точки кластера. По этому признаку кластер может быть плотным (точки находятся вблизи его центра) и неплотным (точки разбросаны вокруг центра).

Размер кластера связан с дисперсией. В случае плотного кластера гиперсферической формы можно измерить его радиус.

Радиус кластера — максимальное расстояние точек от центра кластера.

$$radius(C)=\max_j d(Mean(C),X_j)$$

Пример. Из примера выше: $d(GeoMean(C), A)=1.78$, $d(GeoMean(C), B)=1.83$, $d(GeoMean(C), C)=1.74$. Максимальное из них 1.83, т. е. $radius(C)=1.83$ в соответствии со средним геометрическим.

Теперь $d(ArithMean(C), A)=1.78$, $d(ArithMean(C), B)=1.67$, $d(ArithMean(C), C)=2.11$. Максимальное из них 2.11, т. е. $radius(C)=2.11$ в соответствии со средним арифметическим.

На рисунке 3 представлены две окружности c и d, первая относится к $GeoMean$, вторая к $ArithMean$.

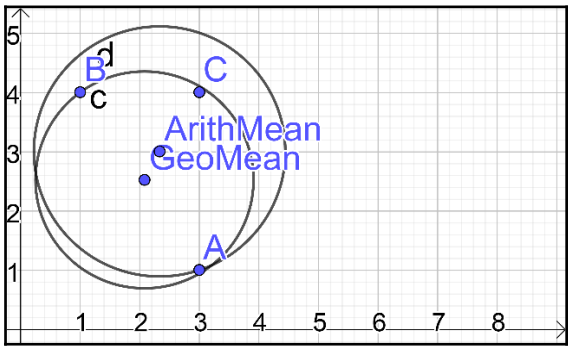

Рисунок 3 — Окружности описывают кластер из трех объектов

Форма — это расположение точек в пространстве данных. Кроме гиперсферической и эллипсоидальной формы кластеров, возможны кластеры удлиненной формы. В этом случае вместо радиуса вычисляют связность точек в кластере — относительную меру расстояния между ними.

В общем случае кластеры по форме могут быть разделены на компактные и сцепленные.

Компактный кластер — набор объектов, которые имеют большое взаимное сходство (рис. 4). Обычно компактное скопление может быть представлено репрезентативным объектом или центром.

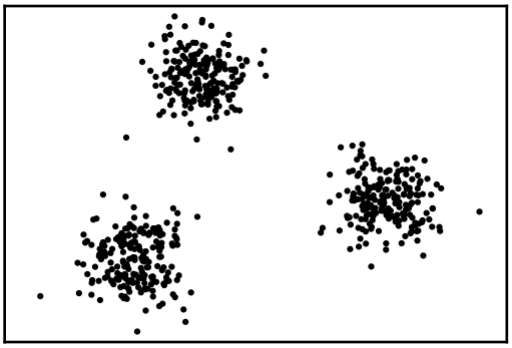

Рисунок 4 — Три хорошо разделенных кластера с центром в двумерном пространстве

Связанный кластер — набор объектов, каждый из которых больше похож на другие объекты в кластере, чем на другие вне кластера (рис. 5).

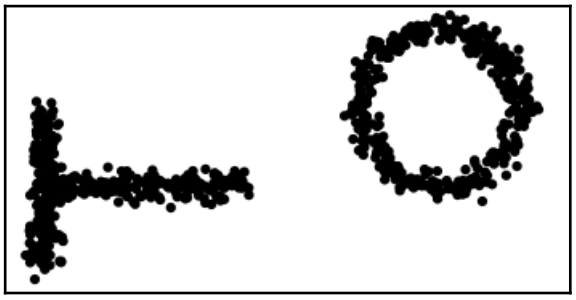

Рисунок 5 — Два сцепленных кластера в двумерном пространстве

Отделимость кластеров характеризует степень их перекрытия и насколько далеко друг от друга они расположены в пространстве данных. Кластеры могут быть относительно близко друг к другу и не иметь четких границ, или же могут быть разделены широкими участками пустого пространства. [7]

Объект относится к рассматриваемому кластеру, если расстояние от этого объекта до центра этого кластера меньше, чем расстояние от этого объекта до центра любого другого кластера. Если это условие выполняется для двух и более кластеров (т. е. когда расстояния от объекта до нескольких кластеров равны), объект является спорным. Спорный объект — объект, который по мере сходства может быть отнесен к нескольким кластерам. В этом случае невозможно при помощи математических процедур однозначно отнести объект к одному кластеров. Поэтому существуют две категории кластеризации: жесткая (четкая) и мягкая (нечеткая).

При жесткой кластеризации каждый объект может принадлежать одному и только одному кластеру. [6]

Математически результат алгоритмов жесткой кластеризации может быть представлен матрицей U размера $k \times m$:

$$U = \begin{pmatrix} U_{1,1} & U_{1,2} & \cdots & U_{1,m} \\ U_{2,1} & U_{2,2} & \cdots & U_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ U_{k,1} & U_{k,2} & \cdots & U_{k,m} \end{pmatrix}$$

где k — количество кластеров, m — количество объектов, а $U_{p,j}$ удовлетворяет следующим условиям:

$$U_{p,j} \in \{0,1\}, 1 \le p \le k, 1 \le j \le m$$

$$\sum_{p=1}^{k} U_{p,j} = 1, 1 \le j \le m$$

$$\sum_{j=1}^{m} U_{p,j} > 0, 1 \le p \le k$$

Первое ограничение подразумевает, что каждый объект либо принадлежит кластеру (единица), либо нет (нуль). Второе ограничение подразумевает, что каждый объект принадлежит ровно одному кластеру. Третье ограничение подразумевает, что каждый кластер содержит хотя бы один объект, т. е. пустые кластеры не допускаются. U в этом случае называется жестким k -разбиением набора данных. [6]

В мягкой кластеризации предположение ослабляется, так что объект может принадлежать одному или нескольким кластерам с вероятностями (от 0 до 1: $[0,1]$). Результат алгоритмов мягкой кластеризации также может быть представлен матрицей U размера $k \times m$ со следующими ослабленными ограничениями: [6]

$$U_{p,j} \in [0,1] \subset \mathbb{R}, 1 \le p \le k, 1 \le j \le m$$

$$\sum_{p=1}^{k} U_{p,j} = 1, 1 \le j \le m$$

$$\sum_{j=1}^{m} U_{p,j} > 0, 1 \le p \le k$$

Аналогично, U в этом случае называется мягким k -разбиением.

Свойства кластеров, выделяемых в процессе кластеризации, во многом зависят от определения понятия расстояния между кластерами. Различают расстояние между кластерами, измеряемое по принципу ближайшего соседа, дальнего соседа и расстояние, измеряемое по центроидам кластеров (рис. 6). [2]

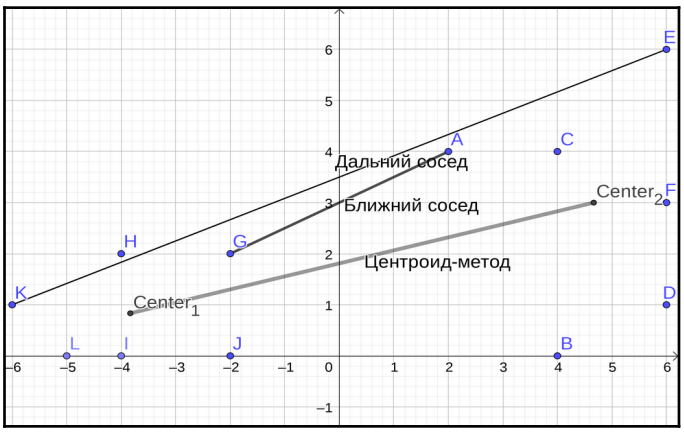

Рисунок 6 — Расстояния между кластерами, измеренные методом ближнего соседа, методом дальнего соседа и методом центроида

Расстояние, измеренное по методу ближайшего соседа определяется как расстояние между ближайшими элементами кластеров:

$$d(C_1, C_2) = \min_{i \in C_1,\, j \in C_2} |C_{1,i} - C_{2,j}|$$

Расстояние, измеренное по методу дальнего соседа определяется как расстояние между наиболее удаленными элементами кластеров:

$$d(C_1, C_2) = \max_{i \in C_1,\, j \in C_2} |C_{1,i} - C_{2,j}|$$

Расстояние, измеренное по методу центроида определяется как:

$$d(C_1, C_2) = d(Mean(C_1), Mean(C_2))$$

где $Mean(C_1)$ и $Mean(C_2)$ — два объекта со средними значениями признаков по всем объектам кластеров C_1 и C_2 соответственно.

Наиболее близкий к реальной группировке результат позволяет получить метод центроида. Но это не означает бесполезность применения двух других методов. Метод ближайшего соседа «сжимает» пространство, образуя минимально возможное число больших кластеров. Метод дальнего соседа «расширяет» пространство, образуя максимально возможное число компактных кластеров. Каждый из трех методов привносит в реальное соотношение объектов свою структуру и представляет собой свою точку зрения. Исследователь, в зависимости от стоящей перед ним задачи, вправе выбрать тот метод, который больше подходит.

Средние величины в статистике дают обобщающую характеристику анализируемого явления. Самая распространённая из них — среднее

арифметическое. Она применяется в случае образования обобщенного показателя с помощью суммы элементов. Очевидно, что среднее арифметическое нескольких вещественных чисел — число, которым можно заменить каждое из этих чисел так, чтобы их сумма не изменилась.

Действительно, $\frac{a_1+...+a_n}{n}=A \Rightarrow A*n=\underbrace{A+...+A}_{n\ операндов}=\underbrace{a_1+...+a_n}_{n\ операндов}$.

При изменении значения какого-либо признака по сравнению с её предыдущим значением (т. е. если данные измеряются по шкале отношений), обобщенный показатель следует образовывать не в результате суммирования, а в результате произведения.

Пример. Месячная инфляция — изменение уровня цен одного месяца по сравнению с предыдущим. Показатели инфляции за каждый месяц образуют цепной индекс, поэтому сумма месячных показателей инфляций не подходит. В таком случае используют произведение и называют средним показателем — среднее геометрическое. Среднее геометрическое нескольких вещественных чисел — число, которым можно заменить каждое из этих чисел так, чтобы их произведение не изменилось. Действительно, $\sqrt[n]{a_1*...*a_n}=G \Rightarrow G^n=\underbrace{G*...*G}_{n\ операндов}=\underbrace{a_1*...*a_n}_{n\ операндов}$.

Среднее арифметическое может значительно отличаться от среднего геометрического. Если, например, представлены объекты: $(1,1)$, $(1,2)$, $(2,1)$, $(2,2)$, $(100,200)$, то при определении центра в соответствие со средним арифметическим мы получили бы точку $(21.2,41.2)$, которая находится довольно далеко от скопления большинства объектов, а в соответствии со средним геометрическим точку $(3.31,3.81)$. Очевидно, что объект $(100,200)$ сильно влияет на положение центра кластера, определяемого средним арифметическим. Выбор средней величины остается за экспертом: он может выбирать и отличные от рассмотренных двух вариантов средних величин.

На практике среднее геометрическое может встретиться в теме индексов развития человеческого потенциала, в которой сравнивают уровень жизни в разных странах.

Работа кластерного анализа опирается на два предположения. Первое предположение: рассматриваемые признаки объекта в принципе допускают желательное разбиение множества объектов на кластеры. Второе предположение: правильность выбора масштаба или единиц измерения признаков.

Решение задачи кластеризации принципиально неоднозначно, и тому есть несколько причин: [4]

1. Не существует однозначно наилучшего критерия качества кластеризации. Известен целый ряд эвристических критериев, а также ряд алгоритмов, не имеющих чётко выраженного критерия, но осуществляющих достаточно разумную кластеризацию «по построению». Все они могут давать разные результаты.

2. Число кластеров, как правило, неизвестно заранее и устанавливается в соответствии с некоторым субъективным критерием.

3. Результат кластеризации существенно зависит от метрики, выбор которой, как правило, также субъективен и определяется экспертом.

1.2.3 Этап представления результатов

Результаты кластеризации должны быть представлены в удобном для обработки виде. Например: [8]

1. Представление кластеров центроидами.

2. Представление кластеров набором характерных точек.

3. Представление кластеров их ограничениями.

Кроме того, для визуального понимания некоторые результаты можно отобразить в виде специальных диаграмм. Например, в виде дендрограмм, которые показывают степень близости отдельных объектов и кластеров, а также наглядно демонстрируют в графическом виде последовательность их объединения или разделения. Количество уровней дендрограммы соответствует числу шагов слияния или разделения кластеров. В

дендрограмме, представленной на рисунке 7, на первом шаге группируются объекты X_2 и X_3, образуя кластер $\{X_2, X_3\}$ с минимальным расстоянием (например, Евклидовым) между объектами, примерно равным 1. Затем объекты X_4 и X_5 группируются в кластер $\{X_4, X_5\}$ с расстоянием между ними, равным 1.5. Расстояние между кластерами $\{X_2, X_3\}$ и $\{X_1\}$ также оказывается равным 1.5, что позволяет сгруппировать их на том же уровне, что и $\{X_4, X_5\}$. И наконец, два кластера $\{X_1, X_2, X_3\}$ и $\{X_4, X_5\}$ группируются на самом высоком уровне иерархии с расстоянием 2.

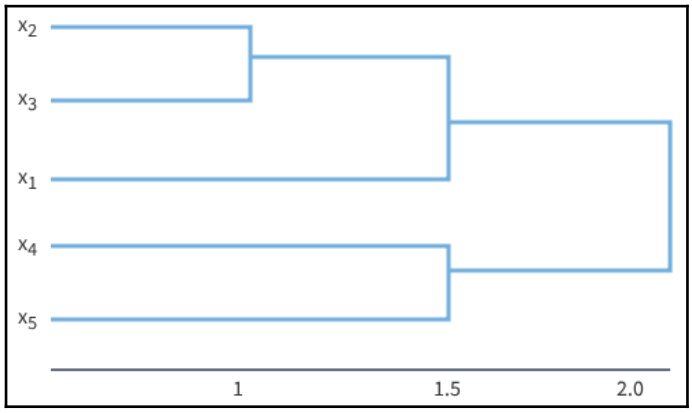

Рисунок 7 — Пример дендрограммы

1.3 Достоинства и недостатки кластерного анализа

1.3.1 Достоинства

Большое достоинство кластерного анализа в том, что он позволяет производить разбиение объектов не по одному параметру, а по целому набору. Кроме того, кластерный анализ в отличие от большинства математико-статистических методов не накладывает никаких ограничений на вид рассматриваемых объектов, и позволяет рассматривать множество исходных данных практически произвольной природы. Кластерный анализ позволяет рассматривать большой объем информации и резко сокращать, сжимать большие массивы, делать их компактными и наглядными. [9]

Важное значение кластерный анализ имеет применительно к совокупностям временных рядов, характеризующих экономическое развитие (например, общехозяйственной и товарной конъюнктуры). Здесь можно выделять периоды, когда значения соответствующих показателей были

достаточно близкими, а также определять группы временных рядов, динамика которых наиболее схожа. [9]

Кластерный анализ можно использовать циклически. В этом случае исследование производится до тех пор, пока не будут достигнуты необходимые результаты. При этом каждый цикл может давать информацию, которая способна сильно изменить направленность и подходы дальнейшего применения кластерного анализа. Этот процесс можно представить системой с обратной связью. В задачах социально-экономического прогнозирования весьма перспективно сочетание кластерного анализа с другими количественными методами (например, с регрессионным анализом). [9]

1.3.2 Недостатки

Как и любой другой метод, кластерный анализ имеет определенные недостатки и ограничения: в частности, состав и количество кластеров зависит от выбираемых критериев разбиения. При сведении исходного массива данных к более компактному виду могут возникать определенные искажения, а также могут теряться индивидуальные черты отдельных объектов за счет замены их характеристиками обобщенных значений параметров кластера. При проведении классификации объектов игнорируется очень часто возможность отсутствия в рассматриваемой совокупности каких-либо значений кластеров. [9]

ГЛАВА 2. КЛАССИФИКАЦИЯ АЛГОРИТМОВ КЛАСТЕРИЗАЦИИ

2.1 Введение

Перечислим основные предостережения: [7]

1. Многие методы кластерного анализа — довольно простые процедуры, которые, как правило, не имеют достаточного статистического обоснования. Т. е. они — не более, чем правдоподобные алгоритмы.

