ДИСЦИПЛИНА **Интеллектуальные системы и технологии**

(полное наименование дисциплины без сокращений)

ИНСТИТУТ **информационных технологий**

КАФЕДРА **корпоративных информационных систем**

полное наименование кафедры

ВИД УЧЕБНОГО МАТЕРИАЛА **Учебно-методическое пособие**

(в соответствии с пп.1-11)

ПРЕПОДАВАТЕЛЬ **Демидова Лилия Анатольевна**

(фамилия, имя, отчество)

СЕМЕСТР **1 семестр (осенний), 2024 – 2025 учебный год**

(семестр обучения, учебный год)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
МИРЭА – РОССИЙСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

КЛАСТЕРНЫЙ АНАЛИЗ. PYTHON

Москва – 2022

УДК 004
ББК 32.97
Д 30
Демидова Л.А. Кластерный анализ. Python [Электронный ресурс]: Учебное пособие / Демидова Л.А. — М.: МИРЭА – Российский технологический университет, 2022. — 1 электрон. опт. диск (CD-ROM)

Аннотация учебного пособия.

В учебном пособии рассматриваются аспекты решения задач кластерного анализа данных средствами языка Python. Предлагается перечень заданий, при выполнении которых применяются различные алгоритмы кластерного анализа. Предназначено для магистрантов, изучающих дисциплину «Интеллектуальные системы и технологии» по направлению 09.04.04 Программная инженерия.

Учебное пособие издается в авторской редакции.

Автор: Демидова Лилия Анатольевна

Рецензенты:

Андреева О.Н., д.т.н., доцент, начальник отдела научной работы АО «Концерн «Моринсис-Агат»

Коняев Г.Б., к.т.н., президент ООО «ЭН-ЭС-ДЖИ»

Минимальные системные требования:
Наличие операционной системы Windows, поддерживаемой производителем.
Наличие свободного места в оперативной памяти не менее 128 Мб.
Наличие свободного места в памяти хранения (на жестком диске) не менее 30 Мб.
Наличие интерфейса ввода информации.
Дополнительные программные средства: программа для чтения pdf-файлов (Adobe Reader).
Подписано к использованию по решению Редакционно-издательского совета
МИРЭА – Российского технологического университета от ___ _____ 2021 г.
Объем ___ Мб
Тираж 10

Оглавление

ВВЕДЕНИЕ 5
КЛАСТЕРНЫЙ АНАЛИЗ 6
1. Основные понятия 6
2. Иерархические алгоритмы кластерного анализа 10
2.1. Иерархические агломеративные алгоритмы кластеризации 13
2.2. Программная реализация алгоритмов иерархической кластеризации на языке Python 19
2.3. Пример иерархического кластерного анализа 25
3. Итерационные алгоритмы кластерного анализа 38
3.1. Алгоритм *k*-средних (*k*-means-алгоритм) 39
3.2. Алгоритм нечетких c-средних (fuzzy c-means-алгоритм) 41
3.3. Алгоритм DBSCAN 43
3.4. Оценка числа кластеров. Показатели качества кластеризации 53
3.5. Программная реализация *k*-means-алгоритма на языке Python 59
3.6. Программная реализация fuzzy-*c*-means-алгоритма на языке Python 65
3.7. Программная реализация DBSCAN-алгоритма на языке Python 66
3.8. Программная реализация показателей качества кластеризации 68
3.9. Пример кластерного анализа с применением *k*-means-алгоритма 69
3.10. Пример кластерного анализа с применением fuzzy-*c*-means-алгоритма 73
3.11. Пример кластерного анализа с применением DBSCAN-алгоритма 76
4. Сложности и проблемы, которые могут возникнуть при выполнении кластерного анализа 81
4.1. Проблема выбора показателя качества кластеризации 82
4.2. Общие выводы по результатам экспериментов с набором данных «Ирисы Фишера» (The Iris Dataset) 92
ВАРИАНТЫ И ЗАДАНИЯ 95
СПИСОК ЛИТЕРАТУРЫ 98

ВВЕДЕНИЕ

Учебное пособие излагает принципы кластерного анализа данных с применением иерархических и итерационных алгоритмов.

Рассматриваются:

– иерархические агломеративные алгоритмы кластеризации, использующие различные метрики для вычисления расстояний между объектами анализируемого набора данных, и различные методы формирования кластеров, позволяющие выполнять визуализацию результатов кластеризации на основе дендрограмм;

– итерационные алгоритмы кластеризации, в частности, *k*-means алгоритм и алгоритм нечетких-*c*-средних, реализующие уточнение координат центроидов кластеров и кластерной принадлежности объектов анализируемого набора данных, а также DBSCAN-алгоритм, реализующий формирование кластеров с учетом плотности расположения объектов в пространстве.

– рассматриваются показатели качества кластеризации, применяемые для оценки оптимального числа кластеров, скрытых в анализируемом наборе данных.

Приводятся примеры, демонстрирующие принципы работы алгоритмов кластерного анализа.

Читателю предлагается перечень заданий: каждое из них содержит 2 варианта набора данных из репозитория данных для машинного обучения, для которых необходимо реализовать кластерный анализ с использованием предлагаемых алгоритмов кластеризации, реализованных в программных библиотеках языка Python. При этом в одном из вариантов известны истинные метки классов (кластеров) для объектов, а в другом – нет.

Читатель может получить дополнительные сведения по теоретическим вопросам, а также по аспектам программной реализации для рассматриваемых алгоритмов кластеризации в источниках, указанных в списке литературы.

КЛАСТЕРНЫЙ АНАЛИЗ

1. Основные понятия

Кластерный анализ – совокупность методов, алгоритмов, подходов и процедур, разработанных для решения проблемы формирования однородных (гомогенных) групп объектов с учетом характеризующих их признаков в произвольной проблемной области.

При выявлении таких групп – кластеров (clusters) – используется неформальное предположение о том, что объекты, относимые к одному кластеру, должны иметь большее сходство между собой, чем с объектами из других кластеров [1, 2].

При выполнении кластерного анализа исследуется некоторый набор данных X, содержащий информацию об объектах той или иной проблемной области. Каждая строка набора данных X соответствует некоторому объекту x_i ($i=\overline{1,n}$, n – число объектов в наборе данных X), описываемому априори заданным набором признаков. Пусть p – число признаков. Тогда каждому объекту x_i сопоставлен p-мерный вектор, содержащий значения признаков. При этом каждый столбец набора данных X соответствует некоторому j-му ($j=\overline{1,p}$) признаку. Признаки могут быть числовыми или нечисловыми (категориальными).

Таким образом, анализируемый набор данных может быть представлен так называемой *матрицей объектов-признаков размера* $n\times p$, в которой строки матрицы – признаковые описания объектов, а столбцы матрицы – признаки:

$$F=(f_j(x_i))_{n\times p}=\begin{bmatrix} f_1(x_1) & \dots & f_p(x_1) \\ \dots & \dots & \dots \\ f_1(x_n) & \dots & f_p(x_n) \end{bmatrix}.$$

При выявлении кластеров, скрытых в некотором анализируемом наборе данных (наборе объектов), следует соблюдать следующие условия:

– каждый кластер должен представлять собой концептуально однородную категорию и содержать похожие объекты с близкими значениями признаков;

– совокупность всех кластеров должна быть исчерпывающей, то есть охватывать все объекты анализируемого набора данных;

– кластеры должны быть взаимно исключающими (ни один объект не должен одновременно принадлежать двум различным кластерам).

Под задачей кластерного анализа некоторого набора объектов понимается задача нахождения некоторого теоретико-множественного разбиения этого набора на непересекающиеся группы – кластеры – таким образом, чтобы объекты, относимые к одному кластеру, отличались друг от друга в значительно меньшей степени, чем объекты из разных кластеров.

Методы и алгоритмы кластерного анализа используются при поиске закономерностей в больших наборах многомерных данных. При этом проблема кластерного анализа приобретает самостоятельное значение в контексте интеллектуального анализа данных (Data Mining).

Кластеризация – это:

– группировка объектов по похожести значений их признаков: каждый кластер состоит из схожих объектов, а объекты разных кластеров существенно отличаются;

– процедура, которая любому объекту x_i некоторого набора данных X ставит в соответствие метку кластера $y_i \in Y = \{1, 2, ..., c\}$, где $i = \overline{1,n}$, n – число объектов в наборе данных X, Y – набор меток, c – число кластеров.

Постановка задачи кластеризации сложна и неоднозначна, так как:

– число кластеров в общем случае неизвестно;

– выбор меры «похожести» или близости объектов между собой, как и выбор критерия (показателя) качества кластеризации, носит субъективный характер.

Кластерный анализ предназначен для разбиения набора объектов на поддающиеся интерпретации группы (кластеры), таким образом, чтобы объекты, входящие в одну группу были максимально «схожи», а объекты из разных групп были максимально «отличными» друг от друга.

В то время как **в факторном анализе** группируются *столбцы* исследуемого набора данных с целью анализа структуры множества признаков и выявления обобщенных факторов, **в кластерном анализе** группируются *строки* исследуемого набора данных с целью анализа структуры множества объектов.

Кластерный анализ позволяет выполнить **классификацию объектов**, назначив объектам метки кластеров.

Каждый объект рассматривается как точка в пространстве признаков.

Задача кластерного анализа – выделение «сгущений» точек, каждой из которых соответствует строка исследуемого набора данных, на однородные группы (кластеры) объектов.

Задача кластерного анализа может рассматриваться как задача распознавания образов с применением **обучения без учителя**.

Обучение без учителя (unsupervised learning) – раздел машинного обучения, изучающий широкий класс задач обработки данных, в которых известны только описания множества объектов (обучающей выборки), и требуется обнаружить внутренние взаимосвязи, зависимости, закономерности, существующие между объектами.

Обучение без учителя часто противопоставляется **обучению с учителем**, когда для каждого обучающего объекта известен «правильный ответ», и требуется найти правило (зависимость) между объектами и ответами.

В задаче классификации (при обучении с учителем) известны метки классов для всех объектов обучающей выборки, требуется классифицировать объекты тестовой выборки (новые объекты) – определить метки классов для всех объектов тестовой выборки (для новых объектов).

В задаче кластеризации (при обучении без учителя) ни для одного объекта обучающей выборки метка кластера неизвестна, тем не менее, требуется классифицировать все объекты обучающей выборки – назначить им метки кластеров.

Цели кластеризации

1. Упростить дальнейшую обработку данных, разбив набор объектов на группы схожих объектов, чтобы работать с каждой группой в отдельности. В этом случае можно говорить о задачах классификации, регрессии, прогнозирования.
2. Сократить объем хранимых данных, оставив по одному представителю от каждого кластера. В этом случае можно говорить о задачах сжатия данных.
3. Выделить нетипичные объекты, которые не подходят ни к одному из кластеров. В этом случае можно говорить о задачах одноклассовой классификации.
4. Построить иерархию множества объектов. В этом случае можно говорить о задачах таксономии.

Задачи, для решения которых используется кластеризация

1. *Изучение данных.* Разбиение набора данных на схожие группы помогает выявить структуру данных, увеличить наглядность их представления, выдвинуть новые гипотезы, понять, насколько информативны признаки объектов.
2. *Облегчение анализа данных.* При помощи кластеризации можно упростить дальнейшую обработку данных и построение моделей: каждый кластер обрабатывается индивидуально – модель создается для каждого кластера в отдельности. При этом кластеризация является подготовительным этапом перед

решением других задач Data Mining, например, для классификации, регрессии, поиска ассоциации, поиска последовательных шаблонов и т.п.

3. *Сжатие данных.* В случае, когда данные имеют большой объем (сотни тысяч и миллионы строк), кластеризация позволяет сократить объем хранимых данных, оставив по одному наиболее типичному представителю от каждого кластера.

4. *Обнаружение аномалий.* Кластеризация применяется для выделения нетипичных объектов, которые не присоединяются ни к одному из кластеров. Эту задачу также называют задачей обнаружения аномалий (outlier detection). При этом интерес представляют кластеры (группы), в которые попадает крайне мало объектов, например, 1 – 3 объекта.

5. *Прогнозирование.* Кластеры используются не только для краткого описания имеющихся объектов, но и для распознавания новых. Каждый новый объект относится к тому кластеру, присоединение к которому наилучшим образом удовлетворяет критерию (показателю) качества кластеризации. При этом можно прогнозировать поведение объекта, предположив, что оно будет схожим с поведением других объектов, принадлежащих кластеру.

