

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

Институт: «Институт кибербезопасности и цифровых технологий»

Направление: 09.03.02 «Информационные системы и технологии»

Кафедра: КБ-14 «Цифровые технологии обработки данных»

Дисциплина: «Информационные хранилища и информационно-аналитические системы»

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**

**«Кластеризация методами k-means и fuzzy c-means»**

Выполнил студент БСБО-08-18 Шайко Т.И.

Руководитель работы: Лебедев А.С.

Москва 2021 г.

Оглавление

[Цель работы 3](#_Toc93449934)

[Теоретическое введение 3](#_Toc93449935)

[Хор работы 6](#_Toc93449936)

[Тестирование 11](#_Toc93449937)

[Вывод 25](#_Toc93449938)

[Список использованной литературы 26](#_Toc93449939)

[Приложение 27](#_Toc93449940)

# Цель работы

Реализовать кластеризацию методами k-means и fuzzy c-means. Набор данных выбирается самостоятельно. Возможен выбор популярного датасета “Ирисы Фишера”.

# Теоретическое введение

[4] С развитием сети Интернет получили развитие вопросы построения распределенных баз данных, создания распределенных глобальных информационных систем. Многократно возросла интенсивность формирования и архивирования различных данных, за которыми следовало развитие масштабируемых программно-аппаратных комплексов, дорогостоящих мощных и недорогих пользовательских компьютеров и накопителей данных.

Все это способствовало всплеску развития индустрии ИКТ и сделало огромное количество баз данных доступными для хранения разнородной информации в значительных объемах и управления транзакциями в них. При этом все больше возникала **потребность анализа имеющихся данных** в разновременном аспекте, с возможностью построения произвольных запросов, при условии обработки сверхбольших объемов данных, полученных, в том числе, из различных регистрирующих БД. Использование для этих задач традиционных регистрирующих систем и БД крайне затруднительно. Например, в регистрирующей системе информация актуальна исключительно на момент обращения к БД, а в следующий момент времени по тому же запросу можно ожидать другой результат. Интерфейс таких систем рассчитан на проведение определенных стандартизованных операций и возможности получения результатов на нерегламентированный произвольный запрос ограничены. Возможности обработки больших массивов данных также могут быть ограничены вследствие ориентации СУБД на нормализованные данные, характерные для стандартных реляционных регистрирующих БД.

Ответом на возникшую потребность стало появление новой технологии организации баз данных — технологии **Хранилищ Данных** (англ. «Data Warehouse»), предполагающей некоторую предварительную обработку данных и их интеграцию, а также онлайновую аналитическую обработку (англ. «On-Line Analytical Processing», OLAP).

Несмотря на очевидную пользу такого инструмента анализа данных, он ориентирован на хорошо нормализованные табличные данные и не предполагает использование целого ряда дополнительного аналитического инструментария типа классификации, кластеризации, регрессионного анализа, моделирования, прогнозирования и интерпретации многомерных данных и т.п.

Таким образом, сегодня наблюдается высокий уровень развития масштабируемой аппаратно-программной ИКТ инфраструктуры, позволяющей увеличивать и без того значительные архивы данных. Имеется достаточно существенный задел в области компьютерных наук и информационных технологий, разработаны теория и прикладные аспекты теории вероятности и математической статистики. Однако при этом следует признать, что присутствует заметный избыток данных при дефиците информации и знаний. Быстро растущие объемы накопленных и пополняемых (автоматически, а не людьми — как это было когда-то) архивов данных пока существенно превышают способности человека в их практически полезной обработке. Для обострения этого тезиса иногда говорят, что «большие базы данных стали могилами, которые редко посещаются». Как следствие, важные решения порой принимаются не на основе аналитических выводов из информативных БД, а на основе интуиции человека, не имеющего подходящих инструментов для извлечения полезных знаний из имеющихся огромных объемов данных.

Поэтому в последние годы стремительное развитие получила область **Data Mining** (в отечественной литературе наиболее используемая аналогия — интеллектуальный анализ Данных, ИАД), направленная на поиск и разработку методов извлечения из имеющихся данных знаний, позволяющих принимать на их основе конкретные, в высокой степени обоснованные, практически полезные управленческие решения [4].

# Хор работы

Для написания кода была использована IDE IntelliJ IDEA. Язык программирования был выбран Java. Помимо стандартных Java-библиотек, использовалась также библиотека для построения графиков и диаграмм «JFreeChart» автора Дэвида Гилберта (David Gilbert; он же jfree) [3].

Как указано, в презентации, которую сделали мои одногруппницы, есть строго определённые шаги при выполнении алгоритмов «k-means» и «fuzzy c-means» [1,2]. Их вы, дорогой читатель, должны придерживаться.

Структура кода как реализации первого из этих алгоритмов кластеризации содержит в себе три класса: Main, KMeans и Universe (от англ. «Вселенная»).

Группируются в этой работе звёзды в шесть заранее выбранных кластеров (т.е. групп) (см. рисунок 1).