2. Методы кластерного анализа разрабатывались для многих научных дисциплин, поэтому не несут на себе отпечатки специфики этих дисциплин.

3. Разные кластерные методы порождают различные решения для одних и тех же данных. Желательно иметь специальную методику, позволяющую проверить, насколько «естественны» выделенные группы.

4. Методы кластеризации необходимы для обнаружения структуры в данных, которую нелегко найти при визуальном обследовании или с помощью экспертов.

Несмотря на то что многие методы кластерного анализа довольно элементарны, применение методов кластерного анализа стало возможным только в 80-е годы с возникновением и развитием вычислительной техники. Это объясняется тем, что эффективное решение задачи поиска кластеров требует большого числа арифметических и логических операций. Рассмотрим три различных подхода к проблеме кластерного анализа: эвристический, экстремальный и статистический.

Эвристический подход характеризуется отсутствием формальной модели изучаемого явления и критерия для сравнения различных решений. Его основой является интуитивно-построенный алгоритм.

При экстремальном подходе также не формулируется исходная модель, а задается критерий, определяющий качество разбиения на кластеры. Такой подход особенно полезен, если цель исследования четко определена. В этом случае качество разбиения может измеряться эффективностью выполнения цели.

Основой статистического подхода решения задачи кластерного анализа является вероятностная модель исследуемого процесса. Статистический

подход особенно удобен для теоретического исследования проблем, связанных с кластерным анализом. Кроме того, он дает возможность ставить задачи, связанные с воспроизводимостью результатов кластерного анализа.

В общем случае алгоритмы жесткой кластеризации можно разделить на две категории: иерархические алгоритмы и алгоритмы разделения. Есть два типа иерархических алгоритмов: разделяющие иерархические алгоритмы и агломеративные иерархические алгоритмы. В разделяющем иерархическом алгоритме алгоритм идет сверху вниз, т. е. алгоритм начинается с одного большого кластера, содержащего все точки данных в наборе данных, и продолжает разбивать кластеры; в агломеративном иерархическом алгоритме алгоритм работает снизу вверх, т. е. алгоритм начинается с кластеров, каждый из которых содержит одну точку данных, и продолжает объединение кластеров. В отличие от иерархических алгоритмов, алгоритмы разделения создают одноуровневое неперекрывающееся разделение точек данных. [6]

Для больших наборов данных иерархические методы становятся непрактичными, если не используются другие методы, так как обычно они составляют $O(m^2)$ по памяти и $O(m^3)$ по времени, где m — количество рассматриваемых объектов. [6]

В реальных наборах данных можно столкнуться с двумя проблемами: некоторые важные данные отсутствуют в наборах данных, и в наборах данных могут быть ошибки. Существует три случая, в зависимости от того, как пропущенные значения могут появляться в наборах данных: [6]

1. Отсутствующие значения встречаются в нескольких объектах.

2. Отсутствующие значения встречаются в ряде признаков.

3. Отсутствующие значения случайным образом появляются в объектах и признаках.

Если в наборе данных существует объект или признак, по которому отсутствуют все измерения, то на самом деле информация об этом объекте или признаке отсутствует, поэтому его необходимо удалить из набора данных. Если в объекте или признаке не так много пропущенных значений,

методы работы с пропущенными значениями можно разделить на две группы: [6]

1. Методы предварительной замены, которые заменяют отсутствующие значения перед процессом интеллектуального анализа данных;

2. Встроенные методы, которые работают с пропущенными значениями в процессе интеллектуального анализа данных.

Существует много методов, способных работать с отсутствующими значениями.

2.2 Классификация алгоритмов

Приведем классификацию алгоритмов кластеризации: [10]

1. По характеру отношения, которое отыскивается как результат классификации.

1.1. Разбиение с непересекающимися кластерами (отношение эквивалентности). Все объекты внутри найденного кластера считаются тождественными, а объекты разных кластеров — нет.

1.2. Разбиение с пересекающимися кластерами. Задается по-разному: введением степени принадлежности объекта к кластеру, определением вероятности принадлежности объекта кластеру или просто перечнем объектов в зоне пересечения.

1.3. Иерархическое дерево. Отыскивает целую систему вложенных разбиений. Такая сложная структура далеко не всегда соответствует представлениям о строении совокупности. Но иерархическими алгоритмами можно отыскивать и «обычные» разбиения.

1.4. Отношение произвольной структуры. Наиболее общий способ анализа структуры множества — аппроксимация его некоторым отношением с произвольной (заранее заданной) структурой. Такая задача, вообще говоря, выходит за рамки кластерного анализа.

2. По степени участия человека в процедуре выделения кластеров.

2.1. Человек не принимает участия в работе алгоритма, он только формирует входные данные (выбирает расстояние, задает параметры), кластеризация производится машинным способом.

2.2. Человек участвует в процессе получения разбиения. Алгоритм выдает не собственно классификацию, а информацию, на основании которой человек принимает решения о разбиении. Такими процедурами фактически являются все методы визуализации данных. Они обычно дают максимально полезные содержательные результаты, несмотря на свою нестрогость.

3. По характеру априорных сведений (задаваемых параметров) для работы алгоритма.

3.1. Априорные сведения отсутствуют (свободная кластеризация). Считается, что отсутствие задаваемых параметров необходимо для выделения естественной, а не навязанной структуры исходных данных. На самом деле лишь немногие алгоритмы не опираются ни на какие параметры (и они имеют человеко-машинный характер).

3.2. Задано число кластеров. Популярный способ — «ограничения» свободной кластеризации. Если такой алгоритм проработает в целом спектре кластеров, «естественный» результат может быть выделен после дополнительной обработки.

3.3. Заданы пороговые значения величины близости объектов. Способов задания порогов очень много. Объединим их, чтобы противопоставить предыдущему параметру — числу кластеров. На деле существует связь между этими параметрами для каждого алгоритма, но в явном виде она не изучена.

3.4. Заданы комбинированные сведения (число кластеров и пороги разных типов). Процедуры носят наиболее «ограниченный» характер, но умелое использование многих параметров может сделать алгоритм довольно реалистичным и гибким.

4. По характеру работы алгоритма.

4.1. Процедуры, зависящие от порядка просмотра точек. Зависимость от нумерации точек является серьезным недостатком алгоритма.

4.2. Процедуры, не зависящие от порядка просмотра точек. Например, иерархические алгоритмы.

ГЛАВА 3. ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ

В качестве примера реализуем алгоритм k-means при помощи фреймворка Qt (https://www.qt.io/, версия 5.7.1; GCC 6.3.0 от 16.05.2017) с дополнительной библиотекой QCustomPlot (https://www.qcustomplot.com, версия 2.0.1 от 25.06.2018), используя язык C++.

Планируемые возможности программы:

— Добавление данных из файла формата CSV[3];

— Генерация случайных данных (заполненная окружность, заполненный прямоугольник);

— Отображение данных в виде неизменяемого списка;

— Сохранение внутренних данных в файл любого формата;

— Наличие интерактивного графика;

— Выбор двух признаков для отображения на графике (оси X и Y);

— Сохранение графика в виде изображения;

— Работа с кластерами: объединение, удаление;

— Стандартизация данных (среднеквадратичная оценка, Z-оценка, MinMax-оценка);

— Кластеризация данных алгоритмом k-means (k-means++).

3.1 Начало работы

Изначально определимся с тем, какие классы будут составлять основу данных:

1. Класс точки. Представление в виде массива вещественных чисел. Стандартные контейнерные возможности: получение первого элемента (значения первого признака), получение последнего элемента, получение элемента по индексу, получение числа признаков, операторы сравнения, итераторы. Также операторы сложения, вычитания, умножения, деления, метод возведения в степень, метод вычисления расстояния Евклида между

3 CSV (от англ. Comma-Separated Values — значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, каждый из которых разделен запятой или точкой с запятой. В нашем случае строка представляет собой один объект, а значения в строке представляют собой значения признаков объекта.

текущей точкой и точкой-аргументом. Динамическое добавление и удаление признаков запрещены. Разрешена инициализация количеством признаков.

2. Кластер. Представление в виде массива точек. Стандартные контейнерные возможности, кроме динамического добавления и удаления точек. Также метод, возвращающий сумму точек кластера. Разрешена инициализация количеством точек и признаков, а также массивом точек.

Воспользуемся схемой разделения «Model-View-Presenter» (MVP, «Модель-Вид-Представитель»): [11]

1. Model предоставляет данные для отображения и реагирует на команды Presenter, изменяя своё состояние;

2. View отвечает за отображение данных и перенаправление события от пользователя в Presenter;

3. Presenter содержит в себе всю бизнес-логику: получает события от View, оповещает Model об изменениях (при необходимости получает от него данные), преобразует данные Model для отображения во View.

В качестве Model реализуем класс PlotModel. Он будет содержать массив всех кластеров. Возможности: добавление и удаление любого кластера по индексу, получение числа кластеров, получение числа признаков. Т. е. будет обеспечивать всю необходимую работу с внутренними данными.

В качестве View будем использовать элемент QCustomPlot (из библиотеки QCustomPlot; наследуется от QWidget). Никакие изменения исходного кода этой библиотеки производиться не будут.

В качестве Presenter реализуем класс PlotController. MainWindow будет получать сигналы от View и использовать логику PlotController. PlotController будет содержать методы добавления, удаления, объединения кластеров, получения числа кластеров и признаков, метод, который будет запускать кластеризацию, настройку стиля кластеров, сохранения графика в файл, методы стандартизации и некоторые вспомогательные методы. MainWindow будет обеспечивать связь View и Presenter.

Также понадобиться два диалоговых окна. Первое для отображения настроек и внутренних данных. Второе для настройки генерации случайных данных.

Наиболее важная работа будет происходить в классе KMeansPlusPlus, предназначенном для кластеризации данных. Один из его методов будет принимать на вход массив точек и возвращать массив кластеров.

3.2 Метод k-means

Метод k-means — популярный метод кластеризации. Был изобретён в 1956 году математиком Гуго Штейнгаузом[4] и почти одновременно (в 1957 году) Стюартом Ллойдом[5]. Особую популярность приобрёл после работы Дж. Маккуина в 1967 году[6].

Кластеризация k-means.

Дано: массив n-мерных точек длиной m, число кластеров k, число итераций $iterations$[7].

Необходимо найти: массив k кластеров $clusters$ и массив центроидов $centroids$ длиной k.

Инициализация:

1. Выбираем случайно k точек из набора в качестве центроидов (добавляем в массив $centroids$), где k — требуемое количество кластеров.

Основной алгоритм:

2. Формируем массив кластеров $clusters$, где каждый кластер изначально пуст.

3. Вычисляем расстояния от каждой точки до каждого центроида. Если к текущей точке ближе всех остальных находится центроид с индексом i (т. е. $centroids[i]$), то добавляем эту точку в i-й кластер массива $clusters$ (т. е. в $clusters[i]$).

---

4 Steinhaus H. (1956). Sur la division des corps materiels en parties. Bull. Acad. Polon. Sci., C1. III vol IV: 801—804

5 Lloyd S. (1957). Least square quantization in PCM's. Bell Telephone Laboratories Paper

6 MacQueen J. (1967). Some methods for classification and analysis of multivariate observations. In Proc. 5th Berkeley Symp. on Math. Statistics and Probability, pages 281—297

7 Время работы k-means в наихудшем случае является суперполиномиальным по входным данным.

4. Вычисляем средние значения по каждому кластеру и заносим их в соответствующие элементы $centroids$. Очищаем кластеры массива $clusters$.

5. Повторяем шаги 3-4 необходимое количество итераций $iterations$.

6. Выполняем шаги 3-4 без очистки массива $clusters$.

7. Получаем результат: массив k кластеров $clusters$ и массив центроидов $centroids$ длиной k.

На каждой итерации происходит изменение границ кластеров и смещение их центров. В результате минимизируется расстояние между элементами внутри каждого кластера. В нашем случае число итераций задается заранее, так как в некоторых случаях центроиды сходятся довольно долго. Однако возможен вариант с остановкой в момент, когда границы кластеров и расположения центроидов перестают изменяться или изменяются незначительно (с погрешностью ε). При таком варианте достаточно одной переменной, которая отвечает за абсолютную максимальную разницу между центроидами при одной итерации и при второй итерации (если эта разница меньше или равна некоторому числу ε, которое может быть равно нулю, алгоритм завершается).

k-means++ — улучшенная версия алгоритма кластеризации k-means. Суть улучшения заключается в нахождении более «хороших» начальных значений центроидов кластеров. Был предложен в 2007 году Дэвидом Артуром и Сергеем Васильвицким в качестве способа избежать иногда плохой кластеризации, обнаруживаемой стандартным алгоритмом.

Алгоритм k-means++ определяет только процедуру инициализации центроидов перед основным алгоритмом k-means.

Этот метод дает значительное улучшение конечной ошибки k-means. Хотя первоначальный выбор в алгоритме k-means++ вычислительно дороже, все же основная часть сходится гораздо быстрее. Это связано с тем, что первоначально выбранные центроиды, вероятно, уже лежат в разных кластерах. Требуемое число итераций можно уменьшать в разы.

Инициализация k-means++ (упрощенный вариант алгоритма):

1. Выбираем любую точку в качестве первого центроида. По итогу один центроид считаем обработанным.

2. Пока все центроиды не обработаны выполняем следующее: находим наиболее удаленную точку от всех обработанных центроидов (для каждой точки находим расстояние до ближайшего обработанного центроида, затем среди всех этих значений находим наибольшее, которое и будет соответствовать искомой наиболее удаленной точке) и выбираем её в качестве текущего центроида, далее переходим к следующему центроиду.

Преимуществами алгоритма k-means (k-means++) является простота реализации и использования.

Метод k-means (k-means++) имеет следующие недостатки:

— Необходимо знать число кластеров заранее.

— Не гарантируется достижение глобального минимума. Один из способов исправить это — запустить алгоритм несколько раз и выбрать тот результат, который дает наилучшее значение целевой функции.

— Метод стремится к обнаружению гиперсферических кластеров.

— Результат сильно зависит от выбора исходных центров кластеров, их самый оптимальный выбор неизвестен, так как и стандартный k-means, и k-means++ являются приближениями к NP-трудной задаче оптимизации.

— Метод не учитывает плотность данных в кластере. Если есть очень плотная область, где размещается много данных, то среднее значение будет находиться ближе к этой плотной области.

Существуют методы кластеризации, которые можно рассматривать как происходящие от k-means. Например, в методе k-medians для вычисления центроидов используется не среднее, а медиана, что делает алгоритм более устойчивым к аномальным значениям в данных. [12]

Алгоритм g-means (от gaussian) строит кластеры, распределение данных в которых стремится к нормальному (гауссовскому) и снимает неопределенность выбора начальных кластеров. Алгоритм c-means использует элементы нечеткой логики, учитывая при вычислении

центроидов не только расстояния, но и степень принадлежности наблюдения к множеству объектов в кластере. [12]

3.3 Обзор программного обеспечения

Запустим ПО (рис. 8).

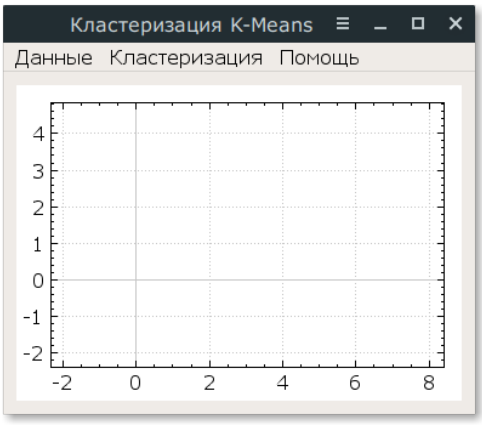

Рисунок 8 — Главное окно ПО

Видим график и главное меню. График интерактивен: можно перемещаться при помощи ведущей кнопки мыши, увеличивать и уменьшать масштаб при помощи колесика мыши. В главном меню имеется три пункта: «Данные», «Кластеризация» и «Помощь».

Посмотрим на пункт «Данные» (рис. 9).