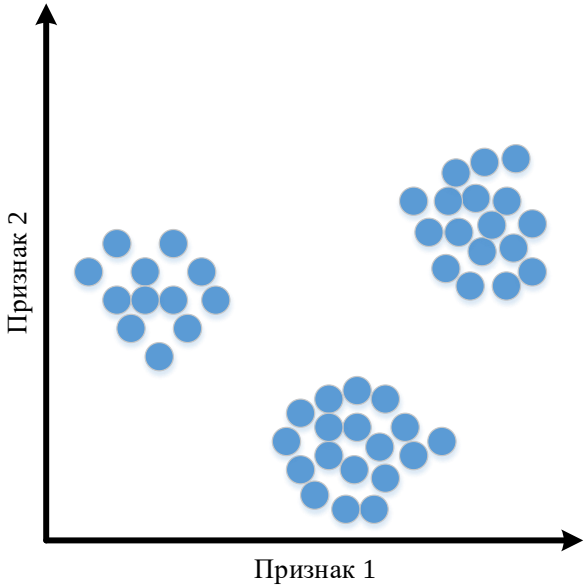


Рисунок 1. Представление данных в теории (в идеальном случае)

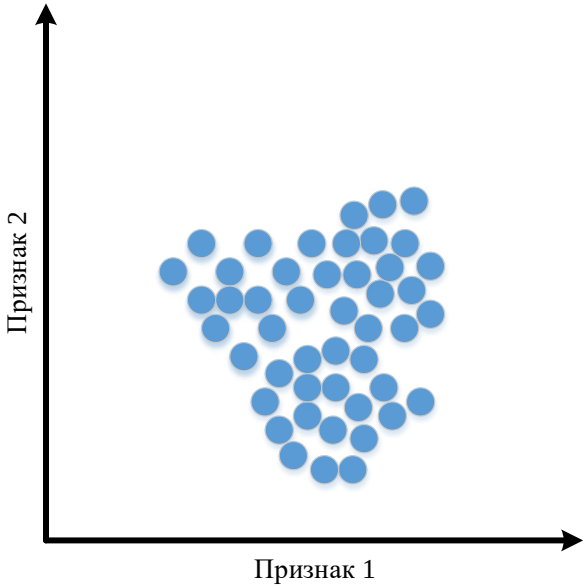


Рисунок 2. Представление данных на практике

Представление данных в двумерном пространстве, рассматриваемое в задачах кластерного анализа, в теории (в идеальном случае, когда кластеры компакты и хорошо отделимы друг от друга) и на практике показано на рис. 1 и 2 соответственно. Очевидно, что даже в двумерном случае задача кластеризации не является тривиальной. При этом в каждом конкретном случае требуется

адекватно определить не только число кластеров, скрытых в наборе данных, но и границы кластеров.

В настоящее время известно большое число методов и алгоритмов кластеризации. Однако не существует универсального метода (алгоритма) кластеризации. Каждый из них имеет свои ограничения и выделяет кластеры лишь некоторых типов: например, гиперсферические, цепочечные и т.п. При этом понятие «тип кластерной структуры» не имеет формального определения. На рис. 3 приведены некоторые примеры различных типов кластерных структур.

Выявление кластеров может быть реализовано с применением *иерархических алгоритмов* кластерного анализа и *итерационных алгоритмов* кластерного анализа. Обычно иерархическую кластеризацию реализуют при малом числе объектов, а итерационную – при большом числе объектов. В общем виде этапы процесса кластерного анализа представлены на рис. 4.

Постановка задачи кластеризации

Дано:

набор объектов $X = \{x_1, x_2, ..., x_n\}$ и функция расстояния между ними (т.е. все признаки оцениваются количественно).

Требуется:

разбить набор X на непересекающиеся кластеры так, чтобы каждый кластер состоял из похожих объектов, а объекты разных кластеров существенно различались.

2. Иерархические алгоритмы кластерного анализа

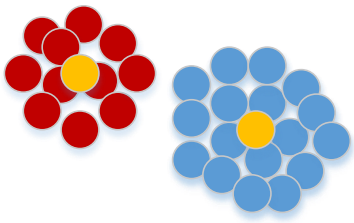
Алгоритмы иерархической кластеризации, называемые также алгоритмами таксономии, – это алгоритмы упорядочивания, реализующие создание иерархии (дерева) вложенных кластеров для объектов анализируемого набора данных [3].

Выделяют *дивизимные* и *агломеративные* алгоритмы иерархической кластеризации.

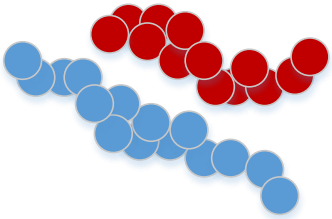
Дивизимные (нисходящие) алгоритмы (divisive algorithms) создают новые кластеры посредством деления более крупных кластеров на более мелкие (при этом дерево иерархии формируется от ствола к листьям).

Агломеративные (восходящие) алгоритмы (agglomerative algorithms) создают новые кластеры посредством объединения более мелких кластеров (при этом дерево иерархии формируется от листьев к стволу).

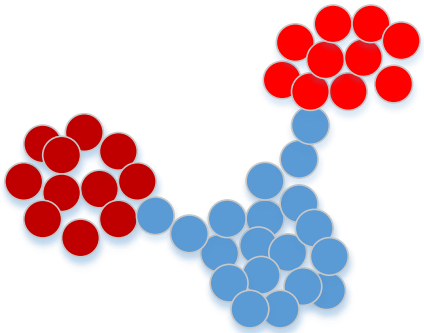
Наибольшее применение в решении различных прикладных задач находят агломеративные алгоритмы кластеризации.



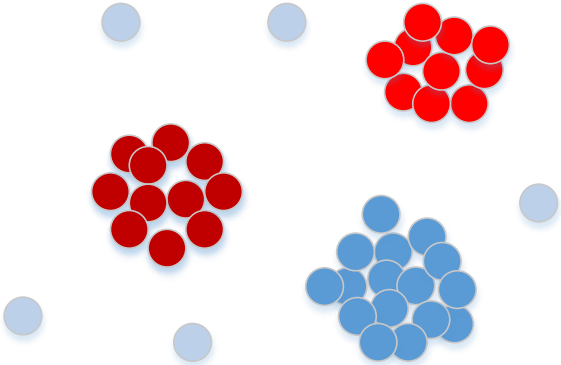
Кластеры с центром



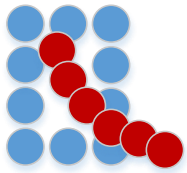
Ленточные (цепочечные) кластеры



Кластеры с перемычками



Кластеры, наложенные на фон из редко расположенных объектов



Кластеры, образованные не по сходству, а с учетом иного типа регулярности

Отсутствие кластеров

Рисунок 3. Примеры различных типов кластерных структур

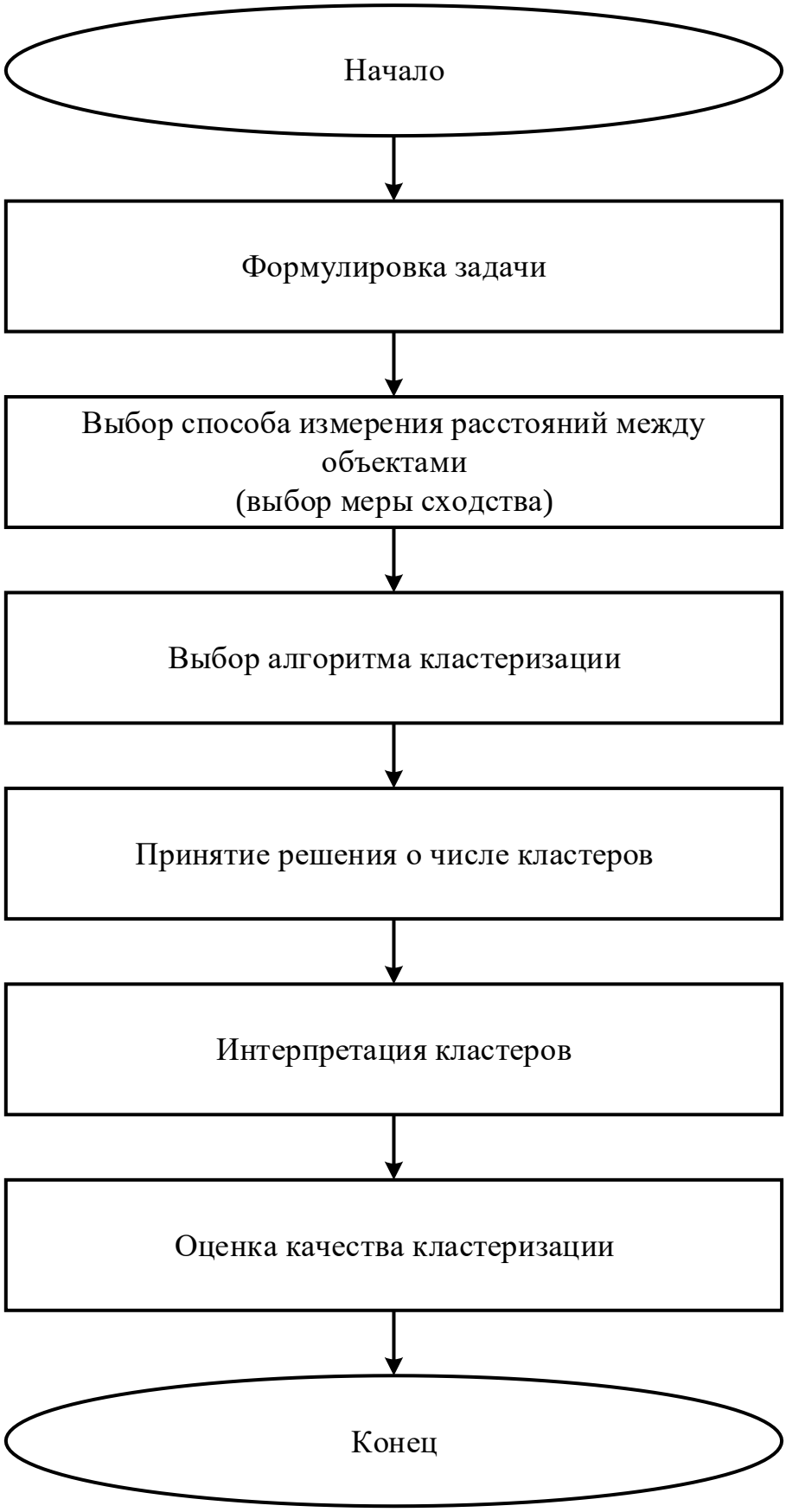


Рисунок 4. Этапы кластерного анализа

2.1. Иерархические агломеративные алгоритмы кластеризации

При реализации агломеративного алгоритма кластеризации сначала отдельным кластером считается каждый объект. При этом для любых двух одноэлементных кластеров U и V, состоящих соответственно из объектов x_i и x_j ($i = \overline{1,n}$; $j = \overline{1,n}$; n – число объектов), расстояние между кластерами вычисляется с помощью метрики расстояния [3]:

$$D(U,V) = d(x_i, x_j), \quad (1)$$

где в качестве метрики $d(x_i, x_j)$ может быть выбрана та или иная метрика вычисления расстояния между объектами.

Так, например, для вычисления расстояния между объектами $x_i = (f_1(x_i), f_2(x_i), \ldots, f_g(x_i))$ и $x_j = (f_1(x_j), f_2(x_j), \ldots, f_g(x_j))$ в g-мерном пространстве могут быть использованы такие метрики расстояний (рассматриваемые в роли мер сходства), как:

– метрика евклидова расстояния:

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{g} (f_r(x_i) - f_r(x_j))^2}\ ; \quad (2)$$

– метрика квадрата евклидова расстояния:

$$d(x_i, x_j) = \sum_{r=1}^{g} (f_r(x_i) - f_r(x_j))^2\ ; \quad (3)$$

– метрика манхэттенского расстояния:

$$d(x_i, x_j) = \sum_{r=1}^{g} | f_r(x_i) - f_r(x_j) |; \quad (4)$$

– метрика расстояния Чебышева:

$$d(x_i, x_j) = \max_r | f_r(x_i) - f_r(x_j) |; \quad (5)$$

– метрика расстояния Минковского:

$$d(x_i, x_j) = (\sum_{r=1}^{g} | f_r(x_i) - f_r(x_j) |^h)^{\frac{1}{h}} \ (h \geq 1); \quad (6)$$

– косинусная метрика расстояния:

$$d(x_i, x_j) = \frac{\sum_{r=1}^{g} f_r(x_i) \cdot f_r(x_j)}{\sqrt{(\sum_{r=1}^{g} f_r^2(x_i)) \cdot (\sum_{r=1}^{g} f_r^2(x_j))}}. \quad (7)$$

Итерационный процесс слияния кластеров осуществляется следующим образом: на каждой итерации на основе двух самых близких кластеров U и V формируется новый кластер $W = U \cup V$. При этом расстояние от нового кластера W до любого другого кластера S вычисляется на основе уже известных

расстояний $D(U,V)$, $D(U,S)$ и $D(V,S)$ с помощью универсальной формулы Ланса и Уильямса [3]:

$$D(U \cup V, S) =$$
$$= \alpha_U \cdot D(U,S) + \alpha_V \cdot D(V,S) + \beta \cdot D(U,V) + \gamma \cdot |D(U,S) - D(V,S)|, \quad (8)$$

где α_U, α_V, β, γ – некоторые числовые параметры.

Формула (8) описывает практически все возможные методы вычисления расстояний между кластерами при определённых комбинациях значений параметров α_U, α_V, β, γ (табл. 1, где $|U|$, $|V|$, $|W|$ и $|S|$ – мощности кластеров U, V, W и S соответственно).

Так, например,

– расчет расстояния между кластерами по методу одиночной связи реализуется как:

$$D_{single}(W,S) = \min_{w \in W, s \in S} d(w,s), \quad (9)$$

– расчет расстояния между кластерами по методу полной связи как:

$$D_{complete}(W,S) = \max_{w \in W, s \in S} d(w,s), \quad (10)$$

– расчет по методу средней связи как:

$$D_{average}(W,S) = \frac{1}{|W| \cdot |S|} \cdot \sum_{w \in W} \sum_{s \in S} d(w,s), \quad (11)$$

– расчет расстояния по методу центроида как:

$$D_{centroid}(W,S) = d^2\left(\sum_{w \in W} \frac{w}{|W|}, \sum_{s \in S} \frac{s}{|S|} \right), \quad (12)$$

– расчет расстояния Уорда как:

$$D_{Ward}(W,S) = \frac{|W| \cdot |S|}{|W| + |S|} \cdot d^2\left(\sum_{w \in W} \frac{w}{|W|}, \sum_{s \in S} \frac{s}{|S|} \right). \quad (13)$$

В методе одиночной связи расстояние между двумя кластерами равно минимальному расстоянию между двумя объектами из разных кластеров. В итоге получаемые кластеры имеют предрасположенность к представлению в виде длинными «цепочек».