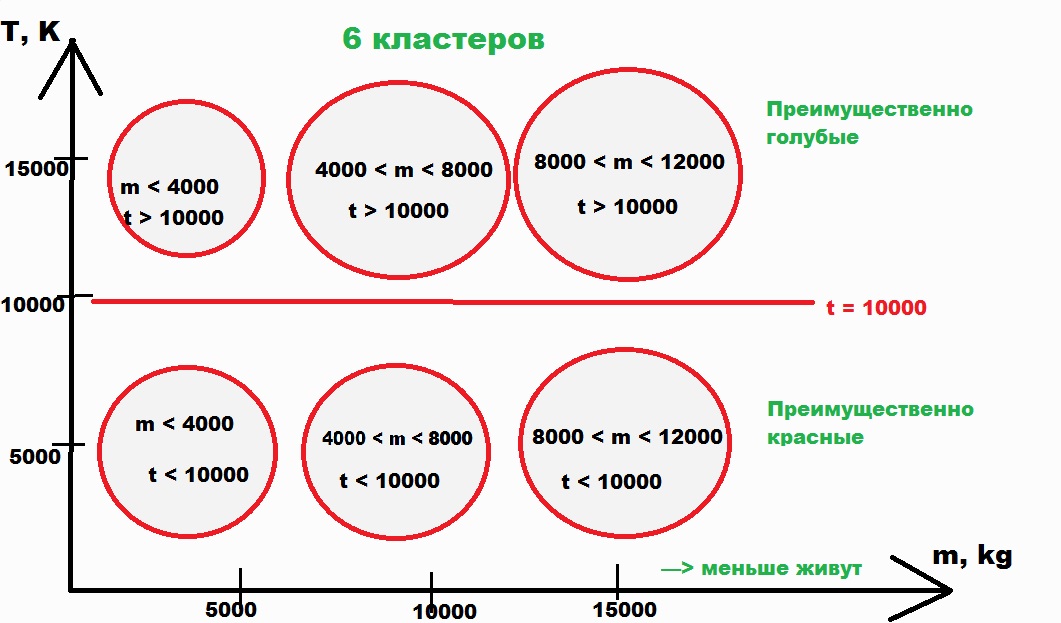


Рисунок 1 — Исследуемая предметная область

Чем выше температура, тем у звезды цвет холоднее. И чем выше масса звезды, тем короче её жизненный цикл. Совсем массивные звёзды превращаются в нейтронные звёзды или чёрные дыры в конце своего существования.

Единицы обсуждаемых величин, хоть и показаны в килограммах и кельвинах (см. рисунок выше), но всё равно являются по большей части выдуманными и несоответствующими реальным значениям температуры и массы подобных светил. Но в целом, для этой работы сойдут и такие, очень-очень приближённые значения. Это сделано для удобства реализации кода.

Класс Universe описывает звёзды и используется в классе KMeans в процессе группировки (кластеризации). Класс Main по умолчанию начальная точка запуска всей программы. В Main рисуется также график при помощи модуля JFreeChart.

Начальная выборка звёзд задаётся в том же классе Main в самом начале (см. код 1).

Код 1 — Выборка кластеризуемых объектов (звёзд)

|  |
| --- |
| public static void main(String[] args) {  String[] stars = new String[18]; stars[0] = "1000:1000.0;-1"; stars[1] = "2050:1400.0;-1"; stars[2] = "3000:2500.0;-1"; stars[3] = "2100:5600.0;-1"; stars[4] = "5050:6000.0;-1"; stars[5] = "3000:6900.0;-1"; stars[6] = "4500:11500.0;-1"; stars[7] = "5050:12000.0;-1"; stars[8] = "7000:10800.0;-1"; stars[9] = "18100:10350.0;-1"; stars[10] = "18800:11110.0;-1"; stars[11] = "16300:12000.0;-1"; stars[12] = "10000:7000.0;-1"; stars[13] = "13000:7300.0;-1"; stars[14] = "15500:7900.0;-1"; stars[15] = "13700:1500.0;-1"; stars[16] = "15000:2100.0;-1"; stars[17] = "16000:3700.0;-1"; |

В классе Universe есть массив строк stars, который и содержит в подобном формате описание всех звёзд (объектов). Массив выше (см. код 1) передаётся в качестве аргумента в конструктор класса Universe при его создании, и тем самым инициализируются все объекты Universe, над которыми и будет проводиться работа.

Этот Universe, как было сказано чуть раньше, передаётся следом в конструктор уже KMeans — класс, который содержит в себе методы алгоритма кластеризации «k-means» [2,1]:

|  |
| --- |
| KMeans kmeans = new KMeans(new Universe(stars)); |

Далее идёт вывод результатов в консоль и в окно модуля JFreeChart:

Код 2 — Остальная часть метода main()

|  |
| --- |
| XYSeries series = new XYSeries("Universe");  int[] temperatures = kmeans.sample.getTemp(); float[] masses = kmeans.sample.getMass();  System.*out*.println(kmeans.sample);  kmeans.run();  System.*out*.println(kmeans.sample);  for(int i = 0; i < kmeans.sample.count(); ++i) {  series.add(masses[i], temperatures[i]); }  XYDataset xyDataset = new XYSeriesCollection(series);  JFreeChart chart = ChartFactory  .*createScatterPlot*("Stars", "MASS", "TEMPERATURE",  xyDataset,  PlotOrientation.*VERTICAL*,  true, true, true);  JFrame frame =  new JFrame("Clusterization"); frame.getContentPane()  .add(new ChartPanel(chart));  frame.setSize(400,300); frame.show(); |

Метод run(), как видно из кода 2 выше, принадлежит классу KMeans и выполняет основную часть алгоритма k-means. Переменная sample у объекта kmeans в том же коде выше и есть объект Вселенной, хранящий массив всех звёзд. А благодаря методам getTemp() и getMass() вытаскиваются из строк (см. код 1) соответственно масса и температура каждой звезды и возвращаются в виде массивов.

Чтобы вам было понятнее, дорогой читатель, нужно пояснить значение этих строк:

|  |
| --- |
| stars[0] = "1000:1000.0;-1"; stars[1] = "2050:1400.0;-1"; stars[2] = "3000:2500.0;-1"; |

Число до двоеточия в них — это *температура*. Число с плавающей точкой после двоеточия обозначает значение *массы* соответствующей звезды; а цифра в конце означает принадлежность звезды к какому-либо кластеру. Кластеры нумеруются от 0-ля до 5-ти, а *-1* означает, что звезда пока что не принадлежит ни к какому кластеру.

Таким образом, заданная выборка объектов во Вселенной будет графически выглядеть так — см. рисунок 2.

Глазом из этого рисунка уже можно примерно выделить группы, небольшие скопления звёзд.

Класс Universe вбирает в себя весь код, необходимый для редактирования и чтения данных о его объектах. К примеру, публичный метод changeCluster() изменяет номер кластера какого-либо объекта, а метод getStars() выступает в роли геттера для переменной-массива класса stars.

Полный код этого класса представлен в приложении к этой работе в самом конце.