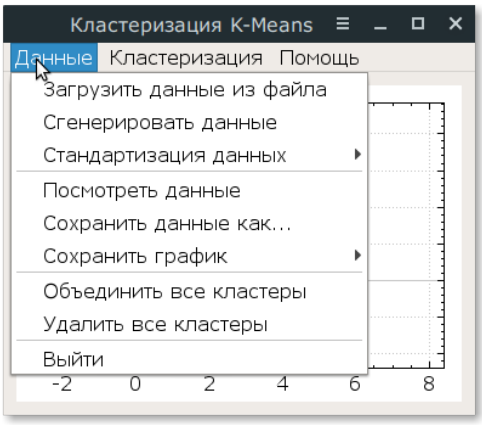

Рисунок 9 — Пункт меню «Данные»

Создадим файл в каком-либо текстовом редакторе под названием «main.csv» со следующими данными:

```
/ этот длинный комментарий в начале файла не является обязательным
/ комментариев может быть множество, в разных местах
/ перед комментариями располагается знак '/'
/ символы ';' и ',' не работают здесь
   1   ;   2   / числа без точки; сколько угодно пробелов
   1.5   ;   2.5   / числа с точкой; сколько угодно пробелов
```

```
/ все числа вводятся через ';' или ',':
   2.5   ;   3.5
/ вместо пробелов могут быть любые другие пробельные символы,
/ в том числе Tab'ы:
    1   ,   2.5
/ символы ';' и ',' в конце строки не ставятся
/ строки могут быть пустыми -- тогда строка не считается объектом

/ длинный комментарий в конце файла
```

Загрузим данные в программу из этого файла (рис. 10).

Рисунок 10 — Результат добавления данных из файла

Появились 4 точки. Все они обозначены синим цветом. Цвет устанавливается случайным образом, исключая темные цвета.

Теперь воспользуемся пунктом «Сгенерировать данные». Появится окно с настройками генерации (рис. 11).

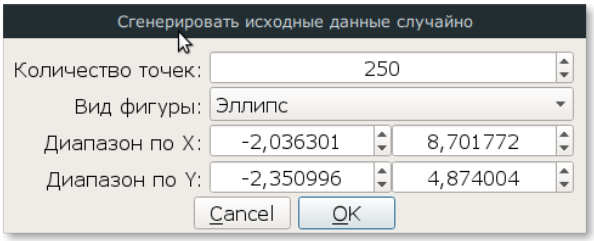

Рисунок 11 — Окно настройки генерации данных

Можно задать количество точек, вид фигуры: эллипс или прямоугольник, диапазон генерации (выбирается автоматически на основе диапазонов графика; точность до 6 знака, не более; можно генерировать данные в одной точке). Оставим как есть и нажмем «ОК» (рис. 12).

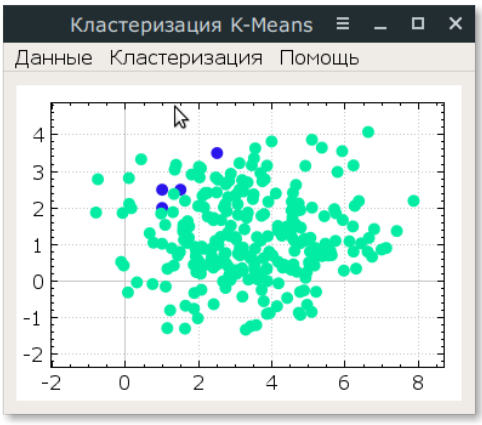

Рисунок 12 — Генерация прошла успешно

Видно, что на графике точки отличаются цветом. На самом деле это два кластера. При добавлении данных с файла или генерации на график добавляется новый кластер, который окрашивается в случайный цвет. Разделение на кластеры при добавлении сделано для удобства просмотра и не играет большой роли. Если нужно объединить все кластеры в один, достаточно выбрать пункт меню «Объединить все кластеры».

Теперь выберем вид фигуры «прямоугольник» (рис. 13).

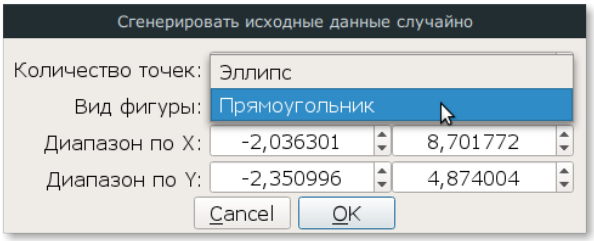

Рисунок 13 — Генерируем новые данные

Получим результат (рис. 14).

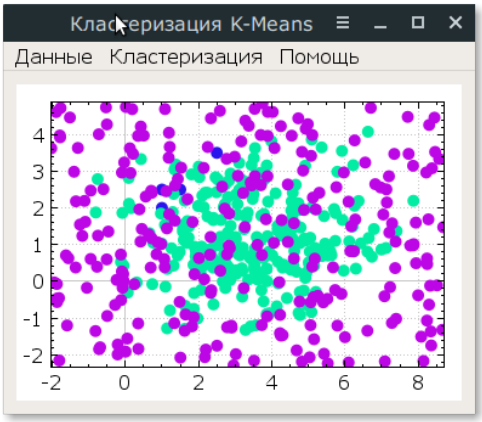

Рисунок 14 — Результат добавления прямоугольного вида фигуры

Данные можно стандартизировать. В программе доступны следующие виды стандартизации: среднеквадратичная оценка, Z-оценка, MinMax-

оценка. Все они работают с каждым кластером отдельно. Для того, чтобы работать вместе со всеми (глобальная стандартизация), нужно выбрать пункт меню «Объединить все кластеры», а затем стандартизировать.

Применим среднеквадратичную оценку (рис. 15).

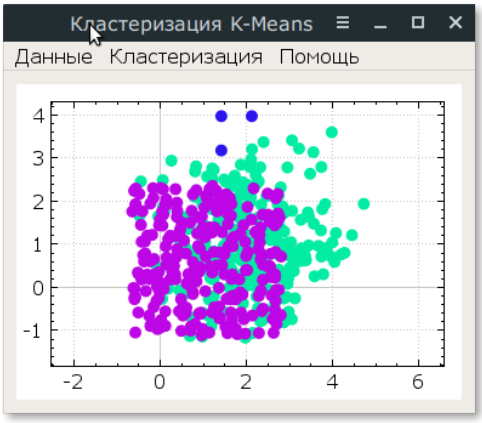

Рисунок 15 — Стандартизация по среднеквадратичной оценке каждого кластера

Применим Z-оценку (рис. 16).

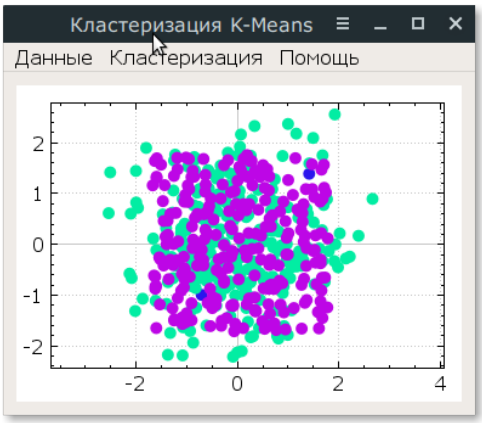

Рисунок 16 — Стандартизация по Z-оценке

Применим MinMax-оценку (рис. 17).

Рисунок 17 — Стандартизация по MinMax-оценке

Выберем пункт «Посмотреть внутренние данные» (рис. 18).

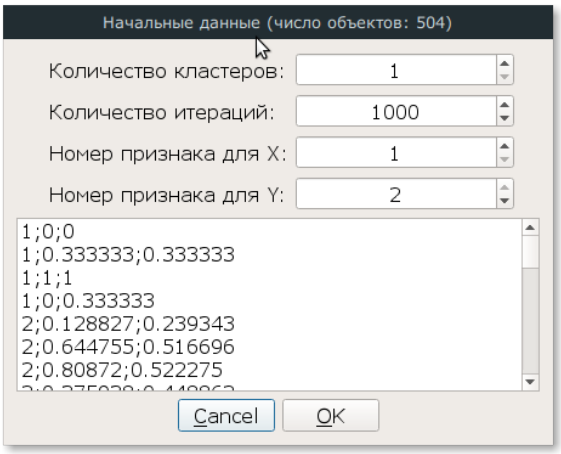

Рисунок 18 — Внутренние данные

Появилось окно со следующими элементами:

1. Настройка количества кластеров для кластеризации. Максимальное возможное число кластеров равно числу объектов. В данном случае количество кластеров равно 1 по умолчанию. Можно выставить 504, так как в списке имеется 504 объекта. По итогу кластеризации получим 504 кластера.

2. Количество итераций тоже относится к кластеризации. При использовании алгоритма k-means число итераций стоит делать достаточно большим, а при использовании k-means++ — относительно малым.

3. Номера признаков относятся к отображению данных на графике. Очевидно, что максимальное значение ограничивается числом признаков. Например, в нашем случае у точек есть только 2 признака, по оси X располагаются значения первого признака, а по оси Y второго. Можно выбрать один признак (например, первый) для осей X и Y, тогда все данные на графике будут выглядеть как одна линия (рис. 19).

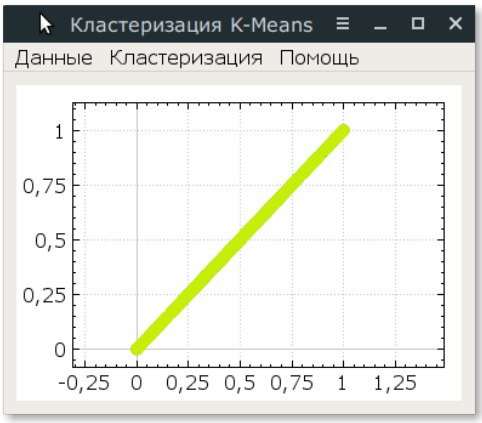

Рисунок 19 — Первый признак в качестве осей X и Y

Цвета кластеров изменяются при изменении номеров признаков случайно.

Также можно назначить первый признак в качестве оси Y, а второй в качестве оси X (рис. 20).

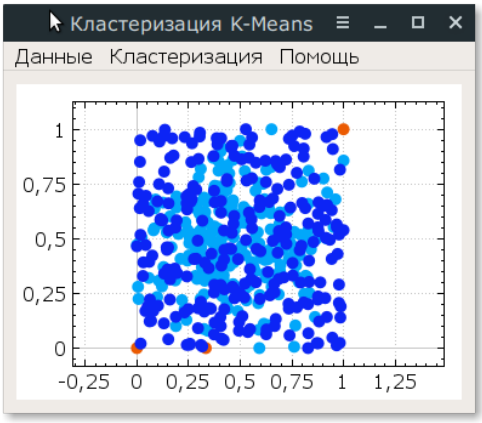

Рисунок 20 — Первый признак по Y, второй признак по X

Цвета также изменились случайным образом.

4. Внутренние данные находятся в неизменяемом списке. Первым идет номер кластера. На рисунке 18 видны только элементы кластеров номер 1 и номер 2, элементы кластера номер 3 можно увидеть ниже: для этого достаточно воспользоваться полосой прокрутки справа от списка. После номера кластера идут значения n-мерных точек.

Внутренние данные можно сохранить в любой формат. При сохранении появляется окно с вопросом: «Добавлять метки кластеров?». Если выбрать «Да», то каждый объект будет содержать в качестве своего первого признака метку кластера, а все остальные признаки будут идти после неё. Если выбрать «Нет», то в файл сохранится информация без меток кластеров. Выберем «Да» и получим следующую информацию в файле:

```
/ This file was generated by the ClusterData program
1;1;2
1;1.5;2.5
1;2.5;3.5
1;1;2.5
...
```

График можно сохранить в форматах PNG, JPG, PDF (рис. 21).

Рисунок 21 — Графики сохранены в три разных формата

Поведения функций объединения и удаления всех кластеров интуитивно понятны.

Теперь перейдем непосредственно к кластеризации. Посмотрим возможные действия с некоторыми данными (рис. 22).

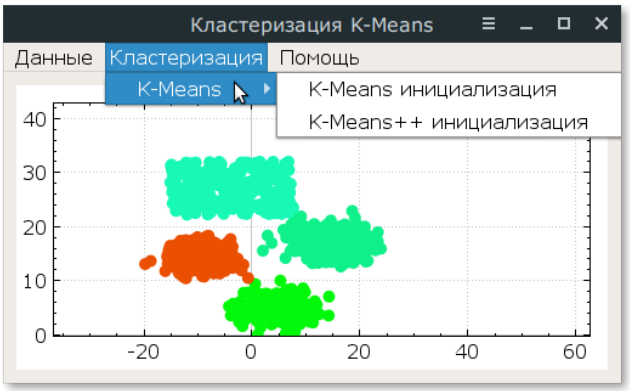

Рисунок 22 — Пункт меню "Кластеризация"

Пункт «K-Means инициализация» означает стандартный алгоритм k-means. А пункт «K-Means++ инициализация» более сходящийся алгоритм k-means++. Оба вида инициализации имеют место быть. Главное — помнить, что стандартный требует достаточное количество итераций, а k-means++ не требует. Выберем первый тип. Появится то же самое окно с настройками, что и при нажатии на «Посмотреть внутренние данные» (рис. 23).

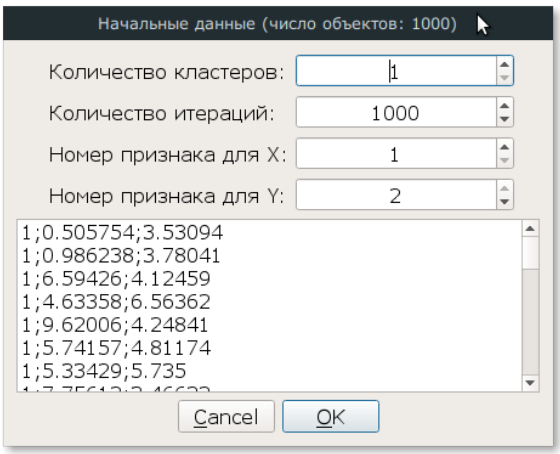

Рисунок 23 — Настройки кластеризации

Количество кластеров поменяем на 100 и посмотрим, каков будет результат (рис. 24).

Рисунок 24 — Результат кластеризации (100 кластеров)

Повернутые на 45 градусов квадраты обозначают центроиды кластеров. Как видим, цвета кластеров различны. При приближении видим 6 центроидов из 100 (рис. 25).

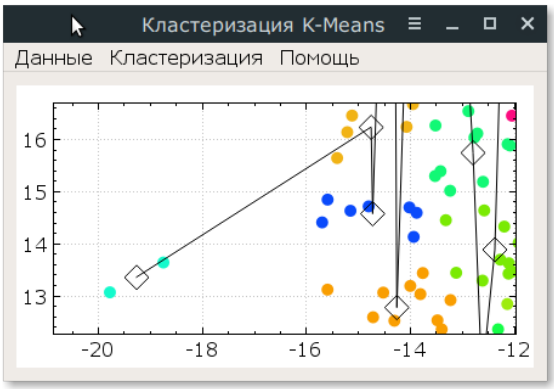

Рисунок 25 — 6 центроидов из 100 текущего набора данных

Некоторые кластеры похожи оттенком цвета. Однако если возникает потребность отличить один кластер от другого, достаточно щелкнуть по одному из представителей и все элементы кластера, к которому принадлежит этот представитель, будут выделены средне светлым оттенком пурпурно-синего цвета. Также появится соединяющая линия этого же цвета (рис. 26).

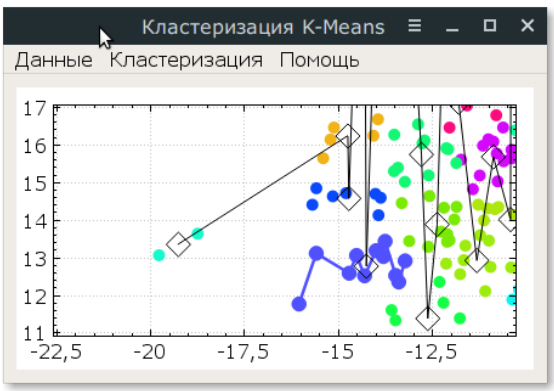

Рисунок 26 — Выделение кластера ведущей кнопкой мыши

Кластеризацию можно было проделать и для меньшего числа кластеров. Например, для восьми (рис. 27).