В методе полной связи расстояние между двумя кластерами равно максимальному расстоянию между двумя объектами. Этот метод хорошо работает, если объекты принадлежат действительно различным хорошо отделимым кластерам. Если же кластеры имеют удлиненную форму (например, если кластеры могут быть представлены в виде длинных «цепочек»), то этот метод является не эффективным.

В методе средней связи расстояние между двумя кластерами равно среднему расстоянию между всеми парами объектов этих кластеров. Этот метод эффекти-

вен, если объекты принадлежат действительно различным хорошо отделимым кластерам. При этом он может давать приемлемые результаты кластеризации и в случае протяженных кластеров «цепочного» типа.

В так называемом методе взвешенной средней связи расстояние $D_{weighted\ average}$ находится как среднее арифметическое расстояний от кластера S до кластеров U и V:

$$D_{weighted\ average}(W,S) = \frac{D(U,S) + D(V,S)}{2}. \qquad (14)$$

Этот метод идентичен методу средней связи, но в нём мощности кластеров используются в качестве весовых коэффициентов для учёта разницы между размерами кластеров. Метод взвешенной средней связи целесообразно использовать, если предполагается, что кластеры будут иметь существенно неравные размеры.

В методе центроида расстояние между кластерами равно расстоянию между их центроидами (центрами тяжести).

Так называемый метод медианы представляет собой взвешенную версию метода центроида, при этом расстояние D_{median} находится как среднее арифметическое для расстояний от центроида кластера S до центроидов кластеров U и V:

$$D_{median}(W,S) = \frac{D(U,S) + D(V,S)}{2} - \frac{D(U,V)}{4}. \qquad (15)$$

В этом методе мощности кластеров используются в качестве весовых коэффициентов для учёта разницы между размерами кластеров и предполагается, что центр нового кластера лежит точно посередине между центрами старых кластеров. Метод медианы целесообразно использовать, если предполагается, что кластеры будут иметь существенно неравные размеры.

В методе Уорда для оценки расстояний между кластерами используются принципы дисперсионного анализа и в качестве расстояния между кластерами полагается прирост суммы квадратов расстояний объектов до центра кластера, получаемого в результате их объединения [3]. На каждом шаге метода Уорда объединяются такие два кластера, группирование которых приводит к минимальному увеличению дисперсии. Этот метод особенно эффективен для решения задач кластеризации с близко расположенными кластерами.

Как и большинство визуальных способов представления зависимостей *дендрограммы теряют наглядность при значительном увеличении числа кластеров*.

Таблица 1. Значения параметров α_U, α_V, β, γ в универсальной формуле (8) для определения метода слияния кластеров

Метод слияния кластеров	α_U	α_U	β	γ
Метод одиночной связи (метод ближайшего соседа, (single-linkage clustering method)	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$
Метод полной связи (метод дальнего соседа, (complete-linkage clustering method)	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
Метод средней связи (метод группового среднего расстояния, average-linkage clustering method)	$\frac{\lvert U\rvert}{\lvert W\rvert}$	$\frac{\lvert V\rvert}{\lvert W\rvert}$	0	0
Метод взвешенной средней связи (метод взвешенного группового среднего расстояния, weighted average-linkage clustering method)	$\frac{1}{2}$	$\frac{1}{2}$	0	0
Метод центроида (centroid-linkage clustering method)	$\frac{\lvert U\rvert}{\lvert W\rvert}$	$\frac{\lvert V\rvert}{\lvert W\rvert}$	$-\alpha_U \cdot \alpha_V$	0
Метод медианы (median-linkage clustering method)	$\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{4}$	0
Метод расчета расстояния Уорда (Ward-linkage clustering method)	$\frac{\lvert U\rvert+\lvert S\rvert}{\lvert W\rvert+\lvert S\rvert}$	$\frac{\lvert V\rvert+\lvert S\rvert}{\lvert W\rvert+\lvert S\rvert}$	$\frac{-\lvert S\rvert}{\lvert W\rvert+\lvert S\rvert}$	0

Алгоритм иерархической кластеризации может быть описан следующей последовательностью шагов.

Шаг 1. Принять номер шага слияния l, равным 1. Выполнить инициализацию множества кластеров одноэлементными кластерами, состоящими из объектов кластеризации:

$C^l = \{\{x_1\},\{x_2\},...,\{x_n\}\}$.

Шаг 2. Выполнить расчет матрицы расстояний между одноэлементными кластерами на основе формулы (1).

Шаг 3. Увеличить номер шага слияния l на 1. Найти в множестве C^{l-1} два ближайших кластера U и V, а затем объединить их в один кластер: $W = U \cup V$. Удалить из множества кластеров C^{l-1} кластеры U и V. Добавить в множество кластеров C^{l-1} новый кластер W: $C^l = (C^{l-1} \setminus \{U,V\}) \cup \{W\}$.

Шаг 4. Для всех кластеров $S \in C^l$ вычислить расстояние $D(W,S)$ по формуле Ланса-Уильямса (8).

Шаг 5. Если число кластеров в множестве C^l больше 1, перейти к шагу 3, в противном случае завершить работу алгоритма.

Существенный интерес представляет анализ свойств используемых функций расстояний $D(W,S)$.

Пусть D^l – расстояние между ближайшими кластерами, выбранными на l-м шаге для слияния.

Функция расстояния $D = D(W,S)$ монотонна, если при каждом слиянии расстояние между объединяемыми кластерами только увеличивается:

$$D^2 \le D^3 \le ... \le D^{k-1} \le D^k, \qquad (16)$$

где k – число шагов слияния (в предположении, что при $l = 1$ выполняется расчет расстояний между объектами кластеризации, а при l, удовлетворяющем условию $1 < l \le k$, – слияние кластеров).

Свойство монотонности позволяет представить процесс кластеризации в виде дендрограммы, при построении которой объекты могут откладываться, например, по горизонтальной оси, а расстояния D^l ($l = \overline{1,k}$) – по вертикальной.

Дендрограмма позволяет представить кластерную структуру графически в двумерном пространстве независимо от того, чему равна размерность исходного пространства.

Доказано, что если процесс кластеризации идёт монотонно, то есть функция расстояния обладает свойством монотонности, то дендрограмму можно построить таким образом, что она не будет иметь самопересечения. В этом случае любой кластер из множества C^l кластеров l-го шага может быть представлен сплошной последовательностью точек на горизонтальной оси. Если процесс кластеризации идёт не монотонно, то в качестве дендрограммы будет получен набор линий с самопересечениями, сложный для интерпретации.

Как показывает анализ, не все комбинации значений параметров в формуле (8) Ланса-Вильямса приводят к монотонности процесса кластеризации. В 1979 году Г. Миллиган (Glenn W. Milligan) доказал теорему, определяющую условия монотонности процесса кластеризации [3]:

если выполняются следующие три условия:

– $\alpha_U \ge 0,\ \alpha_V \ge 0;$

– $\alpha_U + \alpha_V + \beta \ge 1;$

– $\min(\alpha_U, \alpha_V) + \gamma \ge 0,$

то процесс кластеризации является монотонным.

Из рассмотренных выше расстояний только расстояния $D_{centroid}$ (12) и D_{median} (15) не являются монотонными. Расстояние D_{ward} (13), похожее на рас-

стояние $D_{centroid}$ (12), отличается от него мультипликативной поправкой, которая делает его монотонным.

При выборе метода слияния кластеров необходимо учитывать наличие или отсутствие у него свойств растяжения и сжатия.

Если расстояние D обладает свойством растяжения, то это означает, что по мере роста кластера расстояния от него до других кластеров увеличиваются, то есть проявляется эффект растяжения пространства вокруг кластера. Желательно, чтобы расстояние D обладало свойством растяжения, так как это способствует более чёткому отделению кластеров друг от друга, однако сильная выраженность свойства растяжения может привести выявлению в реальности несуществующих кластеров.

Так, свойство растяжения присуще расстояниям $D_{complete}$ и D_{Ward}.

Если расстояние D обладает свойством сжатия, то это означает, что по мере роста кластера расстояния от него до других кластеров уменьшается, то есть проявляется эффект сжатия пространства вокруг кластера, при этом естественность кластеризации может быть нарушена. Так, свойство сжатия сильно выражено у расстояния D_{single}.

Степень выраженности свойств растяжения и сжатия можно оценить через отношение $\frac{D(U,V)}{d(m_U, m_V)}$, где $D(U,V)$ – расстояние между ближайшими кластерами U и V, объединяемыми на текущем шаге; m_U и m_V – центры кластеров U и V соответственно. Если это отношение на каждом шаге больше единицы, то расстояние D является растягивающим; если оно всегда меньше единицы, то сжимающим. Если расстояние D не является ни сжимающим, ни растягивающим, то считается, что оно обладает свойством сохранения метрики пространства. Там свойством сохранения метрики пространства обладают, например, расстояния $D_{average}$ (11), $D_{centroid}$ (12), $D_{weighted\ average}$ (14) и D_{median} (15) [3].

Зачастую на практике используют так называемое гибкое расстояние, которое реализует компромисс между методами одиночной, полной и средней связей. Для гибкого расстояния значения параметров в формуле (8) задаются как:

$$\alpha_U = \alpha_V = \frac{1-\beta}{2},\ \gamma = 0,\ \beta < 1.$$

Гибкое расстояние является сжимающим при $0 < \beta < 1$ и растягивающим при $\beta < 0$. Обычно β выбирают равным числу «-0.25».

Следует отметить, что при выборе метода слияния кластеров необходимо учитывать его совместимость с метрикой для вычисления расстояния между

объектами. Так, метод центроида, метод медианы и метод Уорда предполагают использование метрики евклидова расстояния.

В результате применения того или иного метода слияния кластеров формируется матрица сходства (или различия) кластеров, которая определяет уровень сходства (различия) между парами кластеров и используется для анализа и визуализации результатов кластеризации посредством построения дендрограммы.

2.2. Программная реализация алгоритмов иерархической кластеризации на языке Python

Для демонстрации принципов работы библиотек, реализующих алгоритмы кластерного анализа, будем использовать Jupyter Notebook.

Для работы с алгоритмами иерархической агломеративной кластеризации следует воспользоваться программной библиотекой, предложенной по ссылке:

https://pypi.org/project/scipy/.

Для установки программной библиотеки необходимо выполнить команду:

```
pip install scipy
```

SciPy (https://scipy.org/) – программное обеспечение с открытым исходным кодом для решения различных задач в сферах математики, науки и техники [4]. SciPy предоставляет программные реализации алгоритмов для задач оптимизации, интегрирования, интерполяции, статистики, поиска решений алгебраических и дифференциальных уравнений и многих других классов задач.

Подробная документация по всем функциям, используемым в программной реализации алгоритма иерархической агломеративной кластеризации, приведена по ссылке:

https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html

Подробная документация по программной реализации функции `linkage`, используемой для расчета матрицы связей в алгоритме иерархической агломеративной кластеризации, приведена по ссылке:

https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html#scipy.cluster.hierarchy.linkage

Ниже представлена информация по функции `linkage`, реализующей расчет матрицы связей.

Синтаксис:

```
scipy.cluster.hierarchy.linkage(y, method='single', metric='euclidean', optimal_ordering=False)
```

Входные параметры:

y – сжатая матрица расстояний, в виде плоского (одномерного) массива, содержащего верхний треугольник матрицы расстояний между объектами исходного набора данных, или двумерный массив, соответствующий исходному набору данных;

method – метод вычисления расстояния между кластерами (`'single'`, `'complete'`, `'average'`, `'weighted'`, `'centroid'`, `'median'`, `'ward'`);

metric – метрика вычисления расстояний между объектами (`'braycurtis'`, `'canberra'`, `'chebyshev'`, `'cityblock'`, `'correlation'`, `'cosine'`, `'euclidean'`, `'jensenshannon'`, `'mahalanobis'`, `'minkowski'`, `'seuclidean'`, `'sqeuclidean'`, `'wminkowski'`);

optimal_ordering – параметр типа `boolean`, определяющий, нужно ли минимизировать расстояния между последовательными листьями дерева дендрограммы (если установлено значение `True`, то матрица связей будет переупорядочена таким образом, чтобы расстояние между последовательными листьями было минимальным, что приводит к более интуитивно понятной древовидной структуре при визуализации данных; по умолчанию установлено значение `False`, так как применение алгоритма упорядочения замедляет работу, особенно на больших наборах данных).

Сжатая матрица расстояний может быть вычислена с применением функции *pdist*, реализующей расчет попарных расстояний между объектами. Подробная документация по программной реализации функции *pdist* представлена по ссылке:

https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html#scipy.spatial.distance.pdist

Для вычисления расстояний между объектами, значения признаков которых вещественные, используются указанные выше метрики.