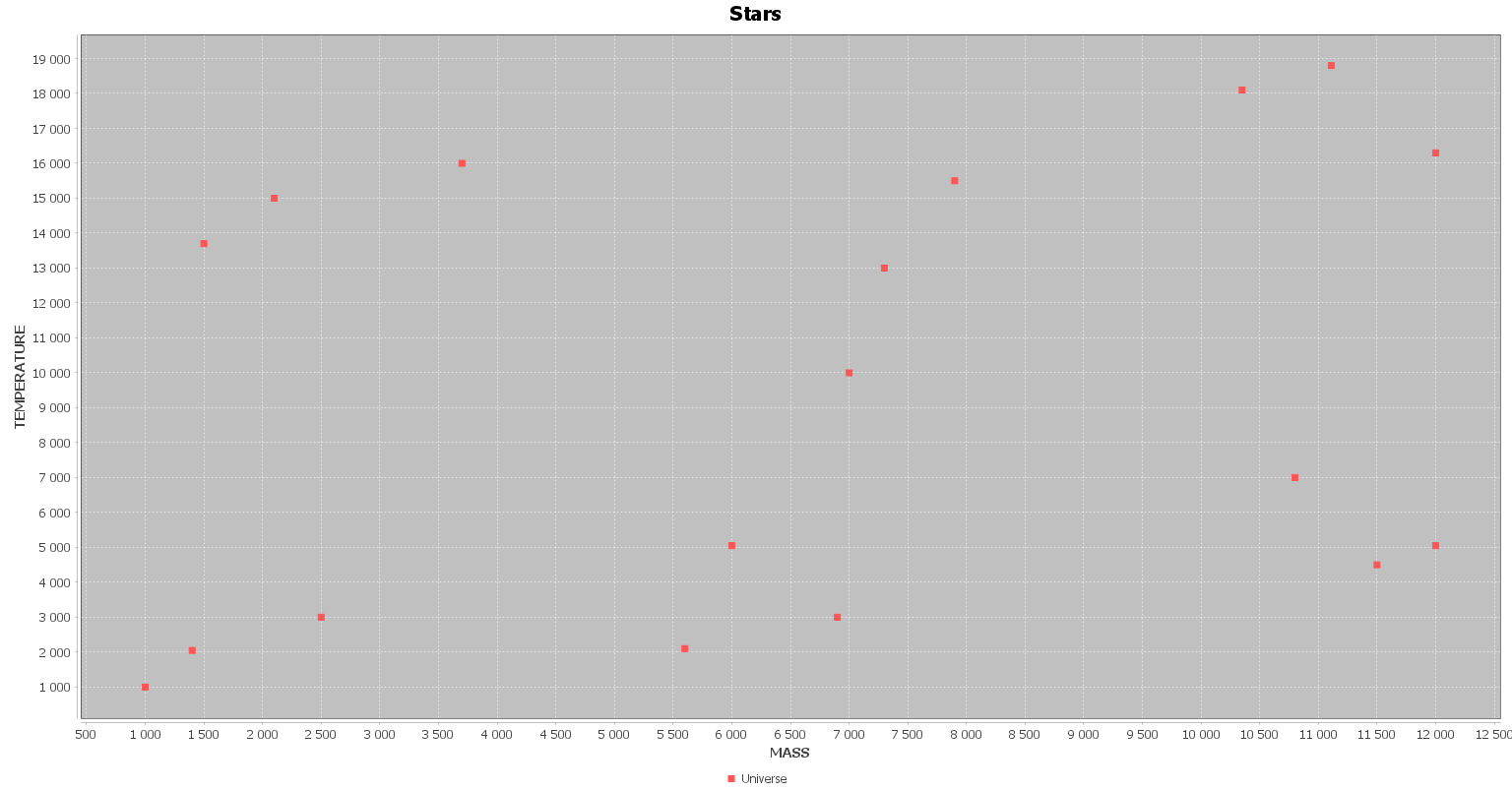


Рисунок 2 — Начальный некластеризованный набор звёзд [3]

Как и код класса KMeans с основной частью этой лабораторной работы тоже представлен в приложении в конце.

## Тестирование

В итоге, из-за *случайной компоненты* при выборе *начальных центров* всех кластеров программа выдаёт несколько возможных вариантов группировки звёзд по шести кластерам.

Первый вариант представлен в блоке Код 3 ниже.

Код 3 — Вывод программы в консоль

|  |
| --- |
| 0)  Temperature: 1000  Mass: 1000.0  Cluster: 0  1)  Temperature: 2050  Mass: 1400.0  Cluster: -1  2)  Temperature: 3000  Mass: 2500.0  Cluster: -1  3)  Temperature: 2100  Mass: 5600.0  Cluster: -1  4)  Temperature: 5050  Mass: 6000.0  Cluster: -1  5)  Temperature: 3000  Mass: 6900.0  Cluster: 1  6)  Temperature: 4500  Mass: 11500.0  Cluster: -1  7)  Temperature: 5050  Mass: 12000.0  Cluster: -1  8)  Temperature: 7000  Mass: 10800.0  Cluster: -1  9)  Temperature: 18100  Mass: 10350.0  Cluster: -1  10)  Temperature: 18800  Mass: 11110.0  Cluster: 5  11)  Temperature: 16300  Mass: 12000.0  Cluster: 5  12)  Temperature: 10000  Mass: 7000.0  Cluster: -1  13)  Temperature: 13000  Mass: 7300.0  Cluster: -1  14)  Temperature: 15500  Mass: 7900.0  Cluster: -1  15)  Temperature: 13700  Mass: 1500.0  Cluster: -1  16)  Temperature: 15000  Mass: 2100.0  Cluster: 3  17)  Temperature: 16000  Mass: 3700.0  Cluster: 3  i = 0 ( 1633.3333333333333 ; 2016.6666666666667 ) cluster: 0.0  i = 1 ( 8800.0 ; 4450.0 ) cluster: 1.0  i = 3 ( 4320.0 ; 13540.0 ) cluster: 3.0  i = 5 ( 10340.0 ; 17175.0 ) cluster: 5.0  v = 1.2798692222222222E8  i = 0 ( 2625.0 ; 2037.5 ) cluster: 0.0  i = 1 ( 9440.0 ; 4920.0 ) cluster: 1.0  i = 3 ( 4320.0 ; 13540.0 ) cluster: 3.0  i = 5 ( 10340.0 ; 17175.0 ) cluster: 5.0  v = 1.20899075E8  i = 0 ( 2625.0 ; 2037.5 ) cluster: 0.0  i = 1 ( 9440.0 ; 4920.0 ) cluster: 1.0  i = 3 ( 4320.0 ; 13540.0 ) cluster: 3.0  i = 5 ( 10340.0 ; 17175.0 ) cluster: 5.0  v = 1.20899075E8  0)  Temperature: 1000  Mass: 1000.0  Cluster: 0  1)  Temperature: 2050  Mass: 1400.0  Cluster: 0  2)  Temperature: 3000  Mass: 2500.0  Cluster: 0  3)  Temperature: 2100  Mass: 5600.0  Cluster: 0  4)  Temperature: 5050  Mass: 6000.0  Cluster: 1  5)  Temperature: 3000  Mass: 6900.0  Cluster: 1  6)  Temperature: 4500  Mass: 11500.0  Cluster: 1  7)  Temperature: 5050  Mass: 12000.0  Cluster: 1  8)  Temperature: 7000  Mass: 10800.0  Cluster: 1  9)  Temperature: 18100  Mass: 10350.0  Cluster: 5  10)  Temperature: 18800  Mass: 11110.0  Cluster: 5  11)  Temperature: 16300  Mass: 12000.0  Cluster: 5  12)  Temperature: 10000  Mass: 7000.0  Cluster: 3  13)  Temperature: 13000  Mass: 7300.0  Cluster: 3  14)  Temperature: 15500  Mass: 7900.0  Cluster: 5  15)  Temperature: 13700  Mass: 1500.0  Cluster: 3  16)  Temperature: 15000  Mass: 2100.0  Cluster: 3  17)  Temperature: 16000  Mass: 3700.0  Cluster: 3 |