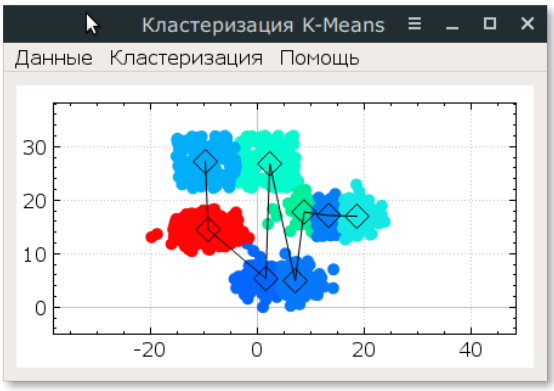

Рисунок 27 — Кластеризация на 8 кластеров

Все это время мы использовали стандартную инициализацию k-means. Теперь можно попробовать инициализацию k-means++. Повторно попробуем разбить на 8 кластеров предыдущее скопление точек (для этого необязательно объединять все кластеры, так как в программе перед кластеризацией все кластеры объединяются автоматически). Получим результат (рис. 28).

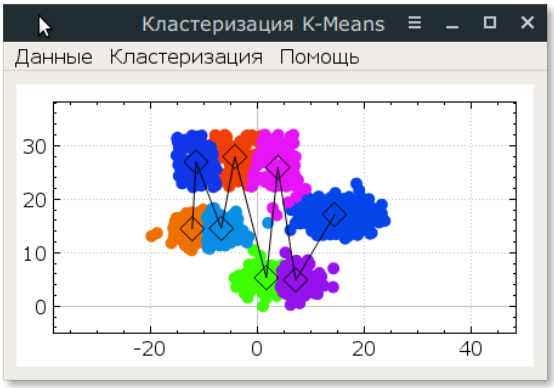

Рисунок 28 — Кластеризация k-means++

При каждом запуске можно получать разные результаты как при использовании k-means, так и при использовании k-means++.

В пункте «О программе» можно посмотреть авторство и используемые инструменты (рис. 29).

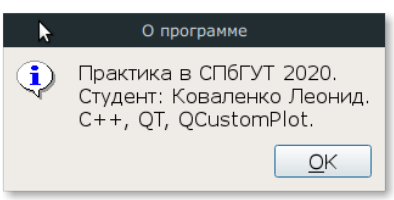

Рисунок 29 — О программе

В пункте «Справка» можно найти краткое описание программы (рис. 30).

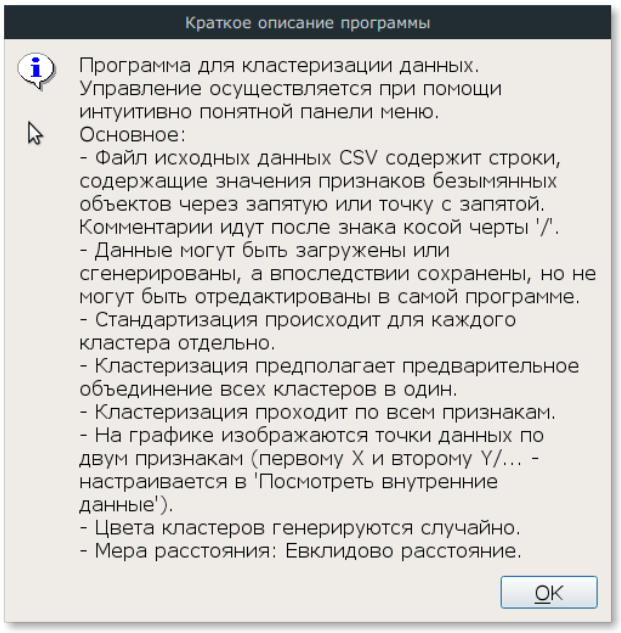

Рисунок 30 — Краткое описание программы

ЗАКЛЮЧЕНИЕ

В результате исследования был выявлен довольно широкий спектр возможностей кластерного анализа: разбиение объектов по целому набору параметров, отсутствие строгих ограничений на вид рассматриваемых объектов, возможность циклического приближения к необходимым исследователю результатам.

Алгоритмы k-means и k-means++ имеют одно важное достоинство — простота реализации и использования — и серьезные недостатки:

— Необходимо знать число кластеров заранее.

— Не гарантируется достижение глобального минимума. Один из способов исправить это — запустить алгоритм несколько раз и выбрать тот результат, который дает наилучшее значение целевой функции.

— Метод стремится к обнаружению гиперсферических кластеров.

— Результат сильно зависит от выбора исходных центров кластеров, их самый оптимальный выбор неизвестен, так как и стандартный k-means, и k-means++ являются приближениями к NP-трудной задаче оптимизации.

— Метод не учитывает плотность данных в кластере. Если есть очень плотная область, где размещается много данных, то среднее значение будет находиться ближе к этой плотной области.

В результате исследования была реализовано оконное ПО, код которого находится в приложениях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ковалев В.В. Теория статистики: учеб. пособие для бакалавров / В.В. Ковалев. — М.: Юрайт, 2014. — 454 с.

2. Шуметов В.Г. Кластерный анализ: подход с применением ЭВМ. Учебное пособие для вузов / В.Г. Шуметов, Л.В. Шуметова. — Орел: ОрелГТУ, 2001. — 119 с.

3. Гитис Л.Х. Статистическая классификация и кластерный анализ / Л.Х. Гитис. — М.: Издательство Московского государственного горного университета, 2003. — 157 с: ил.

4. Лекции по алгоритмам кластеризации и многомерного шкалирования / К.В. Воронцов. — 2007. URL: http://www.ccas.ru/voron/download/Clustering.pdf (31.08.2020).

5. Probabilistic Aspects in Cluster Analysis / H.H. Bock. — 1989. URL: https://doi.org/10.1007/978-3-642-75040-3_2 (31.08.2020).

6. Gan. Data Clustering: Theory, Algorithms, and Applications / Gan, Guojun, Chaoqun Ma, and Jianhong Wu. — 2007.

7. Дж.-О. Ким. Факторный, дискриминантный и кластерный анализ: Пер. с англ./Дж.-О. Ким, Ч. У. Мьюллер, У. Р. Клекка и др.; под ред. И. С. Енюкова. — М.: Финансы и статистика, 1989. — 215 с.: ил.

8. Котов А. Кластеризация данных / А. Котов, Н. Красильников — 2006.

9. Буреева Н.Н. Многомерный статистический анализ с использованием ППП "STATISTICA" / Н.Н. Буреева. — Нижний Новгород, 2007. URL: http:// www.unn.ru/pages/issues/aids/2007/57.pdf (31.08.2020).

10. Мандель И.Д. Кластерный анализ / И.Д. Мандель — М.: Финансы и статистика, 1988. — 176 с.: ил.

11. MVP: Model-View-Presenter. The Taligent Programming Model for C++ and Java / Mike Potel. — 1996. URL: http://www.wildcrest.com/Potel/Portfolio/ mvp.pdf (31.08.2020).

12. Метод k-средних / Loginom Company. URL: https://wiki.loginom.ru/articles/k-means.html (31.08.2020).

ПРИЛОЖЕНИЯ

В приложении приведен код основных модулей разработанного ПО, код программ, которые были использованы в процессе тестирования.

Модуль Point

Файл «point.h»

```
#ifndef POINT_H
#define POINT_H

#include <QVector>
#include <cmath>

// Класс "Точка"
class Point {
  public:
    // Конструктор по умолчанию
    Point();
    // Конструктор по числу признаков
    Point(int nAttributes, int value = 0);

    // Методы для получения значения первого признака
    double &front();
    const double &front() const;

    // Методы для получения значения последнего признака
    double &back();
    const double &back() const;

    // Методы для получения значения i-ого признака
    // (без проверки выхода за пределы в соответствии с подходом STL)
    double &operator[](int i);
    const double &operator[](int i) const;

    // Метод для получения числа признаков
    int size() const;
    // Метод для получения информации об отсутствии признаков
    bool empty() const;

    // Операторы сравнения "равно" и "не равно" двух точек
    bool operator==(const Point &arg) const;
    bool operator!=(const Point &arg) const;
```

```
    // Получение итераторов по признакам
    QVector<double>::iterator begin();
    QVector<double>::iterator end();
    QVector<double>::const_iterator begin() const;
    QVector<double>::const_iterator end() const;
    QVector<double>::const_iterator cbegin() const;
    QVector<double>::const_iterator cend() const;
    QVector<double>::reverse_iterator rbegin();
    QVector<double>::reverse_iterator rend();
    QVector<double>::const_reverse_iterator rbegin() const;
    QVector<double>::const_reverse_iterator rend() const;
    QVector<double>::const_reverse_iterator crbegin() const;
    QVector<double>::const_reverse_iterator crend() const;

    // Операции сложения, вычитания, умножения и деления точек
    Point operator+(const Point &arg) const;
    Point operator-() const; // Унарный минус: смена знака значений признаков
    Point operator-(const Point &arg) const;
    Point operator*(const Point &arg) const;
    Point operator/(const Point &arg) const;
    Point &operator+=(const Point &arg);
    Point &operator-=(const Point &arg);
    Point &operator*=(const Point &arg);
    Point &operator/=(const Point &arg);
    // Операции умножения и деления значений признаков на число d
    Point operator*(double d) const;
    Point operator/(double d) const;
    Point &operator*=(double d);
    Point &operator/=(double d);

    // Возведение в степень значений признаков
    Point pow(double arg) const;
    // Евклидово расстояние между двумя точками
    double getEuclidDistance(const Point &arg) const;

  private:
    // Вектор N признаков
    QVector<double> data_;
};
```

```
#endif // POINT_H
```

Файл «point.cpp»

```
#include <point.h>

// Конструктор по умолчанию
Point::Point() { data_.squeeze(); }

// Конструктор по числу признаков
Point::Point(int nAttributes, int value) : data_(nAttributes, value) {
    data_.squeeze(); // Освободить лишнюю зарезервированную память
}

// Методы для получения значения первого признака
double &Point::front() { return data_.front(); }
const double &Point::front() const { return data_.front(); }

// Методы для получения значения последнего признака
double &Point::back() { return data_.back(); }
const double &Point::back() const { return data_.back(); }

// Методы для получения значения i-ого признака
// (без проверки выхода за пределы в соответствии с подходом STL)
double &Point::operator[](int i) { return data_[i]; }
const double &Point::operator[](int i) const { return data_[i]; }

// Метод для получения числа признаков
int Point::size() const { return data_.size(); }
// Метод для получения информации об отсутствии признаков
bool Point::empty() const { return data_.empty(); }

// Операторы сравнения "равно" и "не равно" двух точек
bool Point::operator==(const Point &arg) const { return data_ == arg.data_; }
bool Point::operator!=(const Point &arg) const { return data_ != arg.data_; }

// Получение итераторов по признакам
QVector<double>::iterator Point::begin() { return data_.begin(); }
QVector<double>::iterator Point::end() { return data_.end(); }
QVector<double>::const_iterator Point::begin() const { return data_.begin(); }
QVector<double>::const_iterator Point::end() const { return data_.end(); }
QVector<double>::const_iterator Point::cbegin() const { return data_.cbegin(); }
QVector<double>::const_iterator Point::cend() const { return data_.cend(); }
```

```
QVector<double>::reverse_iterator Point::rbegin() { return data_.rbegin(); }
QVector<double>::reverse_iterator Point::rend() { return data_.rend(); }
QVector<double>::const_reverse_iterator Point::rbegin() const {
    return data_.rbegin();
}
QVector<double>::const_reverse_iterator Point::rend() const {
    return data_.rend();
}
QVector<double>::const_reverse_iterator Point::crbegin() const {
    return data_.crbegin();
}
QVector<double>::const_reverse_iterator Point::crend() const {
    return data_.crend();
}

// Операции сложения, вычитания, умножения и деления точек
Point Point::operator+(const Point &arg) const {
    Point r(*this);
    auto p1 = r.begin();
    for (auto p2 = arg.begin(); p1 != r.end(); ++p1, ++p2) {
        *p1 += *p2;
    }
    return r;
}
Point Point::operator-() const { // Унарный минус
    Point r(*this);
    for (auto &x : r) {
        x = -x;
    }
    return r;
}
Point Point::operator-(const Point &arg) const { return *this + (-arg); }
Point Point::operator*(const Point &arg) const {
    Point r(*this);
    auto p1 = r.begin();
    for (auto p2 = arg.begin(); p1 != r.end(); ++p1, ++p2) {
        *p1 *= *p2;
    }
    return r;
}
Point Point::operator/(const Point &arg) const {
```

```
    Point r(*this);
    auto p1 = r.begin();
    for (auto p2 = arg.begin(); p1 != r.end(); ++p1, ++p2) {
        *p1 /= *p2;
    }
    return r;
}
Point &Point::operator+=(const Point &arg) { return *this = (*this + arg); }
Point &Point::operator-=(const Point &arg) { return *this = (*this - arg); }
Point &Point::operator*=(const Point &arg) { return *this = (*this * arg); }
Point &Point::operator/=(const Point &arg) { return *this = (*this / arg); }

// Операции умножения и деления значений признаков на число d
Point Point::operator*(double d) const {
    Point r(*this);
    for (auto &x : r) {
        x *= d;
    }
    return r;
}
Point Point::operator/(double d) const {
    Point r(*this);
    for (auto &x : r) {
        x /= d;
    }
    return r;
}
Point &Point::operator*=(double d) { return *this = (*this * d); }
Point &Point::operator/=(double d) { return *this = (*this / d); }

// Возведение в степень значений признаков
Point Point::pow(double arg) const {
    Point r(*this);
    for (double &x : r) {
        x = std::pow(x, arg);
    }
    return r;
}

// Расстояние Евклида между двумя точками
double Point::getEuclidDistance(const Point &arg) const {
```

```
    double r = 0;
    auto p1 = begin();
    for (auto p2 = arg.begin(); p1 != end(); ++p1, ++p2) {
        r += std::pow(*p2 - *p1, 2.0);
    }
    return std::sqrt(r);
}
```

Модуль Cluster

Файл «cluster.h»

```
#ifndef CLUSTER_H
#define CLUSTER_H

#include "point.h"
#include <QVector>

// Класс "Кластер"
class Cluster {
  public:
    // Конструктор по умолчанию
    Cluster();
    // Конструктор по числу точек и числу признаков
    Cluster(int nPoints, int nAttributes, int value = 0);
    // Конструктор по вектору точек
    // (точки должны иметь одинаковое число признаков, иначе исключение
    // std::invalid_argument)
    Cluster(const QVector<Point> &);

    // Оператор приведения кластера к вектору точек
    operator QVector<Point>() const;

    // Методы для получения первой точки кластера
    Point &front();
    const Point &front() const;

    // Методы для получения последней точки кластера
    Point &back();
    const Point &back() const;

    // Методы для получения i-ой точки кластера
    // (без проверки выхода за пределы в соответствии с подходом STL)
```

```
    Point &operator[](int i);
    const Point &operator[](int i) const;

    // Метод для получения числа точек кластера
    int size() const;
    // Метод для получения информации об отсутствии точек в кластере
    bool empty() const;

    // Операторы сравнения "равно" и "не равно" двух кластеров
    bool operator==(const Cluster &) const;
    bool operator!=(const Cluster &) const;

    // Получение итераторов по точкам
    QVector<Point>::iterator begin();
    QVector<Point>::iterator end();
    QVector<Point>::const_iterator begin() const;
    QVector<Point>::const_iterator end() const;
    QVector<Point>::const_iterator cbegin() const;
    QVector<Point>::const_iterator cend() const;
    QVector<Point>::reverse_iterator rbegin();
    QVector<Point>::reverse_iterator rend();
    QVector<Point>::const_reverse_iterator rbegin() const;
    QVector<Point>::const_reverse_iterator rend() const;
    QVector<Point>::const_reverse_iterator crbegin() const;
    QVector<Point>::const_reverse_iterator crend() const;

    // Прямая сумма всех точек кластера
    Point sum() const;

  private:
    // Вектор M точек
    QVector<Point> data_;
};

#endif // CLUSTER_H
```