В случае работы с объектами, значения признаков которых – булевы векторы, можно использовать другие метрики, например, `'dice'`, `'hamming'`, `'jaccard'`, `'kulsinski'`, `'rogerstanimoto'`, `'russellrao'`, `'sokalmichener'`, `'sokalsneath'`, `'yule'`.

Значения входных параметров функции `linkage`, используемые по умолчанию, указаны при описании ее синтаксиса.

Выходной параметр:

Z – матрица связей размером $(n-1)\times 4$ (n – число объектов в наборе данных), содержащая результаты иерархической кластеризации.

Матрица связей содержит $(n-1)$ строку и 4 столбца.

Первый и второй столбцы содержат номера связываемых кластеров на текущей итерации. Третий столбец содержит расстояние между двумя кластерами, указанными в двух первых столбцах, а четвертый столбец содержит число объектов, содержащихся во вновь созданном кластере.

На i-й итерации кластеры с номерами `Z[i,0]` и `Z[i,1]` объединяются для формирования кластера $(n+i)$. Кластер с номером, меньшим, чем n, соответствует одному из n исходных объектов. Расстояние между кластерами с номерами `Z[i,0]` и `Z[i,1]` находится в `Z[i,2]`. Значение `Z[i,3]` содержит число исходных объектов, попавших в новый кластер на i-й итерации.

Подробная документация по программной реализации функции `dendrogram`, реализующей построение дендрограммы, приведена по ссылке:

https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.dendrogram.html#scipy.cluster.hierarchy.dendrogram

Дендрограмма показывает, как сформирован каждый кластер, изображая U-образную связь между неодноэлементным кластером и его дочерними элементами.

Верх U-образной связи указывает на слияние кластеров. Две ветви U-связи указывают, какие кластеры были объединены. Длина двух ветвей U-связи представляет собой расстояние между дочерними кластерами. Это также кофенетическое расстояние, то есть расстояние между исходными объектами в двух дочерних кластерах на дендрограмме.

Ниже представлена информация по функции `dendrogram`, реализующей построение дендрограммы для результатов иерархической агломеративной кластеризации.

Синтаксис:

```
scipy.cluster.hierarchy.dendrogram(Z, p=30, truncate_mode=None, color_threshold=None, get_leaves=True, orientation='top', labels=None, count_sort=False, distance_sort=False, show_leaf_counts=True, no_plot=False, no_labels=False, leaf_font_size=None, leaf_rotation=None, leaf_label_func=None, show_contracted=False, link_color_func=None, ax=None, above_threshold_color='C0')
```

Входные параметры:

Z – матрица связей размером $(n-1)\times 4$ (n – число объектов в наборе данных), содержащая результаты иерархической кластеризации;

p – параметр для `truncate_mode`;

truncate_mode – параметр, отвечающий за усечение с целью сжатия дендрограммы, которая может оказаться трудна для восприятия, если исходный набор данных очень большой (`None` (усечение не выполняется), `'lastp'` (последние `p` неодноэлементных кластеров, образованных в матрице связей, являются единственными нелистовыми узлами в матрице связи, при этом они соответствуют строкам `z[n-p-2:end]` в `z`; все остальные неодноэлементные кластеры стягиваются в конечные узлы), `'level'` (отображается не более `p` уровней дерева дендрограммы, при этом уровень включает в себя все узлы с `p` слияниями из итогового слияния));

color_threshold – параметр, отвечающий за окраску ветвей дендрограммы (все ссылки-потомки ниже узла кластера *k* окрашиваются в один и тот же цвет, если *k* – первый узел ниже порога отсечения *color_threshold*; все ссылки, соединяющие узлы с расстояниями, большими или равными порогу, окрашиваются по умолчанию цветом matplotlib `'C0'`; если *color_threshold* меньше или равно нулю, все узлы окрашиваются в цвет `'C0'`; если для *color_threshold* установлено значение `None` или `'default'`, порог устанавливается равным `0,7*max(z[:, 2])`);

get_leaves – параметр, отвечающий за включение списка листьев в итоговый словарь;

orientation – параметр, определяющий направление построения дендрограммы, то есть фиксирующий расположение корня дерева (`'top'`, `'bottom'`, `'left'`, `'right'`);

labels – массив, отвечающий за метки, приписываемые листовым узлам дерева (по умолчанию *labels* содержит номера исходных объектов, используемые для маркировки листовых узлов, в противном случае – это массив размера `n=z.shape[0]+1`, при этом значение `labels[i]` – текст, который помещается под *i*-м листовым узлом, только если он соответствует исходному объекту);

count_sort – параметр, отвечающий за порядок отображения узлов в дереве при просмотре его слева направо с учетом номеров объектов в исходном наборе данных (`False`, `'ascending'`, `'descending'`);

distance_sort – параметр, отвечающий за порядок отображения узлов в дереве при просмотре его слева направо: дочерний элемент отображается с

учетом упорядочения расстояний между его прямыми потомками (False, 'ascending','descending');

show_leaf_counts – параметр, отвечающий за отображение информации у листовых узлов (если задано значение True, листовые узлы помечаются числом содержащихся в них объектов в скобках);

no_plot – параметр, отвечающий за отрисовку дендрограммы (если задано значение True, финальная отрисовка не выполняется: это полезно, если нужны только структуры данных, вычисленные для отрисовки, или если matplotlib недоступен);

no_labels – параметр, отвечающий за вывод меток листовых узлов на дендрограмме (если задано значение True, метки рядом с листовыми узлами не отображаются);

leaf_font_size – параметр, определяющий размер шрифта (в пунктах) для меток листовых узлов (если значение не указано, размер зависит от числа узлов в дендрограмме);

leaf_rotation – параметр, задающий угол (в градусах) для поворота меток листьев (если значение не указано, вращение основано на числе узлов в дендрограмме (по умолчанию 0));

leaf_label_func – функция для описания листовых узлов;

show_contracted – параметр, отвечающий за отображение высоты неодноэлементных узлов (если задано значение True, высоты неодноэлементных узлов, объединенных в листовой узел, отображаются в виде крестов вдоль связи, соединяющей с этим листовым узлом (это полезно при использовании сечения (с применением truncate_mode)));

link_color_func – функция для определения цвета связи;

ax – параметр, определяющий способ отображения дендрограммы (если значение параметра None и no_plot не равно True, дендрограмма будет построена на текущих осях, в противном случае, если no_plot не равно True, дендрограмма будет построена на заданном экземпляре Axes (это может быть полезно, если дендрограмма является частью более сложной фигуры));

above_threshold_color – параметр, устанавливающий цвет связей выше color_threshold в matplotlib (по умолчанию 'C0').

Значения входных параметров функции dendrogram, используемые по умолчанию, указаны при описании ее синтаксиса.

Выходной параметр:

R – словарь структур данных, используемый для отображения дендрограммы.

Подробная документация по программной реализации функции `fcluster`, реализующей алгоритм формирования плоских кластеров, приведена по ссылке:

https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.fcluster.html#scipy.cluster.hierarchy.fcluster

Формирование плоских кластеров и назначение объектам набора данных соответствующих меток кластеров осуществляется на основе матрицы связей, вычисленной с применением функции `linkage`.

Ниже представлена информация по функции `fcluster`, реализующей алгоритм формирования плоских кластеров.

Синтаксис:

`scipy.cluster.hierarchy.fcluster(Z, t, criterion='inconsistent', depth=2, R=None, monocrit=None)`

Входные параметры:

Z – матрица связей размером $(n-1)\times 4$ (n – число объектов в наборе данных), содержащая результаты иерархической кластеризации;

t – параметр, учитываемый при формировании плоских кластеров: для критериев *'inconsistent'*, *'distance'* и *'monocrit'* – это порог, используемый при формировании кластеров; для критериев *'maxclust'* и *'maxclust_monocrit'* – это максимальное число кластеров;

criterion – критерий, используемый при формировании кластеров:

'inconsistent' (плоские кластеры формируются так, что: если узел кластера и все его потомки имеют значение несогласованности, меньшее или равное *t*, то все его конечные потомки принадлежат одному и тому же плоскому кластеру; если этому критерию не соответствует ни один неодноэлементный кластер, каждый узел назначается своему собственному кластеру);

'distance' (плоские кластеры формируются так, чтобы объекты в каждом кластере имели кофенетическое расстояние не больше, чем *t*);

'maxclust' (плоские кластеры формируются так, что учитывается найденное минимальное пороговое значение *r*, при котором кофенетическое расстояние между любыми двумя объектами в одном и том же плоском кластере не превышает *r*, при этом формируется не более *t* плоских кластеров);

'monocrit'; *'maxclust_monocrit'* (плоские кластеры формируются с учетом информации, хранимой в векторе *monocrit*, который должен быть монотонным);

depth – параметр, определяющий максимальную глубину для вычисления значения несогласованности по критерию *'inconsistent'* (для других критериев этот параметр не учитывается);

R – матрица несогласованности, используемая для критерия *'inconsistent'*, она вычисляется, если не задана в списке параметров;

monocrit – вектор длиной (*n* – 1), при этом вектор *monocrit* должен быть монотонным, т. е. для узла *c* с индексом *i* для всех узлов индексов *j*, соответствующих узлам ниже *c*, *monocrit[i] >= monocrit[j]*, где *monocrit[q]* – статистика, по которой устанавливается пороговое значение для несинглтона *q (q=1, 2)*;

Следует привести некоторые пояснения с примерами работы функции `fcluster` с критериями *'monocrit'* и *'maxclust_monocrit'*.

Работа с критерием *'monocrit'*, например, при пороге `t=0.8` может быть представлена следующими строками программного кода, при условии, что используется функция *maxRstat* (https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.cluster.hierarchy.maxRstat.html), возвращающая максимальную статистику для каждого неодноэлементного кластера и его потомков:

```
MR = maxRstat(Z, R, 3)
fcluster(Z, t=0.8, criterion='monocrit', monocrit=MR)
```

Работа с критерием *'maxclust_monocrit'*, например, при числе кластеров `t=3` может быть представлена следующими строками программного кода, при условии, что используется функция *maxinconsts* (https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.maxinconsts.html), возвращающая максимальный коэффициент несогласованности для каждого неодноэлементного кластера и его дочерних элементов:

```
MI = maxinconsts(Z, R)
fcluster(Z, t=3, criterion='maxclust_monocrit',
monocrit=MI)
```

Значения входных параметров функции `fcluster`, используемые по умолчанию, указаны при описании ее синтаксиса.

Выходной параметр:

fcluster – массив длиной *n*, *i*-й элемент которого содержит номер плоского кластера, к которому отнесен *i*-й объект набора данных.

2.3. Пример иерархического кластерного анализа

Рассмотрим возможности иерархического кластерного анализа на примере набора данных «Ирисы Фишера» (The Iris Dataset), который представлен по ссылкам:

http://archive.ics.uci.edu/ml/datasets/Iris;
https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html.

«Ирисы Фишера» – набор данных для задачи многоклассовой классификации, на примере которого Рональд Фишер в 1936 году продемонстрировал работу предложенного им метода дискриминантного анализа.

Набор данных «Ирисы Фишера» содержит данные о 150 экземплярах ириса таких трёх видов, как (рис. 5):

– ирис щетинистый (*Iris setosa*);
– ирис виргинский (*Iris virginica*);
– ирис разноцветный (*Iris versicolor*).

При этом для каждого вида ириса имеются данные о 50 экземплярах.

Для каждого экземпляра ириса в наборе данных содержится информация по таким четырем характеристикам, как:

– длина наружной доли околоцветника (*sepal length*);
– ширина наружной доли околоцветника (*sepal width*);
– длина внутренней доли околоцветника (*petal length*);
– ширина внутренней доли околоцветника (*petal width*).

Значения измерений по всем характеристикам представлены в сантиметрах (рис. 6).

На основе этого набора данных требуется построить правило классификации, определяющее вид ириса по результатам измерений.

Iris setosa *Iris virginica* *Iris versicolor*

Рисунок 5. Ирисы

Это задача многоклассовой классификации, так как есть три класса, соответствующие трём видам ириса. При этом один из классов, соответствующий виду *Iris setosa*, линейно отделим от двух остальных.

Как видно из рис. 6, значения характеристик лежат в существенно разных диапазонах (как по значениям характеристик, так по длине самих диапазонов). Этот факт может отрицательным образом отразиться на результатах решения

задач классификации и кластеризации. Так, при разбиении объектов некоторого набора данных на кластеры для вычисления расстояний между объектами используются те или иные метрики, например, метрика евклидова расстояния. Метрические классификаторы (например, *k*NN-классификатор (*k*-nearest neighbors, классификатор на основе алгоритма *k*-ближайших соседей)) также используют метрики расстояний при решении задачи классификации объектов. В связи с этим целесообразно выполнение предварительной обработки набора данных посредством его масштабирования (scaling). При масштабировании осуществляется преобразование значений каждого признака объектов набора данных по определенному правилу.