Как видно из этого вывода, алгоритм решил обойтись вообще без двух кластеров (с номерами 2 и 4) и смог кластеризовать объекты и таким способом.

Способ группировки тут зависит от начального выбора центров кластеров, который, как уже упоминалось, имеет случайную компоненту — то есть алгоритм характеризуется чётко прослеживаемой стохастичностью.

Метод init() класса KMeans инициализирует алгоритм и определяет начальные центры всех групп. В нём используется класс Random, генерирующий псевдослучайные числа. См. код 4.

Расстояния (а р*а*вно различия) между объектами представлены Евклидовой метрикой:

Чем больше расстояние , тем больше *различие* между двумя объектами (см. код 4, выделенное жёлтым маркером).

Код 4 — Метод KMeans.init()

|  |
| --- |
| private int[] init() {  Random random = new Random();int[] centers = new int[CLUSTERS\_NUM]; Arrays.*fill*(centers, -2);  int counter = 0; centers[counter++] = random.nextInt(sample.count()-1); int[] temperatures = sample.getTemp(); *// массив температур звёзд* float[] masses = sample.getMass(); *// массив масс звёзд Вселенной* int[] clusters = sample.getClusters(); *// массив кластеров каждой звезды*  *// Поэтапно определить все необходимые центроиды:* while (counter < this.CLUSTERS\_NUM) {  double[] min\_distances = new double[sample.count()]; double[][] distances = new double[sample.count()][sample.count()]; double sum = 0; for (int i = 0; i < temperatures.length; ++i) {  double min = -1; for (int j = 0; j < counter; ++j) {  distances[i][j] = (temperatures[centers[j]] - temperatures[i]) \* (temperatures[centers[j]] - temperatures[i]) + (masses[centers[j]] - masses[i]) \* (masses[centers[j]] - masses[i]);  if (min == -1)  min = distances[i][j];  else if (min > distances[i][j])  min = distances[i][j];   }  min\_distances[i] = min;  sum += min;  }  *// Выбрать следующий случайный центроид, опираясь уже на проставленные (см. выше) вероятности выбора:* double rnd = random.nextDouble() \* sum;  sum = 0;   boolean flag = false;  for (int i = 0; i < temperatures.length; ++i) {  sum += min\_distances[i];  if (sum > rnd) {  centers[counter++] = i;  break;  }  }  }   return centers; } |

Буква в выводе (см. код 3) обозначает *суммарное квадратичное отклонение* точек кластеров от центроидов этих кластеров [5]. Смысл алгоритма заключается в том, чтобы минимизировать это значение :

(1)

Это значит максимально компактно сгруппировать все объекты к центрам кластеров (то есть к центроидам), чтобы получилось несколько вполне различных групп объектов.

В формуле (1) — это центроид для кластера ; а — это количество кластеров в исследовании (в этой работе их, напомним, шесть).

В коде это значение, представленное формулой (1), выражается отдельным методом:

|  |
| --- |
| private double V(double[][] centers) {  int[] temperatures = sample.getTemp(); *// массив температур звёзд* float[] masses = sample.getMass(); *// массив масс звёзд Вселенной* int[] clusters = sample.getClusters(); *// массив кластеров каждой звезды* double sum = 0;   for (int i = 0; i < centers.length; ++i)  for (int j = 0; j < sample.count(); ++j) {  if (clusters[j] == i)  sum += (centers[i][0] - masses[j]) \* (centers[i][0] - masses[j]) + (centers[i][1] - temperatures[j]) \* (centers[i][1] - temperatures[j]);  }   return sum; } |

И используется в главном цикле while метода run().

Как можно заметить в выводе (см. код 3 выделенное жёлтым маркером — значение ), это суммарное квадратичное отклонение по ходу алгоритма уменьшается. А когда оно перестанет изменяться, алгоритм завершает свою работу. Это означает, что группы (кластеры) упакованы как можно компактнее в этой ситуации. Дальше уплотнять уже кластеры н*е*куда.

На рисунке 3 изображено графическое представление вывода (код 3).