Файл «cluster.cpp»

```
#include <cluster.h>

// Конструктор по умолчанию
Cluster::Cluster() { data_.squeeze(); }
```

```cpp
// Конструктор по числу точек и числу признаков
Cluster::Cluster(int nPoints, int nAttributes, int value)
    : data_(nPoints, Point(nAttributes, value)) {
    data_.squeeze(); // Освободить лишнюю зарезервированную память
}

// Конструктор по вектору точек
// (точки должны иметь одинаковое число признаков, иначе исключение
// std::invalid_argument)
Cluster::Cluster(const QVector<Point> &arg) : data_(arg) {
    if (!data_.empty()) {
        auto nAttributes = data_.front().size();
        for (auto &point : data_) {
            if (point.size() != nAttributes) {
                throw std::invalid_argument(
                    "QVector points have a different number of attributes!");
            }
        }
        data_.squeeze();
    }
}

// Оператор приведения кластера к вектору точек
Cluster::operator QVector<Point>() const { return data_; }

// Методы для получения первой точки кластера
Point &Cluster::front() { return data_.front(); }
const Point &Cluster::front() const { return data_.front(); }

// Методы для получения последней точки кластера
Point &Cluster::back() { return data_.back(); }
const Point &Cluster::back() const { return data_.back(); }

// Методы для получения i-ой точки кластера
// (без проверки выхода за пределы в соответствии с подходом STL)
Point &Cluster::operator[](int i) { return data_[i]; }
const Point &Cluster::operator[](int i) const { return data_[i]; }

// Метод для получения числа точек кластера
int Cluster::size() const { return data_.size(); }
```

```
// Метод для получения информации об отсутствии точек в кластере
bool Cluster::empty() const { return data_.empty(); }

// Операторы сравнения "равно" и "не равно" двух кластеров
bool Cluster::operator==(const Cluster &arg) const {
    return data_ == arg.data_;
}
bool Cluster::operator!=(const Cluster &arg) const {
    return data_ != arg.data_;
}

// Получение итераторов по точкам
QVector<Point>::iterator Cluster::begin() { return data_.begin(); }
QVector<Point>::iterator Cluster::end() { return data_.end(); }
QVector<Point>::const_iterator Cluster::begin() const { return data_.begin(); }
QVector<Point>::const_iterator Cluster::end() const { return data_.end(); }
QVector<Point>::const_iterator Cluster::cbegin() const {
    return data_.cbegin();
}
QVector<Point>::const_iterator Cluster::cend() const { return data_.cend(); }
QVector<Point>::reverse_iterator Cluster::rbegin() { return data_.rbegin(); }
QVector<Point>::reverse_iterator Cluster::rend() { return data_.rend(); }
QVector<Point>::const_reverse_iterator Cluster::rbegin() const {
    return data_.rbegin();
}
QVector<Point>::const_reverse_iterator Cluster::rend() const {
    return data_.rend();
}
QVector<Point>::const_reverse_iterator Cluster::crbegin() const {
    return data_.crbegin();
}
QVector<Point>::const_reverse_iterator Cluster::crend() const {
    return data_.crend();
}

// Прямая сумма всех точек кластера
Point Cluster::sum() const {
    if (empty())
        return Point();
    Point r(data_.first().size());
    for (auto &point : data_) {
```

```
        r += point;
    }
    return r;
}
```

Модуль KMeans

Файл «kmeans.h»

```
#ifndef KMEANS_H
#define KMEANS_H

#include "cluster.h"
#include <QPair>
#include <QVector>
#include <random>

// Класс "K-Means алгоритм"
class KMeansPlusPlus {
  public:
    // Конструктор по числу кластеров, числу атрибутов и числу итераций
    KMeansPlusPlus(int nClusters, int nAttributes, int nIterations);

    // Кластеризация (возвращает вектор кластеров и вектор центроидов)
    QPair<QVector<Cluster>, QVector<Point>> compute(const Cluster &,
                                                    bool) const;

  protected:
    // Расстояние между двумя точками
    qreal calcDistance(const Point &, const Point &) const;

    // Точка со средними значениями признаков по всем точкам кластера
    Point calcMean(const Cluster &) const;

  private:
    // Число кластеров
    int nClusters_;
    // Число атрибутов
    int nAttributes_;
    // Число итераций
    int nIterations_;

    // Недетерминированный генератор случайных чисел с использованием
```

```
    // аппаратного источника энтропии
    mutable std::random_device rd;
};

#endif // KMEANS_H
```

Файл «kmeans.cpp»

```
#include "kmeans.h"

// Конструктор по числу кластеров, числу атрибутов и числу итераций
KMeansPlusPlus::KMeansPlusPlus(int nClusters, int nAttributes, int nIterations)
    : nClusters_(nClusters), nAttributes_(nAttributes),
      nIterations_(nIterations) {}

// Кластеризация (возвращает вектор кластеров и вектор центроидов)
QPair<QVector<Cluster>, QVector<Point>>
KMeansPlusPlus::compute(const Cluster &data, bool isPlusPlus) const {
    if (data.size() == 0)
        return {QVector<Cluster>(nClusters_), QVector<Point>()};
    // Вектор точек-центроидов
    QVector<Point> centroids(nClusters_, Point(nAttributes_));
    if (isPlusPlus) {
        // Инициализация K-Means++
        // 1. Выбираем в качестве первого центроида любую из точек входных
        // данных
        std::uniform_int_distribution<int> distN(0, data.size() - 1);
        centroids[0] = data[distN(rd)];
        // 2. Находим остальные центроиды
        qreal minDistanceFrom0 = calcDistance(data[0], centroids[0]);
        for (int i = 1; i < nClusters_; ++i) {
            // 2.1. Находим начальное значение для максимального расстояния
            // Находим расстояние от точки до ближайшего найденного центроида
            minDistanceFrom0 = std::min(
                minDistanceFrom0, calcDistance(data[0], centroids[i - 1]));
            int idx = 0; // Индекс максимально отдаленной точки
            // Расстояние до максимально отдаленной точки
            qreal maxDistance =
                minDistanceFrom0; // Начальное значение установлено
            // Теперь maxDistance можно сравнивать с остальными минимальными
            // значениями
            // 2.2. Находим maxDistance - максимальное из минимальных расстояний
            for (int j = 1, len = data.size(); j < len; ++j) {
```

```
            // Находим расстояние от точки до ближайшего найденного
            // центроида
            qreal minDistance = calcDistance(data[j], centroids[0]);
            for (int k = 1; k < i; ++k) {
                qreal distance = calcDistance(data[j], centroids[k]);
                if (minDistance > distance) {
                    minDistance = distance;
                }
            }
            if (maxDistance < minDistance) {
                maxDistance = minDistance;
                // j-ая точка -- наиболее удаленная точка
                idx = j;
            }
        }
        centroids[i] = data[idx]; // i-й центроид найден!
    }
} else {
    // Инициализация K-Means
    QVector<int> numbers(data.size());
    std::iota(numbers.begin(), numbers.end(), 0);
    std::shuffle(numbers.begin(), numbers.end(), rd);
    for (int i = 0; i < nClusters_; ++i) {
        centroids[i] = data[numbers[i]];
    }
}
// Основной алгоритм K-Means
// 1. Последовательно кластеризуем данные
QVector<QVector<Point>> clusters(nClusters_);
for (int iter = 0; iter < nIterations_; ++iter) {
    // Находим минимальную разницу для каждой точки и каждого кластера
    for (auto point : data) {
        qreal minDistance = calcDistance(point, centroids[0]);
        int clusterIdx = 0;
        for (int i = 1; i < nClusters_; ++i) {
            qreal currentDistance = calcDistance(point, centroids[i]);
            if (minDistance > currentDistance) {
                minDistance = currentDistance;
                clusterIdx = i;
            }
        }
```

```
            // Добавляем текущую точку к наиболее близкому кластеру
            clusters[clusterIdx].append(point);
        }
        // Вычисляем новые положения центроидов
        for (int i = 0; i < nClusters_; ++i) {
            centroids[i] = calcMean(clusters[i]);
            clusters[i].clear(); // Очищаем элементы кластера
        }
    }
    // Выполняем последний обход точек для определения минимальной разницы
    for (auto point : data) {
        qreal minDistance = calcDistance(point, centroids[0]);
        int clusterIdx = 0;
        for (int i = 1; i < nClusters_; ++i) {
            qreal currentDistance = calcDistance(point, centroids[i]);
            if (minDistance > currentDistance) {
                minDistance = currentDistance;
                clusterIdx = i;
            }
        }
        // Добавляем текущую точку к наиболее близкому кластеру
        clusters[clusterIdx].append(point);
    }
    // Вычисляем новые положения центроидов, не очищая элементы clusters
    for (int i = 0; i < nClusters_; ++i) {
        centroids[i] = calcMean(clusters[i]);
    }
    // Преобразование QVector<QVector<Point>> в QVector<Cluster>
    QVector<Cluster> result(nClusters_);
    auto it = result.begin();
    for (auto cluster : clusters) {
        *(it++) = std::move(cluster);
    }
    return {result, centroids};
}

// [private] Расстояние между двумя точками
qreal KMeansPlusPlus::calcDistance(const Point &point,
                                   const Point &mean) const {
    return point.getEuclidDistance(mean);
}
```

```
// [private] Точка со средними значениями признаков по всем точкам кластера
Point KMeansPlusPlus::calcMean(const Cluster &cluster) const {
    return cluster.sum() / cluster.size();
}
```

Модуль PlotModel

Файл «plotmodel.h»

```
#ifndef PLOTMODEL_H
#define PLOTMODEL_H

#include "cluster.h"
#include <QVector>

// Класс "Модель"
class PlotModel {
  public:
    // Добавление кластеров
    // Добавление нового кластера в i-ую позицию
    void addCluster(int, const Cluster &);
    // Добавление нового кластера в конец
    void addCluster(const Cluster &);
    // Получение числа кластеров
    int countOfClusters() const;
    // Получение числа признаков
    int countOfAttributes() const;

    // Удаление кластеров
    // Удаление i-ого кластера
    void removeCluster(int);
    // Удаление последнего кластера
    void removeCluster();
    // Удаление всех кластеров
    void removeAll();

    // Получение данных
    const QVector<Cluster> &data() const;

  private:
    // Список из кластеров точек
    QVector<Cluster> data_;
```

```
};

#endif // PLOTMODEL_H
```

Файл «plotmodel.cpp»

```
#include "plotmodel.h"

// Добавление нового кластера в i-ую позицию
void PlotModel::addCluster(int i, const Cluster &arg) {
    if (i >= 0 && i <= data_.size())
        data_.insert(i, arg);
}

// Добавление нового кластера в конец
void PlotModel::addCluster(const Cluster &arg) { data_.append(arg); }

// Получение числа кластеров
int PlotModel::countOfClusters() const { return data_.size(); }

// Получение числа признаков
int PlotModel::countOfAttributes() const {
    return (!data_.empty()) ? data_.front().front().size() : 0;
}

// Удаление i-ого кластера
void PlotModel::removeCluster(int i) {
    if (i >= 0 && i < data_.size())
        data_.remove(i);
}

// Удаление последнего кластера
void PlotModel::removeCluster() {
    if (!data_.empty())
        data_.pop_back();
}

// Удаление всех кластеров
void PlotModel::removeAll() { data_.clear(); }

// Получение данных
const QVector<Cluster> &PlotModel::data() const { return data_; }
```

Модуль PlotPresenter

Файл «plotpresenter.h»

```
#ifndef PLOTPRESENTER_H
#define PLOTPRESENTER_H

#include "kmeans.h"
#include "plotmodel.h"
#include "point.h"
#include "qcustomplot.h"

// Класс "Представитель"
class PlotPresenter {
  public:
    // Конструктор
    PlotPresenter(PlotModel &, QCustomPlot &);

    // Добавление кластеров
    // Добавление нового кластера в i-ую позицию
    void addCluster(int, const Cluster &, int, int);
    // Добавление нового кластера в конец
    void addCluster(const Cluster &, int, int);
    // Объединение всех кластеров
    void combineAll(int nX, int nY);
    // Получение числа кластеров
    int countOfClusters() const;
    // Получение числа признаков
    int countOfAttributes() const;

    // Удаление кластеров
    // Удаление i-ого кластера
    void removeCluster(int);
    // Удаление последнего кластера
    void removeCluster();
    // Удаление всех кластеров
    void clearAll();

    // Кластеризация
    QPair<QVector<Cluster>, QVector<Point>> clusterData(int, int, bool) const;

    // Добавление кластера центроидов (не имеет значения для модели)
```

```
    void addCentroids(const QVector<Point> &, int, int);
    // Удаление кластера центроидов (не имеет значения для модели)
    void removeCentroids();

    // Настройки отображения
    // Установка цвета фона графика
    void setBackgroundColor(const QBrush &);
    // Установка стиля
    void setStyle(int, const QBrush &, int size = 10);
    // Раскраска всех кластеров в разные цвета
    void
    colorizeAll(const QVector<QPair<QPen, QCPScatterStyle>> *colors = nullptr);

    // Сохранение графика в файл
    // Сохранение графика в файл PDF
    bool savePdf(const QString &) const;
    // Сохранение графика в файл PNG
    bool savePng(const QString &) const;
    // Сохранение графика в файл JPG
    bool saveJpg(const QString &) const;

    // Получение данных
    const QVector<Cluster> &data() const;
    // Установка данных
    void setData(const QVector<Cluster> &, int, int);

    // Стандартизация
    // Стандартизация по Z-оценке
    void zStd(int, int);
    // Стандартизация по среднеквадратическому отклонению
    void sigmaStd(int, int);
    // Стандартизация по минимальному и максимальному значениям
    void minMaxStd(int, int);

    // Генерация случайного целого числа [a, b]
    int randInt(int, int);

protected:
    // Диапазоны цветов HSV для кластеров
    // Hue, Saturation, Value - Тон, Насыщенность, Значение (Яркость)
    const int Hmin = 0, Hmax = 359, Smin = 230, Smax = 255, Vmin = 230,
```

```
            Vmax = 255;

    // Размер центроида
    int centroidSize = 20;
    // Функция, возвращающая вектор стилей каждого кластера
    QVector<QPair<QPen, QCPScatterStyle>> getClustersColors();

  private:
    // Модель
    PlotModel *plotModel_ = nullptr;
    // Вид
    QCustomPlot *customPlot_ = nullptr;
    // Есть ли центроиды кластеров на графике или нет
    bool hasCentroids_ = false;
};

#endif // PLOTPRESENTER_H
```