Ирисы Фишера

Длина чашелистика	Ширина чашелистика	Длина лепестка	Ширина лепестка	Вид ириса
5.1	3.5	1.4	0.2	*setosa*
4.9	3.0	1.4	0.2	*setosa*
4.7	3.2	1.3	0.2	*setosa*
4.6	3.1	1.5	0.2	*setosa*
5.0	3.6	1.4	0.2	*setosa*
5.4	3.9	1.7	0.4	*setosa*
4.6	3.4	1.4	0.3	*setosa*
5.0	3.4	1.5	0.2	*setosa*
4.4	2.9	1.4	0.2	*setosa*
4.9	3.1	1.5	0.1	*setosa*
5.4	3.7	1.5	0.2	*setosa*
4.8	3.4	1.6	0.2	*setosa*
4.8	3.0	1.4	0.1	*setosa*

Рисунок 6. Фрагмент набора данных

Это может быть масштабирование в некоторый диапазон по каждому признаку (например, в диапазон [0, 1] по каждому признаку таким образом, что все объекты после масштабирования находятся в единичном гиперкубе), либо масштабирование с использованием стандартизации (например, так, что по каждому признаку математическое ожидание равно 0, а дисперсия равна 1). Могут

использоваться и другие подходы к масштабированию (в зависимости от контекста решаемой задачи).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn import datasets
iris = datasets.load_iris()

df=pd.DataFrame(iris['data'])
print(df.head())
```

```
     0    1    2    3
0  5.1  3.5  1.4  0.2
1  4.9  3.0  1.4  0.2
2  4.7  3.2  1.3  0.2
3  4.6  3.1  1.5  0.2
4  5.0  3.6  1.4  0.2
```

```
np.unique(iris.target,return_counts=True)
```

```
(array([0, 1, 2]), array([50, 50, 50], dtype=int64))
```

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

Рисунок 7. Фрагмент программного кода (начало)

При написании программ на языке Python можно использовать уже имеющуюся библиотеку `scikit-learn`, предоставляющую различные инструменты для прогнозного анализа данных, в том числе, инструменты машинного обучения.

Эта библиотека предоставляет и различные средства масштабирования данных [5], в частности:

– `sklearn.preprocessing.MinMaxScaler` (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html), предназначенный для масштабирования каждого признака объекта по отдельности в заданный диапазон (например, в диапазон [0, 1] по умолчанию);

– `sklearn.preprocessing.StandardScaler` (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler), предназначенный для масштабирования каждого признака объекта по отдельности так, чтобы математическое ожидание для каждого признака было равно 0, а дисперсия – 1).

Для работы со средствами масштабирования данных следует воспользоваться программной библиотекой, предложенной по ссылке:

https://scikit-learn.org/stable/.

```
# Импортирование пакета предварительной обработки данных
from sklearn import preprocessing
```

```
scaler = preprocessing.MinMaxScaler().fit(df.to_numpy())
scaled_data=scaler.transform(df.to_numpy())
pd.DataFrame(scaled_data).describe()
```

	0	1	2	3	4	5
count	150.000000	150.000000	150.000000	150.000000	150.000000	150.000000
mean	0.428704	0.440556	0.467458	0.458056	0.573333	0.500000
std	0.230018	0.181611	0.299203	0.317599	0.442638	0.409616
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.222222	0.333333	0.101695	0.083333	0.000000	0.000000
50%	0.416667	0.416667	0.567797	0.500000	0.500000	0.500000
75%	0.583333	0.541667	0.694915	0.708333	1.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

```
# Импортирование функции linkage
from scipy.cluster.hierarchy import linkage
```

```
# Расчет матрицы связей
distance_matrix = linkage(scaled_data, method = 'ward', metric = 'euclidean')
distance_matrix
```

```
array([[1.01000000e+02, 1.42000000e+02, 0.00000000e+00, 2.00000000e+00],
       [7.00000000e+00, 3.90000000e+01, 2.77777778e-02, 2.00000000e+00],
       [1.00000000e+01, 4.80000000e+01, 2.77777778e-02, 2.00000000e+00],
```

Рисунок 8. Фрагмент программного кода (продолжение)

Для установки программной библиотеки необходимо выполнить команду:

```
pip install -U scikit-learn
```

Следует отметить, что не существует универсального способа масштабирования данных: целесообразно применять разные варианты масштабирования с целью выбора лучшего для конкретной прикладной задачи.

На рис. 7 приведен фрагмент программного кода, в котором отражены:

– импортирование библиотек;

– загрузка набора данных;

– вывод первых пяти строк этого набора данных;

– вывод информации по меткам классов и числу экземпляров объектов в каждом классе;

– вывод названий классов.

На рис. 8 приведен фрагмент программного кода, в котором отражены:

– импортирование пакета предварительной обработки данных;

– масштабирование набора данных с применением `MinMaxScaler`;

– вывод информации о результатах масштабирования (в частности, видно, что по каждому признаку минимальное и максимальное значения равны соответственно 0 и 1);

– импортирование функции `linkage`;

– расчет матрицы связей с использованием функции `linkage`;

– вывод фрагмента матрицы связей.

При расчете матрицы связей (рис. 8) используется метрика расстояния `'euclidean'` и метод вычисления расстояния между кластерами `'ward'`.

На рис. 9 приведен фрагмент программного кода, в котором отражены:

– импортирование функции для построения дендрограммы;

– создание полотна для рисования;

– построение и вывод дендрограммы.

На рис. 10 – 13 представлены результаты визуализации дендрограмм для методов вычисления расстояний между кластерами `'ward'`, `'single'`, `'complete'`, `'average'` соответственно.

В этой задаче известны метки классов объектов и можно их использовать, чтобы оценить, в том числе, визуально, работу алгоритма иерархической кластеризации.

Кроме того, можно выполнить кластеризацию без масштабирования и сравнить полученные результаты результатами кластеризации, полученными при масштабировании, обратив внимание на метки кластеров (в том числе – на их расположение на дендрограмме).

Это позволит убедиться в необходимости предварительного масштабирования набора данных.

```
# импортирование функции для построения дендрограммы
from scipy.cluster.hierarchy import dendrogram
```

```
# Создание полотна для рисования
fig = plt.figure(figsize=(15, 30))
fig.patch.set_facecolor('white')

# Построение дендрограммы
# Разные цвета - разные автоматически определенные кластеры
R = dendrogram(distance_matrix,
               labels=[iris.target_names[i] for i in iris.target],
               orientation='left',
               leaf_font_size = 12)