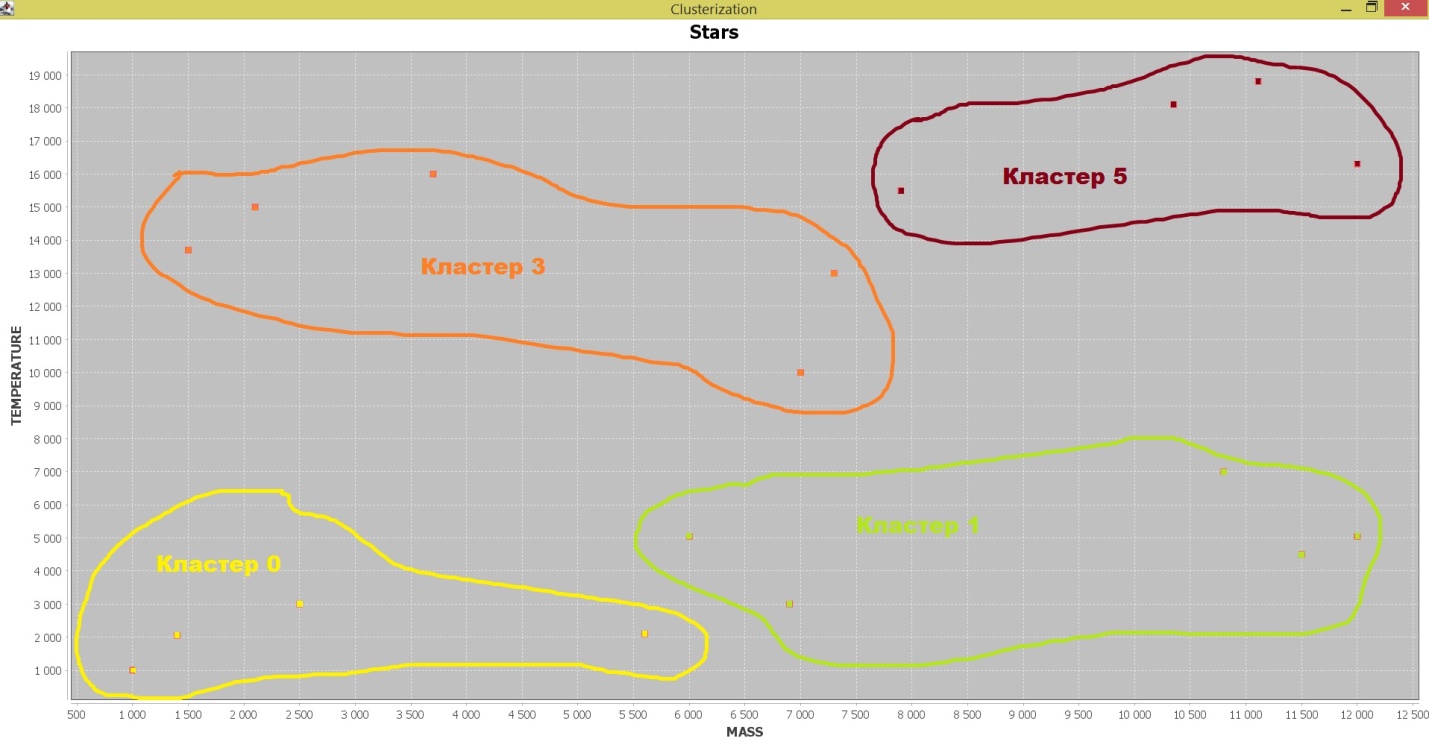


Рисунок 3 — Первый результат работы программы k-means

|  |
| --- |
| 0)  Temperature: 1000  Mass: 1000.0  Cluster: 0  1)  Temperature: 2050  Mass: 1400.0  Cluster: -1  2)  Temperature: 3000  Mass: 2500.0  Cluster: -1  3)  Temperature: 2100  Mass: 5600.0  Cluster: -1  4)  Temperature: 5050  Mass: 6000.0  Cluster: -1  5)  Temperature: 3000  Mass: 6900.0  Cluster: 1  6)  Temperature: 4500  Mass: 11500.0  Cluster: -1  7)  Temperature: 5050  Mass: 12000.0  Cluster: -1  8)  Temperature: 7000  Mass: 10800.0  Cluster: 2  9)  Temperature: 18100  Mass: 10350.0  Cluster: 5  10)  Temperature: 18800  Mass: 11110.0  Cluster: -1  11)  Temperature: 16300  Mass: 12000.0  Cluster: -1  12)  Temperature: 10000  Mass: 7000.0  Cluster: -1  13)  Temperature: 13000  Mass: 7300.0  Cluster: -1  14)  Temperature: 15500  Mass: 7900.0  Cluster: 4  15)  Temperature: 13700  Mass: 1500.0  Cluster: -1  16)  Temperature: 15000  Mass: 2100.0  Cluster: 3  17)  Temperature: 16000  Mass: 3700.0  Cluster: -1  i = 0 ( 1633.3333333333333 ; 2016.6666666666667 ) cluster: 0.0  i = 1 ( 6166.666666666667 ; 3383.3333333333335 ) cluster: 1.0  i = 2 ( 10325.0 ; 6637.5 ) cluster: 2.0  i = 3 ( 2433.3333333333335 ; 14900.0 ) cluster: 3.0  i = 4 ( 7600.0 ; 14250.0 ) cluster: 4.0  i = 5 ( 11153.333333333334 ; 17733.333333333332 ) cluster: 5.0  v = 5.196391041666666E7  i = 0 ( 1633.3333333333333 ; 2016.6666666666667 ) cluster: 0.0  i = 1 ( 6166.666666666667 ; 3383.3333333333335 ) cluster: 1.0  i = 2 ( 11433.333333333334 ; 5516.666666666667 ) cluster: 2.0  i = 3 ( 2433.3333333333335 ; 14900.0 ) cluster: 3.0  i = 4 ( 7400.0 ; 12833.333333333334 ) cluster: 4.0  i = 5 ( 11153.333333333334 ; 17733.333333333332 ) cluster: 5.0  v = 3.836906666666666E7  i = 0 ( 1633.3333333333333 ; 2016.6666666666667 ) cluster: 0.0  i = 1 ( 6166.666666666667 ; 3383.3333333333335 ) cluster: 1.0  i = 2 ( 11433.333333333334 ; 5516.666666666667 ) cluster: 2.0  i = 3 ( 2433.3333333333335 ; 14900.0 ) cluster: 3.0  i = 4 ( 7400.0 ; 12833.333333333334 ) cluster: 4.0  i = 5 ( 11153.333333333334 ; 17733.333333333332 ) cluster: 5.0  v = 3.836906666666666E7  0)  Temperature: 1000  Mass: 1000.0  Cluster: 0  1)  Temperature: 2050  Mass: 1400.0  Cluster: 0  2)  Temperature: 3000  Mass: 2500.0  Cluster: 0  3)  Temperature: 2100  Mass: 5600.0  Cluster: 1  4)  Temperature: 5050  Mass: 6000.0  Cluster: 1  5)  Temperature: 3000  Mass: 6900.0  Cluster: 1  6)  Temperature: 4500  Mass: 11500.0  Cluster: 2  7)  Temperature: 5050  Mass: 12000.0  Cluster: 2  8)  Temperature: 7000  Mass: 10800.0  Cluster: 2  9)  Temperature: 18100  Mass: 10350.0  Cluster: 5  10)  Temperature: 18800  Mass: 11110.0  Cluster: 5  11)  Temperature: 16300  Mass: 12000.0  Cluster: 5  12)  Temperature: 10000  Mass: 7000.0  Cluster: 4  13)  Temperature: 13000  Mass: 7300.0  Cluster: 4  14)  Temperature: 15500  Mass: 7900.0  Cluster: 4  15)  Temperature: 13700  Mass: 1500.0  Cluster: 3  16)  Temperature: 15000  Mass: 2100.0  Cluster: 3  17)  Temperature: 16000  Mass: 3700.0  Cluster: 3 |