Файл «plotpresenter.cpp»

```
#include "plotpresenter.h"

// Конструктор
PlotPresenter::PlotPresenter(PlotModel &pm, QCustomPlot &qcp)
    : plotModel_(&pm), customPlot_(&qcp) {}

// Добавление нового кластера в i-ую позицию
void PlotPresenter::addCluster(int i, const Cluster &arg, int nX, int nY) {
    int nClusters = plotModel_->countOfClusters();
    int nAttributes = plotModel_->countOfAttributes();
    if (nClusters == 0 ||
        (nClusters != 0 && nAttributes == arg.front().size())) {
        removeCentroids();
        plotModel_->addCluster(i, arg);
        customPlot_->addGraph();
        QVector<qreal> x, y;
        for (auto &attr : arg) {
            x.append(attr[nX]);
            y.append(attr[nY]);
        }
        customPlot_->graph()->addData(x, y);
        customPlot_->replot();
    } else {
```

```
        QMessageBox::critical(customPlot_, "Ошибка",
                              "Число признаков текущего объекта: " +
                                  QString::number(arg.size()) +
                                  "\nНеобходимое число признаков: " +
                                  QString::number(nAttributes));
    }
}

// Добавление нового кластера в конец
void PlotPresenter::addCluster(const Cluster &arg, int nX, int nY) {
    addCluster(plotModel_->countOfClusters(), arg, nX, nY);
}

// Объединение всех кластеров
void PlotPresenter::combineAll(int nX, int nY) {
    QVector<Point> points;
    for (auto cluster : data()) {
        for (auto point : cluster) {
            points.append(point);
        }
    }
    clearAll();
    addCluster(Cluster(points), nX, nY);
    colorizeAll();
}

// Получение числа кластеров
int PlotPresenter::countOfClusters() const {
    return plotModel_->countOfClusters();
}

// Получение числа признаков
int PlotPresenter::countOfAttributes() const {
    return plotModel_->countOfAttributes();
}

// Удаление i-ого кластера
void PlotPresenter::removeCluster(int i) {
    if (plotModel_->countOfClusters() != 0) {
        removeCentroids();
        plotModel_->removeCluster(i);
```

```
        customPlot_->removeGraph(i);
        customPlot_->replot();
    } else {
        QMessageBox::critical(
            customPlot_, "Ошибка",
            QString::number(i) +
                " кластер не существует. Удаление невозможно.");
    }
}

// Удаление последнего кластера
void PlotPresenter::removeCluster() {
    removeCluster(plotModel_->countOfClusters() - 1);
}

// Удаление всех кластеров
void PlotPresenter::clearAll() {
    removeCentroids();
    plotModel_->removeAll();
    customPlot_->clearGraphs();
    customPlot_->replot();
}

// Кластеризация
QPair<QVector<Cluster>, QVector<Point>>
PlotPresenter::clusterData(int nClusters, int nIterations,
                           bool isPlusPlus) const {
    if (nClusters == 0 || nIterations == 0)
        return QPair<QVector<Cluster>, QVector<Point>>();
    QVector<Point> points;
    for (auto cluster : data()) {
        for (auto point : cluster) {
            points.append(point);
        }
    }
    if (points.size() == 0) {
        QMessageBox::critical(customPlot_, "Ошибка", "Число точек равно нулю");
        return QPair<QVector<Cluster>, QVector<Point>>();
    }
    if (points.size() < nClusters) {
        QMessageBox::critical(customPlot_, "Ошибка",
```

```
                                "Число объектов должно быть больше или равно "
                                "числу кластеров!\nЧисло объектов: " +
                                    QString::number(points.size()) +
                                    "\nЗапрашиваемое число кластеров: " +
                                    QString::number(nClusters));
        return QPair<QVector<Cluster>, QVector<Point>>();
    }
    KMeansPlusPlus km(nClusters, this->countOfAttributes(), nIterations);
    return km.compute(points, isPlusPlus);
}

// Добавление кластера центроидов (не имеет значения для модели)
void PlotPresenter::addCentroids(const QVector<Point> &data, int attr1,
                                 int attr2) {
    hasCentroids_ = true;
    customPlot_->addGraph();
    customPlot_->graph()->setPen(QPen(QBrush(Qt::black), 0, Qt::SolidLine));
    customPlot_->graph()->setScatterStyle(
        QCPScatterStyle(QCPScatterStyle::ssDiamond, centroidSize));
    for (auto point : data) {
        customPlot_->graph()->addData(point[attr1], point[attr2]);
    }
    customPlot_->replot();
}

// Удаление кластера центроидов (не имеет значения для модели)
void PlotPresenter::removeCentroids() {
    if (hasCentroids_) {
        customPlot_->removeGraph(customPlot_->graphCount() - 1);
        hasCentroids_ = false;
    }
}

// Установка цвета фона графика
void PlotPresenter::setBackgroundColor(const QBrush &arg) {
    customPlot_->setBackground(arg);
    customPlot_->replot();
}

// Установка стиля
void PlotPresenter::setStyle(int i, const QBrush &brush, int size) {
```

```
    customPlot_->graph(i)->setPen(QPen(brush, 0, Qt::NoPen));
    customPlot_->graph(i)->setScatterStyle(
        QCPScatterStyle(QCPScatterStyle::ssDisc, size));
    customPlot_->replot();
}

// Раскраска всех кластеров в разные цвета
void PlotPresenter::colorizeAll(
    const QVector<QPair<QPen, QCPScatterStyle>> *colors) {
    if (colors) {
        for (int i = 0, len = data().size(); i < len; ++i) {
            customPlot_->graph(i)->setPen((*colors)[i].first);
            customPlot_->graph(i)->setScatterStyle((*colors)[i].second);
        }
    } else {
        for (int i = 0, len = data().size(); i < len; ++i) {
            setStyle(i, QBrush(QColor().fromHsv(randInt(Hmin, Hmax),
                                                randInt(Smin, Smax),
                                                randInt(Vmin, Vmax))));
        }
    }
    customPlot_->replot();
}

// Сохранение графика в файл PDF
bool PlotPresenter::savePdf(const QString &filename) const {
    return customPlot_->savePdf(filename);
}

// Сохранение графика в файл PNG
bool PlotPresenter::savePng(const QString &filename) const {
    return customPlot_->savePng(filename);
}

// Сохранение графика в файл JPG
bool PlotPresenter::saveJpg(const QString &filename) const {
    return customPlot_->saveJpg(filename);
}

// Получение данных
const QVector<Cluster> &PlotPresenter::data() const {
```

```
    return plotModel_->data();
}

// Установка данных
void PlotPresenter::setData(const QVector<Cluster> &newData, int nX, int nY) {
    if (&newData != &data()) {
        clearAll();
        for (auto cluster : newData) {
            addCluster(cluster, nX, nY);
        }
    } else {
        setData(QVector<Cluster>(data()), nX, nY);
    }
}

// Стандартизация по Z-оценке
void PlotPresenter::zStd(int nX, int nY) {
    removeCentroids();
    auto clustersColors = getClustersColors();
    QVector<Cluster> newData(data());
    for (auto &cluster : newData) {
        int nAttributes = plotModel_->countOfAttributes();
        Point mean(nAttributes), sigma(nAttributes);
        for (auto point : cluster) {
            mean += point;
        }
        mean /= cluster.size();
        for (auto point : cluster) {
            sigma += (point - mean).pow(2);
        }
        sigma = (sigma / (cluster.size() - (cluster.size() != 1))).pow(0.5);
        for (auto attr : sigma) {
            if (attr == 0) {
                QMessageBox::critical(customPlot_, "Внимание!",
                                      "Размах значений одного из "
                                      "признаков близок к нулю, поэтому "
                                      "стандартизация выполнена не будет.");
                return;
            }
        }
        for (auto &point : cluster) {
```

```
            point = (point - mean) / sigma;
        }
    }
    setData(newData, nX, nY);
    colorizeAll(&clustersColors);
    customPlot_->replot();
}

// Стандартизация по среднеквадратическому отклонению
void PlotPresenter::sigmaStd(int nX, int nY) {
    removeCentroids();
    auto clustersColors = getClustersColors();
    QVector<Cluster> newData(data());
    for (auto &cluster : newData) {
        int nAttributes = plotModel_->countOfAttributes();
        Point mean(nAttributes), sigma(nAttributes);
        for (auto point : cluster) {
            mean += point;
        }
        mean /= cluster.size();
        for (auto point : cluster) {
            sigma += (point - mean).pow(2);
        }
        sigma = (sigma / (cluster.size() - (cluster.size() != 1))).pow(0.5);
        for (auto attr : sigma) {
            if (attr == 0) {
                QMessageBox::critical(customPlot_, "Внимание!",
                                      "Размах значений одного из "
                                      "признаков близок к нулю, поэтому "
                                      "стандартизация выполнена не будет.");
                return;
            }
        }
        for (auto &point : cluster) {
            point /= sigma;
        }
    }
    setData(newData, nX, nY);
    colorizeAll(&clustersColors);
    customPlot_->replot();
}
```

```
// Стандартизация по минимальному и максимальному значениям
void PlotPresenter::minMaxStd(int nX, int nY) {
    removeCentroids();
    auto clustersColors = getClustersColors();
    QVector<Cluster> newData(data());
    for (auto &cluster : newData) {
        int nAttributes = plotModel_->countOfAttributes();
        Point minPoint(nAttributes), maxPoint(nAttributes);
        for (auto point : cluster) {
            for (int attr = 0; attr < nAttributes; ++attr) {
                auto pointAttr = point[attr];
                minPoint[attr] = std::min(minPoint[attr], pointAttr);
                maxPoint[attr] = std::max(maxPoint[attr], pointAttr);
            }
        }
        Point minMaxPoint(maxPoint - minPoint);
        for (auto attr : minMaxPoint) {
            if (attr == 0) {
                QMessageBox::critical(customPlot_, "Внимание!",
                                      "Размах значений одного из "
                                      "признаков близок к нулю, поэтому "
                                      "стандартизация выполнена не будет.");
                return;
            }
        }
        for (auto &point : cluster) {
            point = (point - minPoint) / minMaxPoint;
        }
    }
    setData(newData, nX, nY);
    colorizeAll(&clustersColors);
    customPlot_->replot();
}

// [protected] Функция, возвращающая вектор стилей каждого кластера
QVector<QPair<QPen, QCPScatterStyle>> PlotPresenter::getClustersColors() {
    QVector<QPair<QPen, QCPScatterStyle>> clustersColors(data().size());
    for (int i = 0, len = data().size(); i < len; ++i) {
        clustersColors[i] =
            QPair<QPen, QCPScatterStyle>(customPlot_->graph(i)->pen(),
```

```
                                        customPlot_->graph(i)->scatterStyle());
    }
    return clustersColors;
}

// [protected] Генерация случайного целого числа [a, b]
int PlotPresenter::randInt(int a, int b) { return qrand() % (b - a + 1) + a; }
```

Модуль MainWindow

Файл «mainwindow.h»

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "cluster.h"
#include "gendialog.h"
#include "plotmodel.h"
#include "plotpresenter.h"
#include "rundialog.h"
#include <QMainWindow>
#include <QVector>
#include <random>

namespace Ui {
    class MainWindow;
}

// Класс "Главное окно"
class MainWindow : public QMainWindow {
    Q_OBJECT

  public:
    // Конструктор
    explicit MainWindow(QWidget *parent = 0);
    // Деструктор
    ~MainWindow();

  protected:
    // Диапазоны цветов HSV для кластеров
    // Hue, Saturation, Value - Тон, Насыщенность, Значение (Яркость)
    const int Hmin = 0, Hmax = 359, Smin = 230, Smax = 255, Vmin = 230,
              Vmax = 255;
```

```
    // Событие изменения размера MainWindow
    virtual void resizeEvent(QResizeEvent *);

    // Генерация случайного целого числа [a, b]
    int randInt(int, int);

private slots:

    // Генерация исходных данных
    void on_genData_triggered();

    // Сохранение графика в PNG формате
    void on_saveAsPNG_triggered();

    // Сохранение графика в JPG формате
    void on_saveAsJPG_triggered();

    // Сохранение графика в PDF формате
    void on_saveAsPDF_triggered();

    // Выход
    void on_quit_triggered();

    // Кластеризация данных K-Means
    void on_clusterDataRnd_triggered(bool isPlusPlus);

    // Кластеризация данных K-Means++
    void on_clusterDataPlusPlus_triggered();

    // О программе
    void on_about_triggered();

    // Краткое описание программы
    void on_description_triggered();

    // Открытие файла с данными
    void on_loadData_triggered();

    // Сохранение внутренних данных в файл
    void on_saveData_triggered();
```

```
    // Отображение внутренних данных
    bool on_showData_triggered(bool bNoWarning);

    // Стандартизация по Z-оценке
    void on_stdZ_triggered();

    // Стандартизация по среднеквадратическому отклонению
    void on_stdSigma_triggered();

    // Стандартизация по минимальному и максимальному значениям
    void on_stdMinMax_triggered();

    // Объединение всех кластеров
    void on_combineData_triggered();

    // Очистка всех данных
    void on_clearData_triggered();

  private:
    RunDialog runDialog; // Диалоговое окно просмотра данных
    GenDialog genDialog; // Диалоговое окно генерации данных
    Ui::MainWindow *ui;           // Интерфейс окна
    PlotModel plotModel;          // Модель
    PlotPresenter *plotPresenter; // Представитель
    // Число кластеров, число итераций, номер признака для X координаты, номер
    // признака для Y координаты
    int nClusters_ = 0, nIterations_ = 1000, nX_ = 0, nY_ = 1;
    // Недетерминированный генератор случайных чисел с использованием
    // аппаратного источника энтропии
    mutable std::random_device rd;
};