# Вывод дендрограммы
plt.show()
```

Рисунок 9. Фрагмент программного кода (окончание)

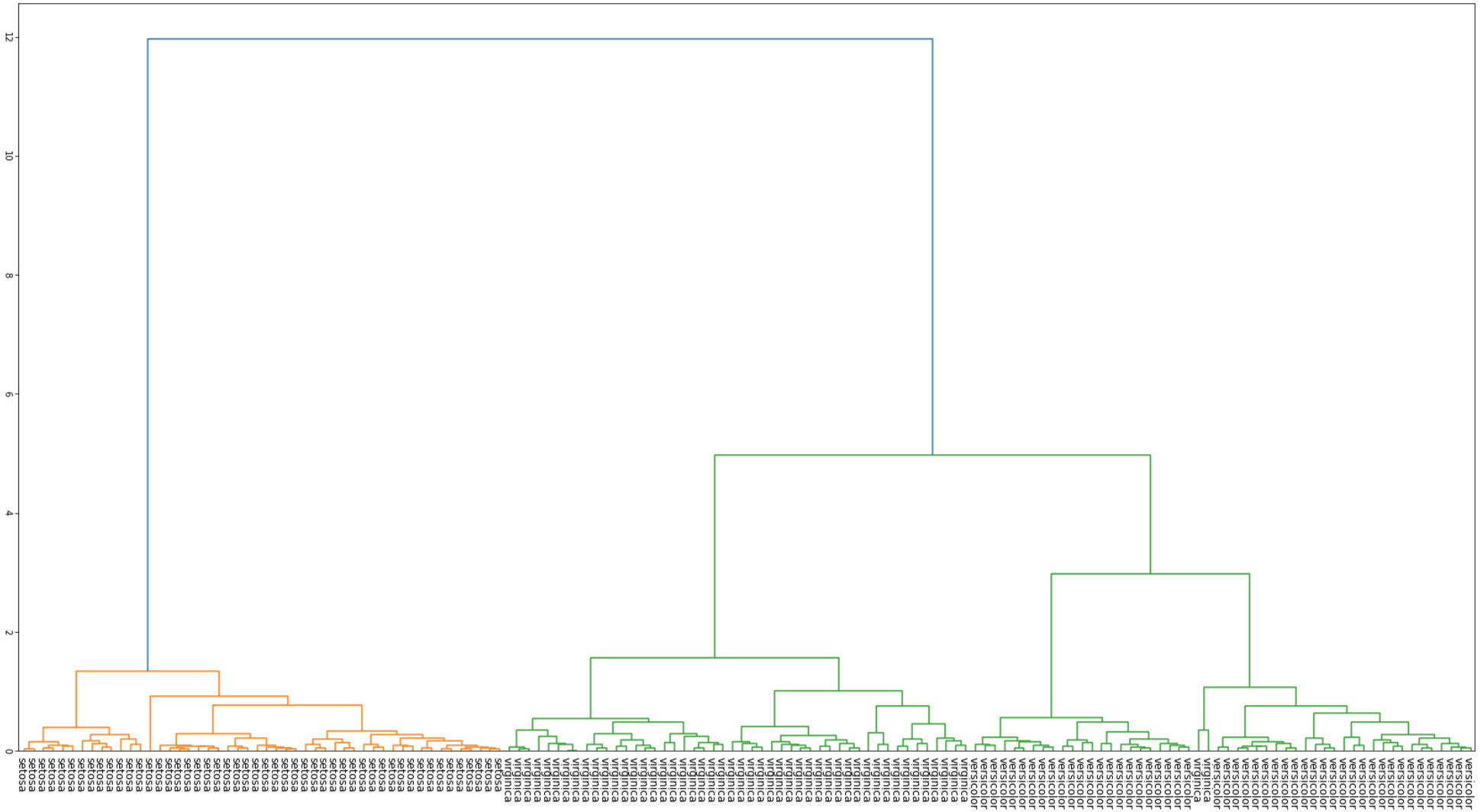


Рисунок 10. Иерархическая кластеризация с применением метода Уорда (`'ward'`)

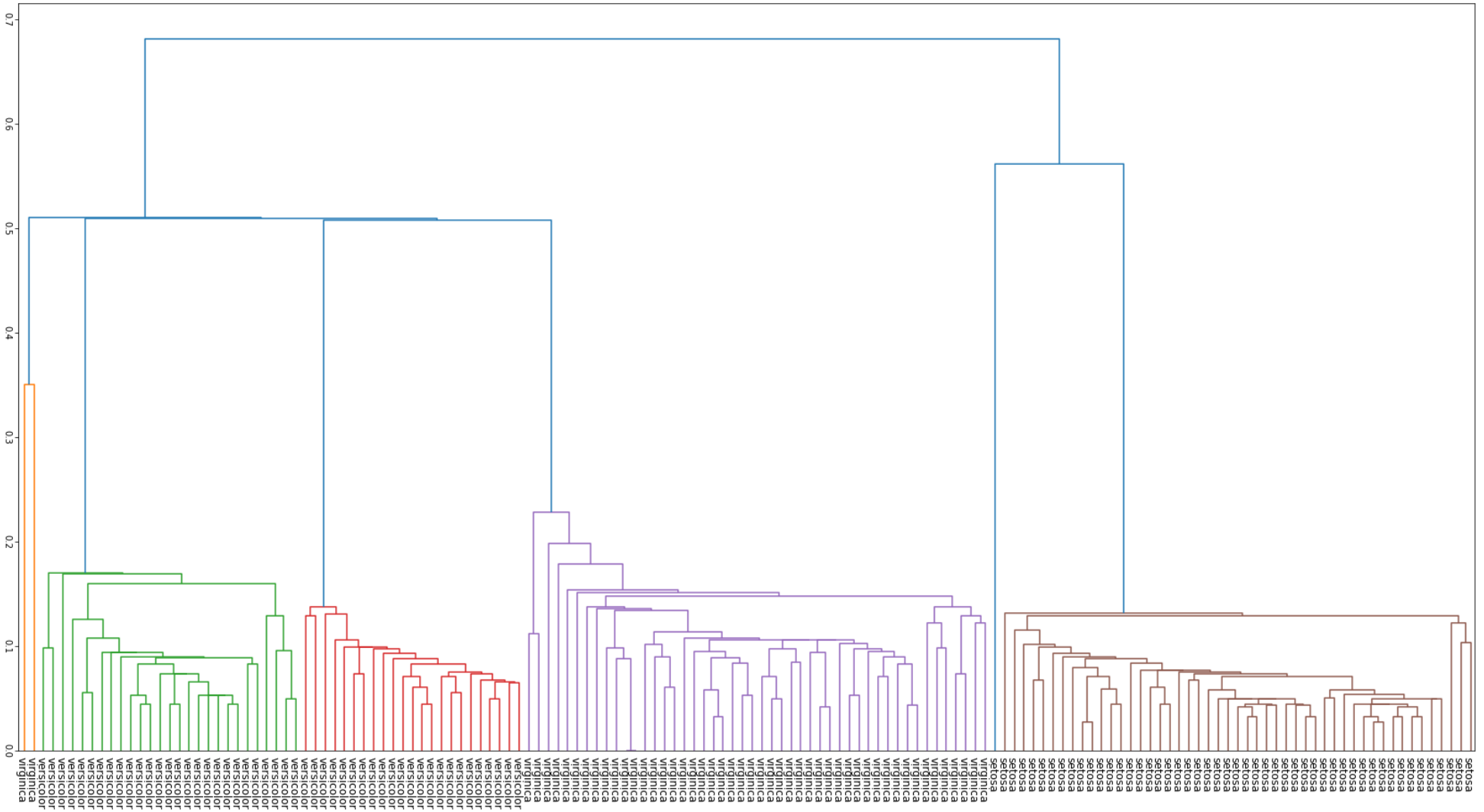


Рисунок 11. Иерархическая кластеризация с применением метода одиночной связи (`'single'`)

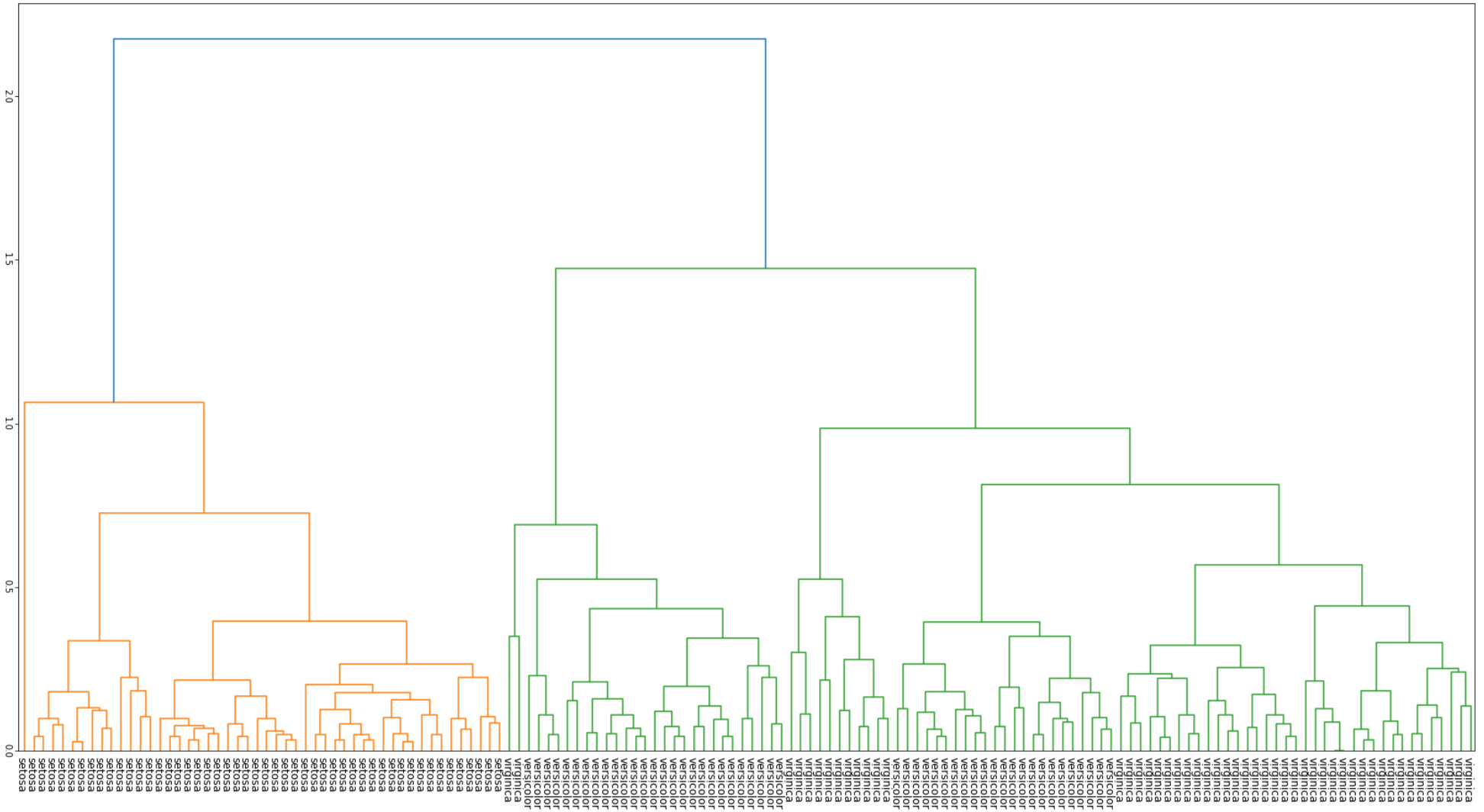


Рисунок 12. Иерархическая кластеризация с применением метода полной связи (`'complete'`)

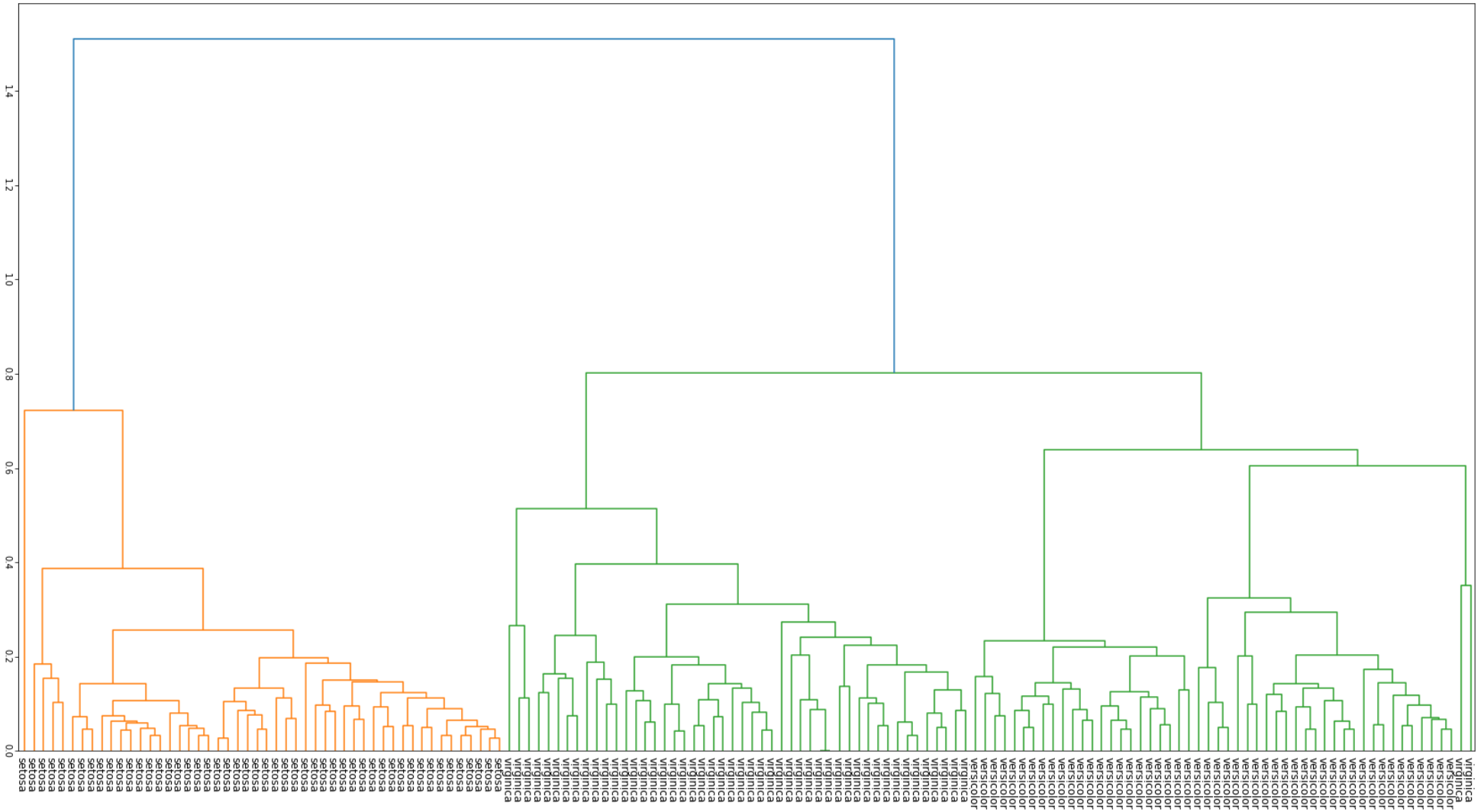


Рисунок 13. Иерархическая кластеризация с применением метода средней связи (`'average'`)

```
# Импортирование функции fcluster
from scipy.cluster.hierarchy import fcluster
```

```
# Назначение меток кластеров
df['cluster_labels'] = fcluster(distance_matrix, 3, criterion='maxclust')
```