Это идеальный для поставленной в этой работе задачи результат! Выделены все шесть кластеров, и объекты сгруппированы в них, как того хотелось изначально.

Графически результат выглядит так, как показано на рисунке ниже.

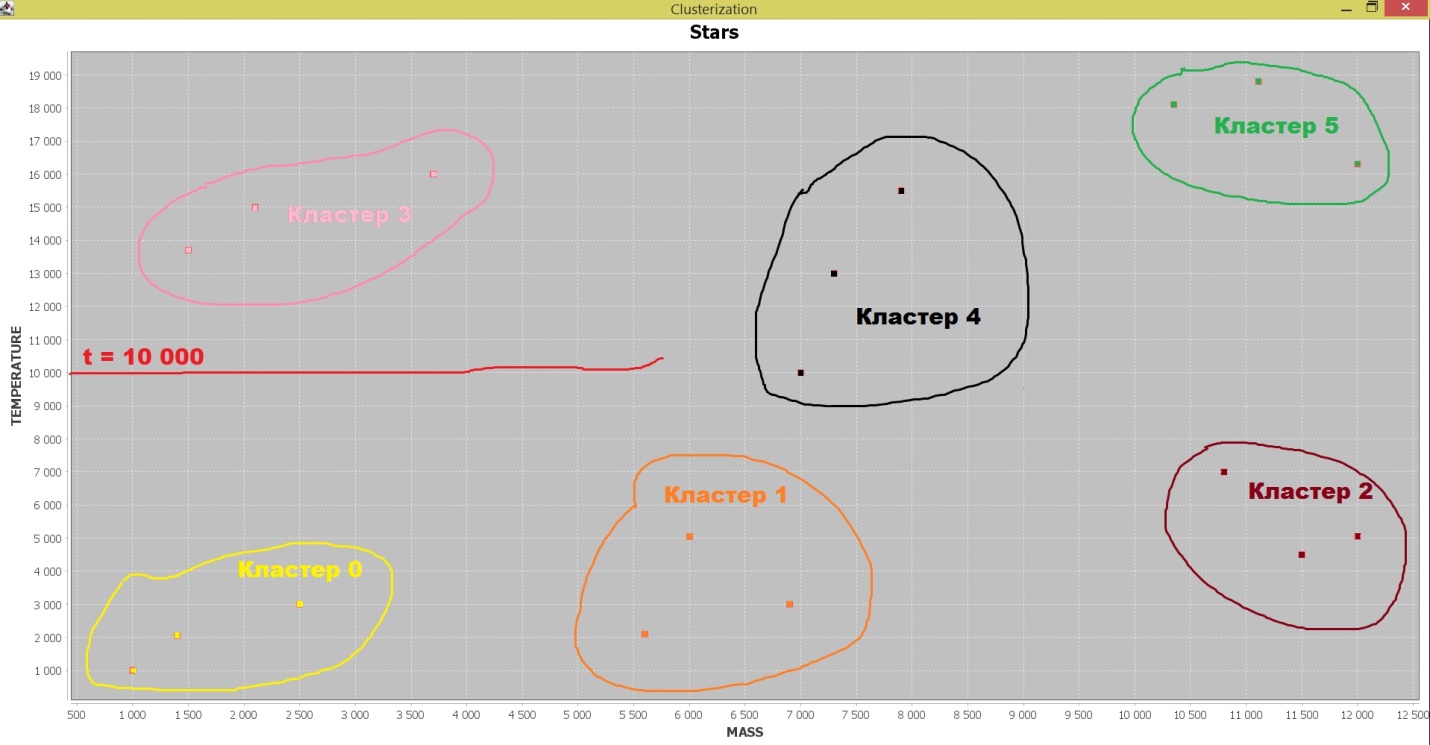


Рисунок 4 — Второй тест алгоритма k-means

# Вывод

Благодаря проделанной работе вы должны понять суть алгоритма кластеризации k-means. Его характерные особенности, в том числе стохастичность (случайность) и условие остановки.

Необходимо также понимать, зачем нужна кластеризация и в каких областях её реализуют. Она нужна для того, чтобы лучше понять представленные в какой-либо области деятельности данные и группировать их по их отличительным свойствам. Это может быть биология, география, понятия из математических и компьютерных наук, история и многое-многое другое.