#endif // MAINWINDOW_H
```

Файл «mainwindow.cpp»

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

// Конструктор
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::MainWindow) {
```

```
    ui->setupUi(this);
    // Определение представителя
    QCustomPlot *customPlot = ui->graphWidget;
    plotPresenter = new PlotPresenter(plotModel, *customPlot);
    // Включение режима взаимодействия с графиком
    customPlot->setInteractions(QCP::iRangeDrag | QCP::iRangeZoom |
                                QCP::iSelectPlottables | QCP::iMultiSelect);
    // Отображение делений правой и верхней оси
    customPlot->xAxis2->setVisible(true);
    customPlot->yAxis2->setVisible(true);
    // Удаление числовых меток правой и верхней оси
    customPlot->xAxis2->setTickLabels(false);
    customPlot->yAxis2->setTickLabels(false);
    // Передача диапазонов левой и нижней оси на правую и верхнюю оси
    connect(customPlot->xAxis, SIGNAL(rangeChanged(QCPRange)),
            customPlot->xAxis2, SLOT(setRange(QCPRange)));
    connect(customPlot->yAxis, SIGNAL(rangeChanged(QCPRange)),
            customPlot->yAxis2, SLOT(setRange(QCPRange)));
    // Установка фона графика в белый
    customPlot->setBackground(QBrush(QColor(Qt::white)));
    // Установка начальных диапазонов
    customPlot->xAxis->setRangeLower(0);
    customPlot->xAxis->setRangeUpper(10);
    customPlot->yAxis->setRangeLower(0);
    customPlot->yAxis->setRangeUpper(10);
    // Перерисовка графика
    customPlot->replot();
    // Переустановка глобального генератора случайных чисел
    qsrand(QTime().secsTo(QTime::currentTime()));
}

// Генерация случайного целого числа [a, b]
int MainWindow::randInt(int a, int b) { return plotPresenter->randInt(a, b); }

// Событие изменения размера MainWindow
void MainWindow::resizeEvent(QResizeEvent *) {
    // Пропорциональное отображение данных (убирает растяжение графика)
    QCPAxis *x, *y;
    x = ui->graphWidget->axisRect()->axis(QCPAxis::atBottom);
    y = ui->graphWidget->axisRect()->axis(QCPAxis::atLeft);
    x->setScaleRatio(y, 1.0);
```

```
    ui->graphWidget->replot();
}

// Деструктор
MainWindow::~MainWindow() { delete ui; }

// Генерация исходных данных
void MainWindow::on_genData_triggered() {
    // Запускаем диалоговое окно
    QCPRange rx = ui->graphWidget->xAxis->range(),
             ry = ui->graphWidget->yAxis->range();
    genDialog.updateInfo(genDialog.getNumberOfPoints(), genDialog.getKind(),
                         rx.lower, rx.upper, ry.lower, ry.upper);
    if (genDialog.exec() != GenDialog::Accepted) {
        return;
    }
    int nPoints = genDialog.getNumberOfPoints();
    if (nPoints == 0)
        return;
    int isCircle = (genDialog.getKind() == 0);
    QPair<qreal, qreal> rangeX = genDialog.getRangeX(),
                        rangeY = genDialog.getRangeY();
    if (rangeX.first > rangeX.second || rangeY.first > rangeY.second) {
        QMessageBox::critical(this, "Ошибка!",
                              "Нижние границы должны быть не больше верхних!");
        return;
    }
    // Добавляем кластер с точками
    Cluster cluster(nPoints, /* nAttributes */ 2);
    if (isCircle) {
        double xMean = rangeX.first + (rangeX.second - rangeX.first) / 2,
               yMean = rangeY.first + (rangeY.second - rangeY.first) / 2;
        // 6 = 3 (правило трех сигм) * 2 (половина от диапазона)
        double xSigma = (rangeX.second - rangeX.first) / 6,
               ySigma = (rangeY.second - rangeY.first) / 6;
        std::normal_distribution<qreal> distX(xMean, xSigma),
            distY(yMean, ySigma);
        for (int i = 0; i < nPoints; ++i) {
            cluster[i][/* id X */ 0] = distX(rd);
            cluster[i][/* id Y */ 1] = distY(rd);
            // Выход за диапазон нежелателен, поэтому ставим ограничения
```

```
            if (cluster[i][0] < rangeX.first) {
                cluster[i][0] = rangeX.first;
            }
            if (cluster[i][0] > rangeX.second) {
                cluster[i][0] = rangeX.second;
            }
            if (cluster[i][1] < rangeY.first) {
                cluster[i][1] = rangeY.first;
            }
            if (cluster[i][1] > rangeY.second) {
                cluster[i][1] = rangeY.second;
            }
        }
    } else {
        std::uniform_real_distribution<qreal> distX(rangeX.first,
                                                    rangeX.second),
            distY(rangeY.first, rangeY.second);
        for (int i = 0; i < nPoints; ++i) {
            cluster[i][/* id X */ 0] = distX(rd);
            cluster[i][/* id Y */ 1] = distY(rd);
        }
    }
    plotPresenter->addCluster(cluster, nX_, nY_);
    plotPresenter->setStyle(
        plotPresenter->countOfClusters() - 1,
        QBrush(QColor().fromHsv(randInt(Hmin, Hmax), randInt(Smin, Smax),
                                randInt(Vmin, Vmax))));
}

// Сохранение графика в PNG формате
void MainWindow::on_saveAsPNG_triggered() {
    plotPresenter->savePng(QFileDialog::getSaveFileName(
        this, "Сохранить как...", ".", "PNG files (*.png)"));
}

// Сохранение графика в JPG формате
void MainWindow::on_saveAsJPG_triggered() {
    plotPresenter->saveJpg(QFileDialog::getSaveFileName(
        this, "Сохранить как...", ".", "JPG files (*.jpg)"));
}
```

```
// Сохранение графика в PDF формате
void MainWindow::on_saveAsPDF_triggered() {
    plotPresenter->savePdf(QFileDialog::getSaveFileName(
        this, "Сохранить как...", ".", "PDF files (*.pdf)"));
}

// Выход
void MainWindow::on_quit_triggered() { QCoreApplication::quit(); }

// Кластеризация данных
void MainWindow::on_clusterDataRnd_triggered(bool isPlusPlus) {
    if (!on_showData_triggered(true)) { // Если нажата кнопка отмены
        return;
    }
    auto result =
        plotPresenter->clusterData(nClusters_, nIterations_, isPlusPlus);
    if (result.second.empty()) // Если центроидов нет, то нет и кластеров
        return;
    // Обновление данных (старые удаляются, новые добавляются)
    plotPresenter->setData(result.first, nX_, nY_);
    for (int i = 0, cnt = plotPresenter->countOfClusters(); i < cnt; ++i) {
        plotPresenter->setStyle(
            i, QBrush(QColor().fromHsv(randInt(Hmin, Hmax), randInt(Smin, Smax),
                                       randInt(Vmin, Vmax))));
    }
    // Добавление центроидов
    plotPresenter->addCentroids(result.second, nX_, nY_);
}

void MainWindow::on_clusterDataPlusPlus_triggered() {
    on_clusterDataRnd_triggered(true);
}

// О программе
void MainWindow::on_about_triggered() {
    QMessageBox::information(this, "О программе", "Практика в СПбГУТ 2020.\n"
                                                  "Студент: Коваленко Леонид.\n"
                                                  "C++, QT, QCustomPlot.");
}

// Краткое описание программы
```

```
void MainWindow::on_description_triggered() {
    QMessageBox::information(
        this, "Краткое описание программы",
        "Программа для кластеризации данных.\nУправление "
        "осуществляется при помощи интуитивно понятной "
        "панели меню.\nОсновное:\n- Файл исходных данных "
        "CSV содержит строки, содержащие значения "
        "признаков безымянных объектов через запятую или точку с запятой. "
        "Комментарии идут после знака косой черты '/'.\n- Данные могут быть "
        "загружены или сгенерированы, а впоследствии сохранены, но не могут "
        "быть отредактированы в самой программе.\n- Стандартизация происходит "
        "для каждого кластера отдельно.\n- Кластеризация предполагает "
        "предварительное объединение всех кластеров в один.\n- "
        "Кластеризация проходит по всем признакам.\n- На графике изображаются "
        "точки данных по двум признакам (первому X и второму Y/... - "
        "настраивается в 'Посмотреть внутренние данные').\n- Цвета кластеров "
        "генерируются случайно.\n- Мера расстояния: Евклидово расстояние.");
}

// Открытие CSV-файла данных
void MainWindow::on_loadData_triggered() {
    // Открываем файл для чтения
    int nAttributes = -1, linePos = 0;
    QString filename = QFileDialog::getOpenFileName(this, "Открыть csv-файл",
                                                    ".", "CSV files (*.csv)");
    if (filename.size() == 0) // Если была нажата кнопка отмены
        return;
    QFile fileIn(filename);
    if (fileIn.exists() && fileIn.open(QIODevice::ReadOnly)) {
        QTextStream qts(&fileIn);
        QVector<Point> points;
        while (!qts.atEnd()) {
            QString line = qts.readLine();
            ++linePos;
            int pos = line.indexOf('/');
            if (pos >= 0) {
                line = line.left(pos);
            }
            line = line.simplified();
            QStringList s = line.split(QRegExp("[;,]+"));
            if (line.size() == 0)
```

```
                continue;
            if (nAttributes == -1)
                nAttributes = s.size();
            if (nAttributes == s.size()) {
                Point point(nAttributes);
                bool isOk = true;
                for (int i = 0; i < nAttributes; ++i) {
                    double temp = s[i].toDouble(&isOk);
                    if (isOk)
                        point[i] = temp;
                    else {
                        QMessageBox::critical(
                            this, "Ошибка в строке " + QString::number(linePos),
                            "Неверный формат файла: одно из "
                            "чисел написано неверно");
                        return;
                    }
                }
                points.append(point);
            } else {
                QMessageBox::critical(
                    this, "Ошибка в строке " + QString::number(linePos),
                    "Неверный формат файла: в каждой строке "
                    "должно быть одинаковое количество чисел, "
                    "характеризующих значения признаков каждого объекта");
                return;
            }
        }
        if (!points.empty()) {
            plotPresenter->addCluster(points, nX_,
                                      (nAttributes >= 2) ? nY_ : nX_);
            plotPresenter->setStyle(
                plotPresenter->countOfClusters() - 1,
                QBrush(QColor().fromHsv(randInt(Hmin, Hmax),
                                        randInt(Smin, Smax),
                                        randInt(Vmin, Vmax))));
        } else {
            QMessageBox::warning(this, "Внимание!", "Файл не содержит данных!");
        }
        fileIn.close();
    } else {
```

```
        QMessageBox::critical(this, "Ошибка",
                              "Не удалось открыть файл\n" + filename);
        return;
    }
}

// Сохранение внутренних данных в файл
void MainWindow::on_saveData_triggered() {
    QString filename = QFileDialog::getSaveFileName(this, "Сохранить как...",
                                                    ".", "All files (*.*)");
    if (filename.size() == 0) // Если была нажата кнопка отмены
        return;
    QFile fileOut(filename);
    if (fileOut.open(QIODevice::WriteOnly)) {
        QMessageBox::StandardButton answer =
            QMessageBox::question(this, "Ошибка", "Добавлять метки кластеров?");
        bool clusterLabel = (answer == QMessageBox::Yes);
        QTextStream qts(&fileOut);
        qts << "/ This file was generated by the ClusterData program\n";
        int n = 1;
        for (const auto &cluster : plotPresenter->data()) {
            if (cluster.size() > 0) {
                for (auto point : cluster) {
                    if (clusterLabel) {
                        qts << n;
                        for (auto x : point) {
                            qts << ';' << x;
                        }
                        qts << '\n';
                    } else {
                        qts << point.front();
                        for (auto it = point.begin() + 1; it != point.end();
                             ++it) {
                            qts << ';' << *it;
                        }
                        qts << '\n';
                    }
                }
                ++n;
            }
        }
```

```
        fileOut.close();
    } else {
        QMessageBox::critical(this, "Ошибка",
                              "Не удалось открыть файл\n" + filename);
        return;
    }
}

// Отображение внутренних данных
// bNoWarning -- если true, то предупреждений никаких не будет
bool MainWindow::on_showData_triggered(bool bNoWarning) {
    int oldX = nX_, oldY = nY_;
    runDialog.updateInfo(plotPresenter->data(), nClusters_, nIterations_, nX_,
                         nY_);
    if (runDialog.exec() == RunDialog::Accepted) {
        nClusters_ = runDialog.getClusters();
        nIterations_ = runDialog.getIterations();
        nX_ = runDialog.getX();
        nY_ = runDialog.getY();
        if (bNoWarning || (nX_ == oldX && nY_ == oldY)) {
            return true;
        }
        QMessageBox::Button answer = QMessageBox::question(
            this, "Внимание!", "Номера признаков были изменены! Вы уверены?");
        if (answer == QMessageBox::No) {
            nX_ = oldX;
            nY_ = oldY;
            return true;
        }
        // Обновление данных (старые удаляются, новые добавляются)
        plotPresenter->setData(plotPresenter->data(), nX_, nY_);
        for (int i = 0, cnt = plotPresenter->countOfClusters(); i < cnt; ++i) {
            plotPresenter->setStyle(
                i, QBrush(QColor().fromHsv(randInt(Hmin, Hmax),
                                           randInt(Smin, Smax),
                                           randInt(Vmin, Vmax))));
        }
        return true;
    }
    return false;
}
```

```
// Стандартизация по Z-оценке
void MainWindow::on_stdZ_triggered() { plotPresenter->zStd(nX_, nY_); }

// Стандартизация по среднеквадратическому отклонению
void MainWindow::on_stdSigma_triggered() { plotPresenter->sigmaStd(nX_, nY_); }

// Стандартизация по минимальному и максимальному значениям
void MainWindow::on_stdMinMax_triggered() {
    plotPresenter->minMaxStd(nX_, nY_);
}

// Объединение всех кластеров
void MainWindow::on_combineData_triggered() {
    plotPresenter->combineAll(nX_, nY_);
}

// Удаление всех кластеров
void MainWindow::on_clearData_triggered() { plotPresenter->clearAll(); }
```

Файл «mainwindow.ui»

```
<?xml version="1.0" encoding="UTF-8"?><ui version="4.0"><class>MainWindow</
class><widget class="QMainWindow" name="MainWindow"><property
name="geometry"><rect><x>0</x><y>0</y><width>320</width><height>120</height></rect></
property><property name="sizePolicy"><sizepolicy hsizetype="Expanding"
vsizetype="Expanding"><horstretch>0</horstretch><verstretch>0</verstretch></
sizepolicy></property><property name="minimumSize"><size><width>240</
width><height>120</height></size></property><property name="windowTitle"><string>Кла-
стеризация K-Means</string></property><property
name="toolButtonStyle"><enum>Qt::ToolButtonIconOnly</enum></property><widget
class="QWidget" name="centralWidget"><layout class="QVBoxLayout"
name="verticalLayout_2"><item><widget class="QCustomPlot" name="graphWidget"
native="true"><property name="sizePolicy"><sizepolicy hsizetype="Ignored"
vsizetype="Ignored"><horstretch>0</horstretch><verstretch>0</verstretch></
sizepolicy></property><property name="minimumSize"><size><width>0</width><height>0</
height></size></property><property name="sizeIncrement"><size><width>0</
width><height>0</height></size></property><layout class="QVBoxLayout"
name="verticalLayout"><property name="spacing"><number>0</number></property><property
name="sizeConstraint"><enum>QLayout::SetDefaultConstraint</enum></property><property
name="leftMargin"><number>0</number></property><property name="topMargin"><number>0</
number></property><property name="rightMargin"><number>0</number></property><property
name="bottomMargin"><number>0</number></property></layout></widget></item></layout></
widget><widget class="QMenuBar" name="menuBar"><property name="geometry"><rect><x>0</
x><y>0</y><width>320</width><height>24</height></rect></property><widget class="QMenu"
name="fileMenu"><property name="title"><string>Данные</string></property><widget
class="QMenu" name="saveGraph"><property name="title"><string>Сохранить график</
string></property><addaction name="saveAsPNG"/><addaction name="saveAsJPG"/><addaction
name="saveAsPDF"/></widget><widget class="QMenu" name="std"><property
name="title"><string>Стандартизация данных</string></property><addaction
name="stdSigma"/><addaction name="stdZ"/><addaction name="stdMinMax"/></
widget><addaction name="loadData"/><addaction name="genData"/><addaction name="std"/
><addaction name="separator"/><addaction name="showData"/><addaction name="saveData"/
><addaction name="saveGraph"/><addaction name="separator"/><addaction
name="combineData"/><addaction name="clearData"/><addaction name="separator"/
```

```
><addaction name="quit"/></widget><widget class="QMenu" name="kmeansMenu"><property
name="title"><string>Кластеризация</string></property><widget class="QMenu"
name="menu"><property name="title"><string>K-Means</string></property><addaction
name="clusterDataRnd"/><addaction name="clusterDataPlusPlus"/></widget><addaction
name="menu"/></widget><widget class="QMenu" name="help"><property
name="title"><string>Помощь</string></property><addaction name="description"/
><addaction name="about"/></widget><addaction name="fileMenu"/><addaction
name="kmeansMenu"/><addaction name="help"/></widget><action name="loadData"><property
name="text"><string>Загрузить данные из файла</string></property></action><action
name="genData"><property name="text"><string>Сгенерировать данные</string></
property></action><action name="saveData"><property name="text"><string>Сохранить дан-
ные как...</string></property></action><action name="clusterDataPlusPlus"><property
name="text"><string>K-Means++ инициализация</string></property></action><action
name="saveAsPNG"><property name="text"><string>как PNG...</string></property></
action><action name="saveAsJPG"><property name="text"><string>как JPG...</string></
property></action><action name="saveAsPDF"><property name="text"><string>как PDF...</
string></property></action><action name="quit"><property name="text"><string>Выйти</
string></property></action><action name="showData"><property name="text"><string>По-
смотреть данные</string></property></action><action name="stdZ"><property
name="text"><string>Z-оценка</string></property></action><action
name="stdSigma"><property name="text"><string>Среднеквадратичная</string></property></
action><action name="description"><property name="text"><string>Краткое описание</
string></property></action><action name="about"><property name="text"><string>О про-
грамме</string></property></action><action name="stdMinMax"><property
name="text"><string>Min-Max</string></property></action><action
name="combineData"><property name="text"><string>Объединить все кластеры</string></
property></action><action name="clearData"><property name="text"><string>Удалить все
кластеры</string></property></action><action name="clusterDataRnd"><property
name="text"><string>K-Means инициализация</string></property></action></
widget><layoutdefault spacing="6" margin="11"/
><customwidgets><customwidget><class>QCustomPlot</class><extends>QWidget</
extends><header>qcustomplot.h</header><container>1</container></customwidget></
customwidgets><resources/><connections/></ui>
```

Модуль RunDialog

Файл «rundialog.h»

```
#ifndef RUNDIALOG_H
#define RUNDIALOG_H

#include "cluster.h"
#include <QDialog>
#include <QTextStream>
namespace Ui {
    class RunDialog;
}

// Класс "Диалоговое окно внутренних данных"
class RunDialog : public QDialog {
    Q_OBJECT

  public:
    // Конструктор
    explicit RunDialog(QWidget *parent = 0);
```

```
    // Деструктор
    ~RunDialog();

    // Обновление информации на форме
    void updateInfo(const QVector<Cluster> &, int, int, int, int);

    // Получение числа кластеров из поля ввода формы
    int getClusters() const;
    // Получение числа итераций из поля ввода формы
    int getIterations() const;
    // Получение номера признака для координаты X
    int getX() const;
    // Получение номера признака для координаты Y
    int getY() const;

  private:
    Ui::RunDialog *ui; // Интерфейс окна
};

#endif // RUNDIALOG_H
```