```
df['target'] = iris.target

fig, axes = plt.subplots(1, 2, figsize=(16,8))
axes[0].scatter(scaled_data[:,0], scaled_data[:,1],
                c=df['target'])
axes[1].scatter(scaled_data[:,0], scaled_data[:,1],
                c=df['cluster_labels'], cmap=plt.cm.Set1)
axes[0].set_title('Реальные метки кластеров', fontsize=16)
axes[1].set_title('Метки кластеров, найденные алгоритмом', fontsize=16)
```

Рисунок 14. Фрагмент программного кода, позволяющий назначить метки кластеров объектам на основе функции fcluster, и выполнить визуализацию

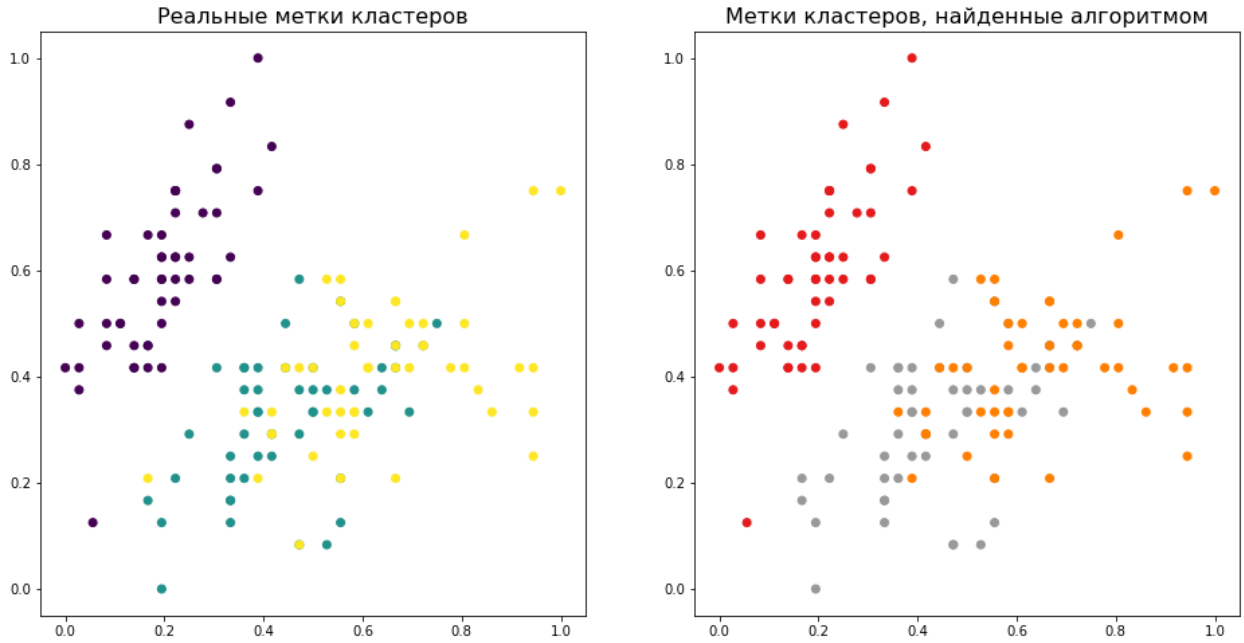


Рисунок 15. Результаты визуализации набора данных по первому и второму признаку с применением метода Уорда (`'ward'`)

Анализ рис. 10 – 13 позволяет сделать следующие выводы: лучше всего с задачей кластеризации справились алгоритмы с использованием методов `'ward'` и `'average'`. Они точно выделили класс *Iris setosa* и заметно отделили класс *Iris virginica* от *Iris versicolor*.

На рис. 14 приведен фрагмент программного кода, который может быть использован для назначения меток кластеров объектам на основе функции `fcluster`. Здесь же реализована возможность визуализации набора данных по первому и второму признаку (в двумерном пространстве):

– с учетом реальных меток кластеров;

– с учетом меток кластеров, найденных алгоритмом.

На рис. 15 – 18 представлены результаты визуализации набора данных по первому и второму признаку (в двумерном пространстве) для методов вычисления расстояний между кластерами `'ward'`, `'single'`, `'complete'`, `'average'` соответственно. При этом визуализация выполнена для набора данных, к которому применено масштабирование.

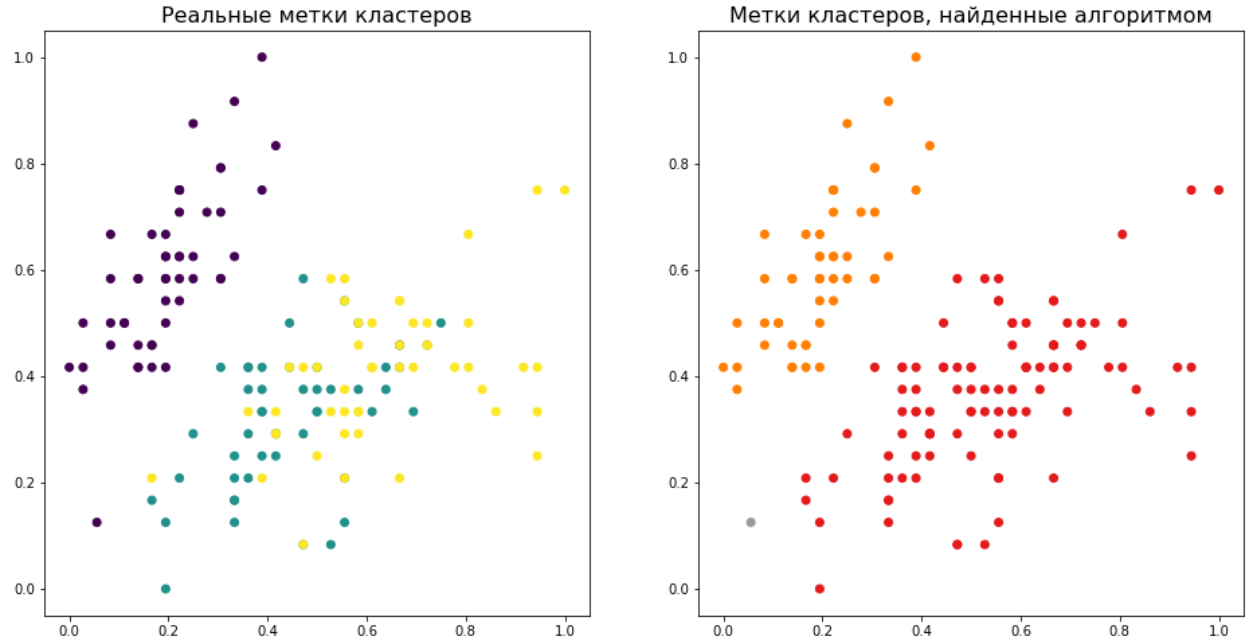


Рисунок 16. Результаты визуализации набора данных по первому и второму признаку с применением метода одиночной связи (`'single'`)

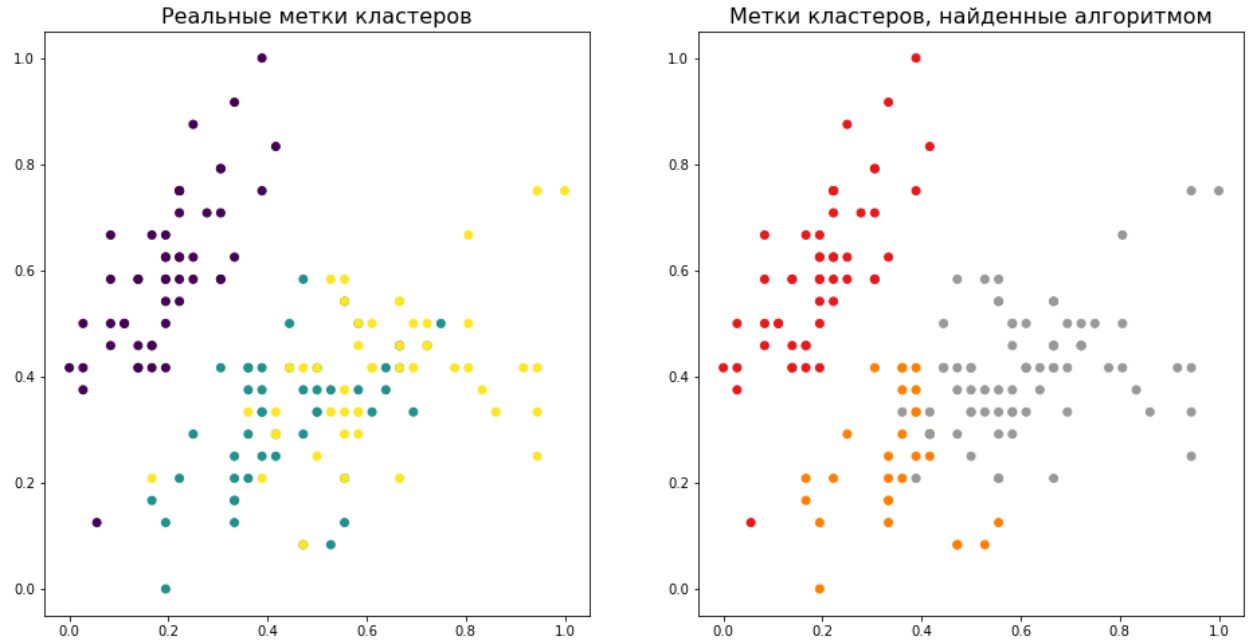


Рисунок 17. Результаты визуализации набора данных по первому и второму признаку с применением метода полной связи (`'complete'`)

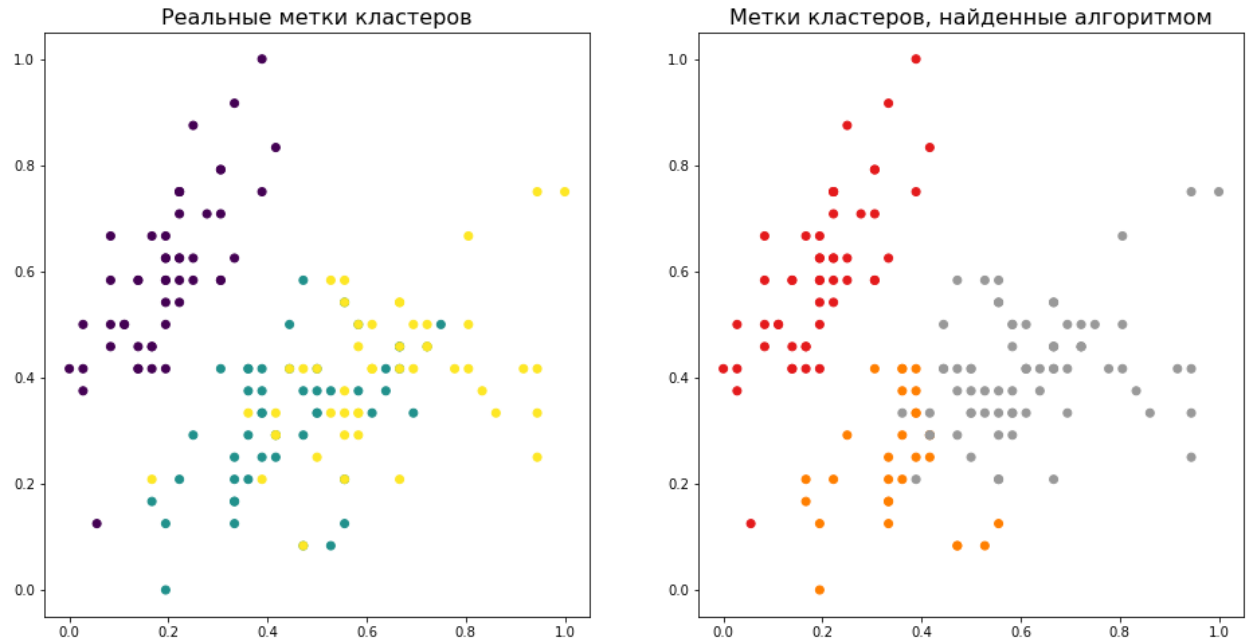


Рисунок 18. Результаты визуализации набора данных по первому и второму признаку с применением метода средней связи (`'average'`)

Следует отметить, что, поскольку в рассматриваемой задаче масштабирование выполнялось в [0, 1] по каждому признаку, результаты визуализации для исходного набора данных будут идентичными.

Использование функции `fcluster` с применением критерия `'maxclust'` для выделения кластеров позволило разбить набор данных на основе матрицы связей `distance_matrix`, на 3 кластера, назначив объектам метки кластеров. Число кластеров, равное 3, выбрано потому, что в этой задаче априори известно число классов (оно равно 3).

В общем случае число кластеров неизвестно, и необходимо использовать какие-либо показатели качества кластеризации для определения оптимального числа кластеров, скрытых в наборе данных.

Так как вид матрицы связей `distance_matrix` зависит от применяемого метода группирования объектов в кластеры, то могут получиться различные разбиения на кластеры при использовании разных методов группирования объектов в кластеры (что можно уже заметить, проанализировав построенные дендрограммы).

В рассматриваемой задаче были получены следующие соотношения для числа объектов в составе кластеров (при разбиении на 3 кластера) для разных методов группирования объектов в кластеры.

`'ward'`: 50:48:52;

`'single'`: 100:49:1;

`'complete'`: 50:29:71;

'average': 50:48:52.

Как известно, для реальных меток классов соотношение для числа объектов в составе классов имеет вид: 50:50:50. Таким образом, можно сделать следующие выводы. Наилучшие результаты показали алгоритмы кластеризации с использованием методов 'ward' и 'average' (они ошиблись относительно кластерной принадлежности всего 2 объектов). Наихудший результат показал алгоритм кластеризации с использованием метода 'single'.

Следует еще раз отметить, что *в общем случае число кластеров неизвестно, и необходимо использовать какие-либо показатели качества кластеризации для определения оптимального числа кластеров*, скрытых в наборе данных. В частности, может быть использован индекс локтя или индекс кластерного силуэта.

3. Итерационные алгоритмы кластерного анализа

Итерационные алгоритмы кластерного анализа – это алгоритмы упорядочивания, реализующие поиск наилучшего разбиения объектов из анализируемого набора данных на кластеры в ходе некоторого числа итераций в соответствии с тем или иным критерием качества кластеризации [6].

В таких алгоритмах требуется задать некоторые начальные условия, с учетом которых и выполняется поиск. Многие из этих алгоритмов используют априори заданное число кластеров (которое, вообще говоря, может быть не оптимальным) для инициализации разбиения объектов на кластеры или, например, для инициализации координат центроидов кластеров, а затем в ходе некоторого числа итераций уточняют разбиение объектов на кластеры или координаты центроидов кластеров. Также априори может быть задан некоторый набор параметров, на основе значений которого реализуется работа итерационного алгоритма и определяется число кластеров, а также их состав, соответствующие априори заданному набору параметров.

Известно большое число итерационных алгоритмов кластерного анализа, среди которых, в первую очередь, следует назвать алгоритм *k* средних (*k*-means-алгоритм) и алгоритм нечетких *c*-средних (fuzzy *c*-means-алгоритм, FCM-алгоритм), отличительными особенностями которых является выделение объектов-эталонов (прототипов), представляющих кластеры. Также следует обратить внимание на DBSCAN-алгоритм (Density-Based Spatial Clustering of Applications with Noise, алгоритм кластеризации, учитывающий плотность расположения объектов в пространстве и выделяющий шумовые объекты), отличительными особенностями которого являются масштабируемость, выделение шумо-

вых объектов и отказ от выделения объектов-эталонов (прототипов), представляющих кластеры.

3.1. Алгоритм *k*-средних (*k*-means-алгоритм)

Алгоритм k-средних, который можно также назвать алгоритмом *чётких* k-средних (в противовес алгоритму *нечётких* *c*-средних) предполагает, что число кластеров k априори известно [7, 8].

Пусть $X = \{x_1, x_2, ..., x_n\}$ – набор данных, содержащий объекты, подлежащие кластеризации, при этом $x_i = (f_1(x_i), f_2(x_i), \ldots, f_g(x_i))$, где n – число объектов; g – число признаков; $f_r(x_i)$ – значение r-го признака i-го объекта; $i = \overline{1,n}$; $r = \overline{1,g}$.

В метрическом пространстве «схожесть» объектов обычно определяется через расстояние, которое может рассчитываться как между исходными объектами, так и между объектами и прототипами (например, центрами) кластеров.

При кластеризации с использованием алгоритма k-средних набор данных X разбивается на подмножества X_j ($j = \overline{1,k}$) так, что:

$$\bigcup_{j=1}^{k} X_j = X, \tag{17}$$

$$X_j \cap X_h = \varnothing,\ j = \overline{1,k},\ h = \overline{1,k},\ j \neq h, \tag{18}$$

$$\varnothing \subset X_j \subset X,\ j = \overline{1,k}. \tag{19}$$

Условие (17) указывает, что все объекты должны быть распределены по кластерам. При этом каждый объект должен принадлежать только одному кластеру, как следует из условия (18) и ни один из кластеров не может быть пустым или содержать все объекты, как следует из условия (19).

Задачу кластеризации удобно формулировать, используя характеристическую функцию $u_j(x_i)$, которая может принимать два значения: 0 – если объект не принадлежит кластеру, и 1 – если объект принадлежит кластеру.

Тогда кластеры могут быть описаны матрицей $U = [u_j(x_i)]$ ($u_j(x_i) \in \{0,1\}$, $j = \overline{1,k}$, $i = \overline{1,n}$), характеризующей k-разбиение набора данных X, где число $u_j(x_i)$ определяет принадлежность i-го объекта j-му кластеру.

Матрица $U = [u_j(x_i)]$ должна обладать такова, что:

$$\sum_{j=1}^{k} u_j(x_i) = 1\ (k \in N \text{ и } k > 1;\ i = \overline{1,n}), \tag{20}$$

$$0 < \sum_{i=1}^{n} u_j(x_i) < n\ (j = \overline{1,k}). \tag{21}$$

Для оценки качества разбиения объектов набора данных на кластеры может быть использована целевая функция, представляющая собой сумму квадратов расстояний и позволяющая оценить разброс данных [7, 8]:

$$J(U,V)=\sum_{j=1}^{k}\sum_{x_i\in X_j} d^2(v_j,x_i)=\sum_{j=1}^{k}\sum_{i=1}^{n}u_j(x_i)\cdot d^2(v_j,x_i), \qquad (22)$$

где $U=[u_j(x_i)]$ – k-разбиение набора объектов X на основе характеристических функций $u_j(x_i)$, определяющих принадлежность объекта $x_i=(f_1(x_i), f_2(x_i),\ldots, f_g(x_i))$ кластеру X_j; $V=(v_1,\ldots,v_k)$ – прототипы (центроиды) кластеров; $d(v_j,x_i)$ – расстояние между объектом x_i и центроидом кластера v_j; $v_j=(f_1(v_j), f_2(v_j),\ldots, f_g(v_j))$ – вектор координат центроида j-го кластера в g-мерном нормированном пространстве, изоморфном R^g ($f_r(v_j)\in R^g$); k – число кластеров; n – число объектов; $j=\overline{1,k}$; $i=\overline{1,n}$.

При использовании метрики евклидова расстояния (2) целевая функция (22) приобретает вид:

$$J(U,V)=\sum_{j=1}^{k}\sum_{x_i\in X_j}\sum_{r=1}^{g}(f_r(x_i)-f_r(v_j))^2=\sum_{j=1}^{k}\sum_{i=1}^{n}\sum_{r=1}^{g}u_j(x_i)\cdot(f_r(x_i)-f_r(v_j))^2. \qquad (23)$$

При этом координаты центроида j-го кластера находятся как:

$$f_r(v_j)=\frac{1}{n_j}\cdot\sum_{x_i\in X_j}f_r(x_i)=\frac{\sum_{i=1}^{n}u_j(x_i)\cdot f_r(x_i)}{\sum_{i=1}^{n}u_j(x_i)}, \qquad (24)$$

где n_j – мощность j-го кластера (число объектов, отнесенных к j-му кластеру); $r=\overline{1,g}$.

Задачу кластеризации набора объектов X можно сформулировать как задачу оптимизации: найти матрицу $U=[u_j(x_i)]$, минимизирующую значение целевой функции (22).

Алгоритм k-средних предполагает выполнение следующих шагов.

1. Инициализация начального разбиения $U=[u_j(x_i)]$, удовлетворяющего условиям (20) и (21).
2. Вычисление координат центроидов кластеров:

$$f_r(v_j)=\frac{\sum_{i=1}^{n}u_j(x_i)\cdot f_r(x_i)}{\sum_{i=1}^{n}u_j(x_i)}.$$