# Список использованной литературы

1. k-means++ // Википедия. Свободная энциклопедия URL: https://ru.wikipedia.org/wiki/K-means%2B%2B (дата обращения: 18.01.2022).
2. Метод k-средних // Википедия. Свободная энциклопедия URL: https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4\_k-%D1%81%D1%80%D0%B5%D0%B4%D0%BD%D0%B8%D1%85 (дата обращения: 18.01.2022).
3. JFreeChart // GitHub URL: https://github.com/jfree/jfreechart (дата обращения: 18.01.2022).
4. Лебедев А.С. Лекция 1 - Технологии анализа данных. Методы и средства извлечения знаний // Лекции по дисциплине "Информационные хранилища и информационно-аналитические системы". - М.: 2021
5. Кластеризация: метод k-средних // Портал Знаний. Глобальный интеллектуальный ресурс. StatSoft URL: http://statistica.ru/theory/klasterizatsiya-metod-k-srednikh/#:~:text=%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%20k%2D%D1%81%D1%80%D0%B5%D0%B4%D0%BD%D0%B8%D1%85%20%E2%80%93%20%D1%8D%D1%82%D0%BE%20%D0%BC%D0%B5%D1%82%D0%BE%D0%B4,(%D1%86%D0%B5%D0%BD%D1%82%D1%80%D0%BE%D0%B8%D0%B4%D1%83)%20%D0%BA%D0%BE%D1%82%D0%BE%D1%80%D0%BE%D0%B3%D0%BE%20%D0%BE%D0%BD%D0%BE%20%D0%B1%D0%BB%D0%B8%D0%B6%D0%B5%20%D0%B2%D1%81%D0%B5%D0%B3%D0%BE (дата обращения: 18.01.2022).
6. Плетнёва Мария, Конюшко Анастасия КЛАСТЕРИЗАЦИЯ. МЕТОДЫ K-MEANS, FUZZY C-MEANS // Презентация. – 2021

# Приложение

Код 5 — Класс KMeans

|  |
| --- |
| package mirea.student.shayko;  import java.util.\*;  public class KMeans {  **public Universe sample;**  **public final int CLUSTERS\_NUM = 6;**   **private int[] centers;**   public KMeans() {  this(new Universe());  }   public KMeans(Universe uni) {  sample = uni;  this.centers = this.init();   int[] temperature = sample.getTemp();  float[] mass = sample.getMass();  for (int i = 0; i < this.centers.length; ++i)  if (temperature[centers[i]] < 10000) { *// красные звёзды...* if (mass[centers[i]] < 4000.0)  sample.changeCluster(centers[i], 0); *// красная звезда-долгожитель* else if (mass[centers[i]] < 8000.0)  sample.changeCluster(centers[i], 1); *// красная звезда-среднежитель* else if (mass[centers[i]] <= 12000.0)  sample.changeCluster(centers[i], 2); *// красная звезда с относительно непродолжительным жизненным циклом* } else { *// голубые звёзды...* if (mass[centers[i]] < 4000.0)  sample.changeCluster(centers[i], 3); *// голубая звезда-долгожитель* else if (mass[centers[i]] < 8000.0)  sample.changeCluster(centers[i], 4); *// голубая звезда-среднежитель* else if (mass[centers[i]] <= 12000.0)  sample.changeCluster(centers[i], 5); *// голубая звезда с относительно непродолжительным жизненным циклом* }  }   public int[] getCenters() {  int[] result = new int[CLUSTERS\_NUM];  System.*arraycopy*(this.centers, 0, result, 0, this.centers.length);   return result;  }   *// k-means++ (see https://ru.wikipedia.org/wiki/K-means%2B%2B)  // Определить все начальные центроиды алгоритма (улучшенным способом, а не просто случайной выборкой):* private int[] init() {  Random random = new Random(); *// генератор псевдослучайных* int[] centers = new int[CLUSTERS\_NUM]; *// массив с индексами изначальных центроидов алгоритма k-means* Arrays.*fill*(centers, -2);  int counter = 0; *// счётчик заполнения массива (см. выше) новыми центроидами* centers[counter++] = random.nextInt(sample.count()-1); *// первый центроид выбирается случайным образом* int[] temperatures = sample.getTemp(); *// массив температур звёзд* float[] masses = sample.getMass(); *// массив масс звёзд Вселенной* int[] clusters = sample.getClusters(); *// массив кластеров каждой звезды   // Поэтапно определить все необходимые центроиды:* while (counter < this.CLUSTERS\_NUM) {  double[] min\_distances = new double[sample.count()]; *// расстояние каждой точки до её ближайшего центроида* double[][] distances = new double[sample.count()][sample.count()]; *// расстояние каждой точки до всех центроидов (определённых на данный момент)* double sum = 0; *// сумма расстояний до ближайших центроидов (понадобится в конце цикла)* for (int i = 0; i < temperatures.length; ++i) {  double min = -1; *// минимальное расстояние среди всех расстояний у текущей точки (точка = звезда)* for (int j = 0; j < counter; ++j) {  distances[i][j] = (temperatures[centers[j]] - temperatures[i]) \* (temperatures[centers[j]] - temperatures[i]) + (masses[centers[j]] - masses[i]) \* (masses[centers[j]] - masses[i]);  if (min == -1)  min = distances[i][j];  else if (min > distances[i][j])  min = distances[i][j];   }  min\_distances[i] = min;  sum += min;  }   *// Выбрать следующий случайный центроид, опираясь уже на проставленные (см. выше) вероятности выбора:* double rnd = random.nextDouble() \* sum;  sum = 0;   boolean flag = false;  for (int i = 0; i < temperatures.length; ++i) {  sum += min\_distances[i];  if (sum > rnd) {  centers[counter++] = i;  break;  }  }  }   return centers;  }   private double V(double[][] centers) {  int[] temperatures = sample.getTemp(); *// массив температур звёзд* float[] masses = sample.getMass(); *// массив масс звёзд Вселенной* int[] clusters = sample.getClusters(); *// массив кластеров каждой звезды* double sum = 0;   for (int i = 0; i < centers.length; ++i)  for (int j = 0; j < sample.count(); ++j) {  if (clusters[j] == i)  sum += (centers[i][0] - masses[j]) \* (centers[i][0] - masses[j]) + (centers[i][1] - temperatures[j]) \* (centers[i][1] - temperatures[j]);  }   return sum;  }   public void run() {  int[] temperatures = sample.getTemp(); *// массив температур звёзд* float[] masses = sample.getMass(); *// массив масс звёзд Вселенной* int[] clusters = sample.getClusters(); *// массив кластеров каждой звезды* for (int i = 0; i < sample.count(); ++i)  if (clusters[i] == -1) {  double distance; *// расстояние текущей точки до следующего центроида* double min = -1; *// минимальное расстояние среди всех расстояний у текущей точки (точка = звезда)* for (int j = 0; j < this.centers.length; ++j) {  distance = (temperatures[this.centers[j]] - temperatures[i]) \* (temperatures[this.centers[j]] - temperatures[i]) + (masses[this.centers[j]] - masses[i]) \* (masses[this.centers[j]] - masses[i]);  if (min == -1) {  min = distance;  sample.changeCluster(i, clusters[this.centers[j]]);  }  else if (min > distance) {  min = distance;  sample.changeCluster(i, clusters[this.centers[j]]);  }  }  }   double v = -1;  double v\_past = -1;  while (true) {  v\_past = v;   double[][] centers = new double[this.CLUSTERS\_NUM][3];  clusters = sample.getClusters();  for (int i = 0; i < this.CLUSTERS\_NUM; ++i) {  double temp\_sum = 0;  double mass\_sum = 0;  int counter = 0;   for (int j = 0; j < this.sample.count(); ++j) {  if (clusters[j] == i) {  temp\_sum += temperatures[j];  mass\_sum += masses[j];  ++counter;  }  }   if (counter == 0) {  centers[i][0] = -10000;  centers[i][1] = -10000;  continue;  }  centers[i][0] = mass\_sum/counter;  centers[i][1] = temp\_sum/counter;  centers[i][2] = i;  System.*out*.println("i = " + i + "\t( " + centers[i][0] + " ; " + centers[i][1] + " )\t" + "cluster: " + centers[i][2]);  }   for (int i = 0; i < sample.count(); ++i) {  double distance; *// расстояние текущей точки до следующего центроида* double min = -1; *// минимальное расстояние среди всех расстояний у текущей точки (точка = звезда)* for (int j = 0; j < centers.length; ++j) {  distance = (centers[j][1] - temperatures[i]) \* (centers[j][1] - temperatures[i]) + (centers[j][0] - masses[i]) \* (centers[j][0] - masses[i]);  if (min == -1) {  min = distance;  sample.changeCluster(i, (int)centers[j][2]);  }  else if (min > distance) {  min = distance;  sample.changeCluster(i, (int)centers[j][2]);  }  }  }   v = V(centers);  System.*out*.println("v = " + v);  if (v\_past == v)  break;  }  }  } |