Файл «rundialog.cpp»

```
#include "rundialog.h"
#include "ui_rundialog.h"

// Конструктор
RunDialog::RunDialog(QWidget *parent) : QDialog(parent), ui(new Ui::RunDialog) {
    ui->setupUi(this);
}

// Деструктор
RunDialog::~RunDialog() { delete ui; }

// Обновление информации на форме
void RunDialog::updateInfo(const QVector<Cluster> &data, int nClusters,
                           int nIterations, int nX, int nY) {
    int n = 1;
    ui->dataView->clear();
    for (auto &cluster : data) {
        for (auto &point : cluster) {
            QString s;
            QTextStream qts(&s);
```

```
            qts << n;
            for (auto x : point) {
                qts << ';' << x;
            }
            ui->dataView->addItem(s);
        }
        ++n;
    }
    int nPoints = ui->dataView->count();
    this->setWindowTitle("Начальные данные (число объектов: " +
                         QString::number(nPoints) + ")");
    ui->spinClusters->setValue(nClusters);
    ui->spinClusters->setMaximum(nPoints);
    ui->spinIterations->setValue(nIterations);
    if (!data.empty() && !data.first().empty() &&
        !data.first().front().empty()) {
        ui->spinX->setMaximum(data.first().front().size());
        ui->spinY->setMaximum(data.first().front().size());
    } else {
        ui->spinX->setMaximum(1);
        ui->spinY->setMaximum(1);
    }
    ui->spinX->setValue(nX + 1);
    ui->spinY->setValue(nY + 1);
}

// Получение числа кластеров из поля ввода формы
int RunDialog::getClusters() const { return ui->spinClusters->text().toInt(); }

// Получение числа итераций из поля ввода формы
int RunDialog::getIterations() const {
    return ui->spinIterations->text().toInt();
}

// Получение номера признака для координаты X
int RunDialog::getX() const { return ui->spinX->text().toInt() - 1; }

// Получение номера признака для координаты Y
int RunDialog::getY() const { return ui->spinY->text().toInt() - 1; }
```

Файл «rundialog.ui»

```
<?xml version="1.0" encoding="UTF-8"?><ui version="4.0"><class>RunDialog</
class><widget class="QDialog" name="RunDialog"><property
name="windowModality"><enum>Qt::ApplicationModal</enum></property><property
name="geometry"><rect><x>0</x><y>0</y><width>453</width><height>334</height></rect></
property><property name="windowTitle"><string>Внутренние данные</string></
property><layout class="QVBoxLayout" name="verticalLayout_2"><item><layout
class="QFormLayout" name="formLayout"><property
name="fieldGrowthPolicy"><enum>QFormLayout::AllNonFixedFieldsGrow</enum></
property><property name="rowWrapPolicy"><enum>QFormLayout::DontWrapRows</enum></
property><property name="labelAlignment"><set>Qt::AlignCenter</set></
property><property name="formAlignment"><set>Qt::AlignCenter</set></property><item
row="0" column="0"><widget class="QLabel" name="labelClusters"><property
name="font"><font><pointsize>12</pointsize></font></property><property
name="text"><string>Количество кластеров:</string></property><property
name="textFormat"><enum>Qt::AutoText</enum></property><property
name="alignment"><set>Qt::AlignCenter</set></property><property
name="wordWrap"><bool>false</bool></property></widget></item><item row="0"
column="1"><widget class="QSpinBox" name="spinClusters"><property
name="maximumSize"><size><width>180</width><height>16777215</height></size></
property><property name="alignment"><set>Qt::AlignCenter</set></property><property
name="correctionMode"><enum>QAbstractSpinBox::CorrectToNearestValue</enum></
property><property name="minimum"><number>1</number></property><property
name="maximum"><number>1000</number></property><property name="value"><number>5</
number></property></widget></item><item row="1" column="0"><widget class="QLabel"
name="labelIterations"><property name="text"><string>Количество итераций:</string></
property><property name="alignment"><set>Qt::AlignCenter</set></property></widget></
item><item row="1" column="1"><widget class="QSpinBox" name="spinIterations"><property
name="maximumSize"><size><width>180</width><height>16777215</height></size></
property><property name="alignment"><set>Qt::AlignCenter</set></property><property
name="buttonSymbols"><enum>QAbstractSpinBox::UpDownArrows</enum></property><property
name="correctionMode"><enum>QAbstractSpinBox::CorrectToNearestValue</enum></
property><property name="minimum"><number>1</number></property><property
name="maximum"><number>10000000</number></property><property name="value"><number>50</
number></property></widget></item><item row="2" column="0"><widget class="QLabel"
name="labelX"><property name="text"><string>Номер признака для X:</string></
property><property name="alignment"><set>Qt::AlignCenter</set></property></widget></
item><item row="3" column="0"><widget class="QLabel" name="labelY"><property
name="text"><string>Номер признака для Y:</string></property><property
name="alignment"><set>Qt::AlignCenter</set></property></widget></item><item row="2"
column="1"><widget class="QSpinBox" name="spinX"><property
name="maximumSize"><size><width>180</width><height>16777215</height></size></
property><property name="alignment"><set>Qt::AlignCenter</set></property><property
name="correctionMode"><enum>QAbstractSpinBox::CorrectToNearestValue</enum></
property><property name="minimum"><number>1</number></property><property
name="maximum"><number>2</number></property></widget></item><item row="3"
column="1"><widget class="QSpinBox" name="spinY"><property
name="maximumSize"><size><width>180</width><height>16777215</height></size></
property><property name="alignment"><set>Qt::AlignCenter</set></property><property
name="correctionMode"><enum>QAbstractSpinBox::CorrectToNearestValue</enum></
property><property name="minimum"><number>1</number></property><property
name="maximum"><number>2</number></property><property name="value"><number>2</
number></property></widget></item></layout></item><item><widget class="QListWidget"
name="dataView"><property name="editTriggers"><set>QAbstractItemView::NoEditTriggers</
set></property></widget></item><item><widget class="QDialogButtonBox"
name="buttonBox"><property name="orientation"><enum>Qt::Horizontal</enum></
property><property name="standardButtons"><set>QDialogButtonBox::Cancel|
QDialogButtonBox::Ok</set></property><property name="centerButtons"><bool>true</
bool></property></widget></item></layout></widget><resources/
><connections><connection><sender>buttonBox</sender><signal>accepted()</
signal><receiver>RunDialog</receiver><slot>accept()</slot><hints><hint
type="sourcelabel"><x>248</x><y>254</y></hint><hint type="destinationlabel"><x>157</
x><y>274</y></hint></hints></connection><connection><sender>buttonBox</
sender><signal>rejected()</signal><receiver>RunDialog</receiver><slot>reject()</
slot><hints><hint type="sourcelabel"><x>316</x><y>260</y></hint><hint
```

```
type="destinationlabel"><x>286</x><y>274</y></hint></hints></connection></
connections></ui>
```

Модуль GenDialog

Файл «gendialog.h»

```
#ifndef GENDIALOG_H
#define GENDIALOG_H

#include <QDialog>

namespace Ui {
    class GenDialog;
}

class GenDialog : public QDialog {
    Q_OBJECT

  public:
    // Конструктор
    explicit GenDialog(QWidget *parent = 0);
    // Деструктор
    ~GenDialog();

    // Обновление информации на форме
    void updateInfo(int, int, qreal, qreal, qreal, qreal);

    // Получение числа точек
    int getNumberOfPoints();
    // Получение вида фигуры
    int getKind();
    // Получение диапазона по X
    QPair<qreal, qreal> getRangeX();
    // Получение диапазона по Y
    QPair<qreal, qreal> getRangeY();

  private:
    Ui::GenDialog *ui;
};

#endif // GENDIALOG_H
```

Файл «gendialog.cpp»

```
#include "gendialog.h"
#include "ui_gendialog.h"

GenDialog::GenDialog(QWidget *parent) : QDialog(parent), ui(new Ui::GenDialog) {
    ui->setupUi(this);
    this->setFixedSize(this->size().rwidth(), this->size().rheight());
}

GenDialog::~GenDialog() { delete ui; }

// Обновление информации на форме
void GenDialog::updateInfo(int nPoints, int kind, qreal xFrom, qreal xTo,
                           qreal yFrom, qreal yTo) {
    ui->spinPoints->setValue(nPoints);
    ui->comboKind->setCurrentIndex(kind);
    ui->spinXfrom->setValue(xFrom);
    ui->spinXto->setValue(xTo);
    ui->spinYfrom->setValue(yFrom);
    ui->spinYto->setValue(yTo);
}

// Получение числа точек
int GenDialog::getNumberOfPoints() { return ui->spinPoints->value(); }

// Получение вида фигуры
int GenDialog::getKind() { return ui->comboKind->currentIndex(); }

// Получение диапазона по X
QPair<qreal, qreal> GenDialog::getRangeX() {
    return QPair<qreal, qreal>(ui->spinXfrom->value(), ui->spinXto->value());
}

// Получение диапазона по Y
QPair<qreal, qreal> GenDialog::getRangeY() {
    return QPair<qreal, qreal>(ui->spinYfrom->value(), ui->spinYto->value());
}
```

Файл «gendialog.ui»

```
<?xml version="1.0" encoding="UTF-8"?><ui version="4.0"><class>GenDialog</
class><widget class="QDialog" name="GenDialog"><property
name="windowModality"><enum>Qt::ApplicationModal</enum></property><property
name="geometry"><rect><x>0</x><y>0</y><width>480</width><height>160</height></rect></
```

```
property><property name="windowTitle"><string>Сгенерировать исходные данные случайно</
string></property><widget class="QDialogButtonBox" name="buttonBox"><property
name="geometry"><rect><x>10</x><y>130</y><width>460</width><height>27</height></
rect></property><property name="orientation"><enum>Qt::Horizontal</enum></
property><property name="standardButtons"><set>QDialogButtonBox::Cancel|
QDialogButtonBox::Ok</set></property><property name="centerButtons"><bool>true</
bool></property></widget><widget class="QSpinBox" name="spinPoints"><property
name="geometry"><rect><x>170</x><y>10</y><width>300</width><height>28</height></
rect></property><property name="sizePolicy"><sizepolicy hsizetype="Minimum"
vsizetype="Fixed"><horstretch>0</horstretch><verstretch>0</verstretch></sizepolicy></
property><property name="maximumSize"><size><width>16777215</width><height>16777215</
height></size></property><property name="alignment"><set>Qt::AlignCenter</set></
property><property
name="correctionMode"><enum>QAbstractSpinBox::CorrectToNearestValue</enum></
property><property name="maximum"><number>1000000</number></property><property
name="singleStep"><number>10</number></property><property name="value"><number>250</
number></property></widget><widget class="QLabel" name="labelKind"><property
name="geometry"><rect><x>5</x><y>45</y><width>160</width><height>20</height></rect></
property><property name="sizePolicy"><sizepolicy hsizetype="Preferred"
vsizetype="Preferred"><horstretch>0</horstretch><verstretch>0</verstretch></
sizepolicy></property><property name="text"><string>Вид фигуры:</string></
property><property name="alignment"><set>Qt::AlignRight|Qt::AlignTrailing|
Qt::AlignVCenter</set></property><property name="wordWrap"><bool>false</bool></
property></widget><widget class="QComboBox" name="comboKind"><property
name="geometry"><rect><x>170</x><y>40</y><width>300</width><height>28</height></
rect></property><property name="sizePolicy"><sizepolicy hsizetype="Minimum"
vsizetype="Fixed"><horstretch>0</horstretch><verstretch>0</verstretch></sizepolicy></
property><property name="maximumSize"><size><width>16777215</width><height>16777215</
height></size></property><item><property name="text"><string>Эллипс</string></
property></item><item><property name="text"><string>Прямоугольник</string></
property></item></widget><widget class="QLabel" name="labelPoints"><property
name="geometry"><rect><x>5</x><y>15</y><width>160</width><height>20</height></rect></
property><property name="maximumSize"><size><width>16777215</width><height>16777215</
height></size></property><property name="text"><string>Количество точек:</string></
property><property name="alignment"><set>Qt::AlignRight|Qt::AlignTrailing|
Qt::AlignVCenter</set></property></widget><widget class="QLabel"
name="labelX"><property name="geometry"><rect><x>5</x><y>75</y><width>160</
width><height>20</height></rect></property><property name="text"><string>Диапазон по
X:</string></property><property name="alignment"><set>Qt::AlignRight|
Qt::AlignTrailing|Qt::AlignVCenter</set></property></widget><widget class="QLabel"
name="labelY"><property name="geometry"><rect><x>5</x><y>105</y><width>160</
width><height>20</height></rect></property><property name="text"><string>Диапазон по
Y:</string></property><property name="alignment"><set>Qt::AlignRight|
Qt::AlignTrailing|Qt::AlignVCenter</set></property></widget><widget
class="QDoubleSpinBox" name="spinXfrom"><property name="geometry"><rect><x>170</
x><y>70</y><width>150</width><height>28</height></rect></property><property
name="maximumSize"><size><width>16777215</width><height>16777215</height></size></
property><property name="alignment"><set>Qt::AlignCenter</set></property><property
name="correctionMode"><enum>QAbstractSpinBox::CorrectToNearestValue</enum></
property><property name="decimals"><number>6</number></property><property
name="minimum"><double>-1000000000.000000000000000</double></property><property
name="maximum"><double>1000000000.000000000000000</double></property><property
name="singleStep"><double>0.500000000000000</double></property></widget><widget
class="QDoubleSpinBox" name="spinXto"><property name="geometry"><rect><x>320</
x><y>70</y><width>150</width><height>28</height></rect></property><property
name="maximumSize"><size><width>16777215</width><height>16777215</height></size></
property><property name="alignment"><set>Qt::AlignCenter</set></property><property
name="correctionMode"><enum>QAbstractSpinBox::CorrectToNearestValue</enum></
property><property name="decimals"><number>6</number></property><property
name="minimum"><double>-1000000000.000000000000000</double></property><property
name="maximum"><double>1000000000.000000000000000</double></property><property
name="singleStep"><double>0.500000000000000</double></property><property
name="value"><double>5.000000000000000</double></property></widget><widget
class="QDoubleSpinBox" name="spinYfrom"><property name="geometry"><rect><x>170</
x><y>100</y><width>150</width><height>28</height></rect></property><property
name="maximumSize"><size><width>16777215</width><height>16777215</height></size></
property><property name="alignment"><set>Qt::AlignCenter</set></property><property
```

```
name="correctionMode"><enum>QAbstractSpinBox::CorrectToNearestValue</enum></
property><property name="decimals"><number>6</number></property><property
name="minimum"><double>-1000000000.000000000000000</double></property><property
name="maximum"><double>1000000000.000000000000000</double></property><property
name="singleStep"><double>0.500000000000000</double></property><property
name="value"><double>0.000000000000000</double></property></widget><widget
class="QDoubleSpinBox" name="spinYto"><property name="geometry"><rect><x>320</
x><y>100</y><width>150</width><height>28</height></rect></property><property
name="maximumSize"><size><width>16777215</width><height>16777215</height></size></
property><property name="alignment"><set>Qt::AlignCenter</set></property><property
name="correctionMode"><enum>QAbstractSpinBox::CorrectToNearestValue</enum></
property><property name="decimals"><number>6</number></property><property
name="minimum"><double>-1000000000.000000000000000</double></property><property
name="maximum"><double>1000000000.000000000000000</double></property><property
name="singleStep"><double>0.500000000000000</double></property><property
name="value"><double>5.000000000000000</double></property></widget></
widget><resources/><connections><connection><sender>buttonBox</
sender><signal>accepted()</signal><receiver>GenDialog</receiver><slot>accept()</
slot><hints><hint type="sourcelabel"><x>248</x><y>254</y></hint><hint
type="destinationlabel"><x>157</x><y>274</y></hint></hints></
connection><connection><sender>buttonBox</sender><signal>rejected()</
signal><receiver>GenDialog</receiver><slot>reject()</slot><hints><hint
type="sourcelabel"><x>316</x><y>260</y></hint><hint type="destinationlabel"><x>286</
x><y>274</y></hint></hints></connection></connections></ui>
```

Файл «ClusterData.pro»

```
#-------------------------------------------------
#
# Project created by QtCreator 2020-07-30T20:10:50
#
#-------------------------------------------------

QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets printsupport

TARGET = ClusterData
TEMPLATE = app

# The following define makes your compiler emit warnings if you use
# any feature of Qt which as been marked as deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

SOURCES += main.cpp\
        mainwindow.cpp \
    qcustomplot.cpp \
    plotmodel.cpp \
    plotpresenter.cpp \
    kmeans.cpp \
```

```
    rundialog.cpp \
    cluster.cpp \
    point.cpp \
    gendialog.cpp

HEADERS  += mainwindow.h \
    qcustomplot.h \
    plotmodel.h \
    plotpresenter.h \
    cluster.h \
    kmeans.h \
    point.h \
    rundialog.h \
    gendialog.h

FORMS    += mainwindow.ui \
    rundialog.ui \
    gendialog.ui
```