Код 6 — Класс Universe

|  |
| --- |
| package mirea.student.shayko;  public class Universe {  **private String[] stars**; *// коллекция звёзд во Вселенной   // Конструкторы:* public Universe() {  this(new String[] {});  }   public Universe(String[] values) {  this.stars = new String[values.length];  System.*arraycopy*(values, 0, this.stars, 0, values.length);  }   *// Получить "сырой" массив stars:* public String[] getStars() {  String[] result = new String[this.stars.length];  System.*arraycopy*(this.stars, 0, result, 0, this.stars.length);  return result;  }   *// Получить температуру всех звёзд в виде массива целых:* public int[] getTemp() {  int temp[] = new int[this.stars.length];  for (int i = 0; i < this.stars.length; ++i) {  int limit = stars[i].indexOf(':');  String temp\_str = "";  for (int j = 0; j < limit; ++j)  temp\_str += stars[i].charAt(j);  try {  temp[i] = Integer.*parseInt*(temp\_str);  } catch (NumberFormatException ex) {  System.*out*.println(ex.getMessage() + "\n");  System.*out*.println(ex.getStackTrace());  temp[i] = -1;  }  }   return temp;  }   *// Получить массы всех звёзд в виде массива дробных (float):* public float[] getMass() {  float mass[] = new float[this.stars.length];  for (int i = 0; i < this.stars.length; ++i) {  int limit = stars[i].indexOf(':');  int up\_limit = stars[i].indexOf(';');  String temp\_str = "";  for (int j = limit + 1; j < up\_limit; ++j)  temp\_str += stars[i].charAt(j);  try {  mass[i] = Float.*parseFloat*(temp\_str);  } catch (NumberFormatException ex) {  System.*out*.println(ex.getMessage() + "\n");  System.*out*.println(ex.getStackTrace());  mass[i] = -1.0f;  }  }   return mass;  }   *// Получить массив кластеров каждой звезды:* public int[] getClusters() {  int[] clusters = new int[this.stars.length];   for (int i = 0; i < this.stars.length; ++i) {  int beg = this.stars[i].indexOf(';') + 1;  try {  clusters[i] = Integer.*parseInt*(this.stars[i].substring(beg));  } catch (NumberFormatException ex) {  System.*out*.println(ex.getMessage() + "\n");  System.*out*.println(ex.getStackTrace());  clusters[i] = -1;  }  }   return clusters;  }   *// Вернуть число звёзд:* public int count() {  return this.stars.length;  }   *// Изменить кластер звезды с индексом index:* public void changeCluster(int index, int cluster\_number) {  if ((index < 0)||(index >= this.stars.length)) {  System.*out*.println("Failed to change clusters! Wrong star index!");  return;  }   String edit\_star = this.stars[index];   try {  if (cluster\_number == Integer.*parseInt*(edit\_star.substring(edit\_star.indexOf(';') + 1)))  return;  } catch (NumberFormatException ex) {  System.*out*.println(ex.getMessage() + "\n");  System.*out*.println(ex.getStackTrace());  System.*out*.println("Failed to change clusters! Cluster number error in stars data:\n" + (index + 1) + " star:" + edit\_star);  }   String result\_star = "" + edit\_star.substring(0, edit\_star.indexOf(';') + 1) + cluster\_number;   this.stars[index] = result\_star;  }   public String toString() {  String result = "";  for (int i = 0; i < this.stars.length; ++i)  result += i + ")\n\t" + "Temperature: " + getTemp()[i] + "\n\tMass: " + getMass()[i] + "\n\tCluster: " + getClusters()[i] + "\n\n";   return result;  } } |