3. Вычисление новых значений характеристических функций с учетом расстояний до центроидов кластеров: *i*-й объект принадлежит к *j*-му тому кластеру, к центроиду которого он ближе.

4. Повторение шагов 2 и 3 до тех пор, пока не будет выполнено заданное число итераций s или не будет достигнута заданная точность $\left|J(U,V) - J'(U,V)\right| \le \varepsilon$, где $J(U,V)$, $J'(U,V)$ – значения целевой функции на двух последовательных итерациях.

При применении алгоритма k-средних определяются локально-оптимальное разбиение, описываемое совокупностью характеристических функций, и координаты центроидов кластеров. Для получения адекватных результатов кластеризации необходимо многократное выполнение алгоритма k-средних при заданном числе кластеров для различных исходных разбиений для принятия окончательного решения об искомой кластеризации.

Следует отметить, что работа алгоритма k-средних может быть начата с инициализации центроидов кластеров с последующим распределением объектов набора данных в кластеры с учетом расстояния до центроидов.

Для оценки качества кластеризации могут использоваться различные показатели качества.

Дискретный характер четкого разбиения приводит к трудностям нахождения оптимальной кластеризации из-за негладкости целевой функции (используемого показателя качества кластеризации).

3.2. Алгоритм нечетких с-средних (fuzzy c-means-алгоритм)

Требование нахождения однозначной кластеризации объектов анализируемого набора данных является достаточно грубым и жестким, особенно при решении плохо или слабо структурированных задач. Методы и алгоритмы нечеткой кластеризации ослабляют это требование посредством введения в рассмотрение нечетких кластеров и соответствующих им функций принадлежности, принимающих значения из отрезка $[0,1]$.

Концептуальная взаимосвязь между кластерным анализом и теорией нечётких множеств основана на том, что при решении задач структуризации сложных систем большинство формируемых классов объектов размыты по своей природе. Эта размытость состоит в том, что переход от принадлежности к непринадлежности объектов к этим классам постепенен, а не скачкообразен.

Алгоритм нечетких c-средних (fuzzy c-means-алгоритм, FCM-алгоритм) предполагает минимизацию целевой функции, представляющей собой сумму *взвешенных* квадратов расстояний [9]:

$$J(U,V) = \sum_{j=1}^{c}\sum_{i=1}^{n}(u_j(x_i))^m \cdot d^2(v_j, x_i) \qquad (25)$$

при

$$\sum_{j=1}^{c} u_j(x_i) = 1 \ (c \in N \ \text{и} \ c > 1; \ i = \overline{1,n}), \qquad (26)$$

где $U = [u_j(x_i)]$ – нечеткое c-разбиение набора объектов на основе функций принадлежности $u_j(x_i)$, определяющих степень принадлежности i-го объекта j-му кластеру; $V = (v_1,...,v_c)$ – центроиды кластеров; $d(v_j, x_i)$ – расстояние между объектом x_i и центроидом кластера v_j; $v_j = (f_1(v_j), f_2(v_j),..., f_g(v_j))$ – вектор координат центроида j-го кластера в g-мерном нормированном пространстве, изоморфном R^g ($f_r(v_j) \in R^g$); m – фаззификатор ($m \in R$, $m > 1$); c – число кластеров; n – число объектов; $j = \overline{1,c}$; $i = \overline{1,n}$.

Следует отметить, что условие (26) определяет нечеткое разбиение набора объектов X в виде: $\bigcup_{j=1}^{c} X_j = X$.

Функции принадлежности играют роль весовых коэффициентов, определяя степень вклада объекта в оценку центроидов кластеров и, соответственно, степень принадлежности i-го объекта j-му кластеру. Размер вклада зависит от выбора фаззификатора m, управляющего степенью «нечеткости».

При малых значениях m нечеткое разбиение вырождается в четкое, при $m \to \infty$ степени принадлежности объектов каждому кластеру становятся равны $1/c$, то есть каждый объект равновероятно принадлежит каждому кластеру. Обычно в качестве значения фаззификатора m выбирается значение, равное 2.

Задачу кластеризации набора объектов X можно сформулировать как задачу оптимизации: найти матрицу $U = [u_j(x_i)]$, минимизирующую значение целевой функции (25).

Алгоритм нечетких c-средних предполагает выполнение следующих шагов [9].

1. Инициализация начального нечеткого разбиения $U = [u_j(x_i)]$, удовлетворяющего условию (26).

2. Вычисление координат центроидов кластеров:

$$f_r(v_j) = \frac{\sum_{i=1}^{n} u_j(x_i)^m \cdot f_r(x_i)}{\sum_{i=1}^{n} u_j(x_i)^m}. \qquad (27)$$

3. Вычисление новых значений функций принадлежности:

$$u_j(x_i) = \frac{1}{\sum_{t=1}^{c}\left(\frac{d(v_j, x_i)}{d(v_t, x_i)}\right)^{\frac{2}{m-1}}}. \qquad (28)$$

4. Повторение шагов 2 и 3 до тех пор, пока не будет выполнено заданное число итераций s или не будет достигнута заданная точность $|J(U,V) - J'(U,V)| \le \varepsilon$, где $J(U,V)$, $J'(U,V)$ – значения целевой функции на двух последовательных итерациях.

При применении FCM-алгоритма определяются локально-оптимальное нечеткое разбиение, описываемое совокупностью функций принадлежности, и координаты центроидов кластеров. Для получения адекватных результатов нечеткой кластеризации необходимо многократное выполнение FCM-алгоритма при заданном числе кластеров для различных исходных нечетких разбиений для принятия окончательного решения об искомой нечеткой кластеризации.

Следует отметить, что алгоритм k-средних и алгоритм нечетких c-средних очень похожи друг на друга. Основное отличие заключается в способе определения матрицы $U = [u_j(x_i)]$. Можно сказать, что алгоритм k-средних является частным случаем алгоритма нечетких c-средних, когда значения функций принадлежности принимают только два значения – 0 или 1.

3.3. Алгоритм DBSCAN

Алгоритм DBSCAN (Density-Based Spatial Clustering of Applications with Nois) реализует кластеризации объектов с учётом их плотности расположения в пространстве и выделяет шумовые объекты [10, 11].

Рисунок 19. Кластеры сферической формы, которые компакты и хорошо отделимы друг от друга

Рисунок 20. Кластеры произвольной формы

По сути, кластеры – это плотные области в пространстве данных, разделенные областями с меньшей плотностью – шумами. Алгоритм DBSCAN основан на этом интуитивном понятии «кластеров» и «шума». Ключевая идея состоит в том, что для каждого объекта в кластере окрестность заданного радиуса должна содержать хотя бы минимальное число объектов.

Иерархические алгоритмы и итерационные алгоритмы (например, алгоритм *k*-средних) предполагают, что кластеры являются выпуклыми (возможно, имеют гиперсферическую форму, если используется метрика евклидова расстояния), компактными и хорошо отделимыми друг от друга (рис. 19). Существенное влияние на качество работы этих алгоритмов оказывают наличие шума и выбросов в данных.

Реальные наборы данных могут быть таковы, что кластеры, скрытые в них, могут иметь произвольную форму, кроме того, набора данных могут содержать шумовые объекты.

На рис. 20 показан набор данных, содержащий невыпуклые кластеры и выбросы/шумы. Очевидно, что алгоритм *k*-средних при работе с таким набором данных будет испытывать трудности с идентификацией кластеров.

Для работы алгоритма DBSCAN необходимо задать два глобальных параметра – `Eps` и `MinPts` [10, 11].

1. Параметр `Eps`, который определяет расстояние соседства вокруг объекта: если расстояние между двумя объектами меньше или равно `Eps`, то они считаются соседями. Если значение `Eps` выбрано слишком маленьким, большая часть данных будет рассматриваться как выбросы. Если значение `Eps` выбрано очень большим, то кластеры объединятся, и большинство объектов будет в одних и тех же кластерах. При выборе значения `Eps` целесообразно использовать график *k*-расстояний.

2. Параметр MinPts, который определяет минимальное число соседей-объектов на расстоянии Eps. Чем больше набор данных, тем большее значение MinPts должно быть выбрано. Как правило, минимальное значение MinPts может быть получено из числа измерений p в наборе данных как MinPts >= p+1. Очевидно, что минимальное значение MinPts должно быть не менее 3.

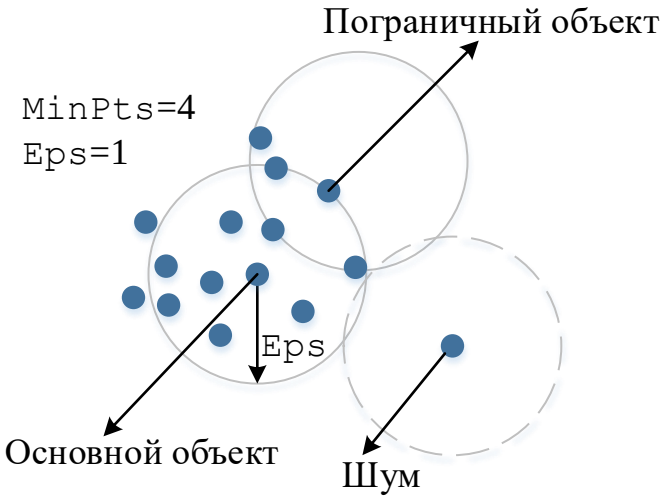


Рисунок 21. Примеры объектов разных типов

В алгоритме DBSCAN выделяют 3 типа объектов [10, 11].

1. Основной объект: объект является основным, если он имеет более чем MinPts объектов-соседей в пределах расстояния Eps.

2. Пограничный объект: объект, который имеет менее MinPts объектов-соседей в пределах расстояния Eps, но находится по соседству с основным объектом.

3. Шум (шумовой объект, выброс): объект, который не является основным или пограничным объектом.

На рис. 21 приведены примеры основного, пограничного и шумового объектов.

Основной факт, опираясь на который, можно распознать кластеры, заключается в том, что внутри каждого кластера имеется типичная плотность объектов, которая значительно выше, чем вне кластера. Кроме того, плотность внутри областей шума ниже, чем плотность в любом из кластеров.

Ключевая идея состоит в том, что для каждого объекта в кластере окрестность на заданном расстоянии должна содержать хотя бы минимальное число объектов, т.е. плотность в окрестности должна превышать некоторый порог [10]. Форма окрестности определяется выбором метрики расстояния $dist(p, q)$ для двух объектов p и q, обозначаемой. Например, при использовании метрики

манхэттенского расстояния (4) в двухмерном пространстве окрестность имеет прямоугольную форму, а при использовании метрики евклидова расстояния (2) окрестность имеет форму окружности.

Определение 1 (Eps-соседство объекта *p*, Eps-окрестность объекта *p*).

Eps-окрестность объекта *p* может быть определена как:

$N_{\mathrm{Eps}}(p) = \{q \in X \mid dist(p, q) \leq \mathrm{Eps}\}$.

Наивный подход мог бы потребовать, чтобы для каждого объекта в кластере существовало по крайней мере минимальное число объектов MinPts в Eps-окрестности этого объекта.

Однако этот подход терпит неудачу, потому что в кластере есть два типа объектов: объекты внутри кластера (основные объекты) и объекты на границе кластера (пограничные объекты). В общем, Eps-окрестность пограничного объекта содержит значительно меньше объектов, чем Eps-окрестность основного объекта. В результате пришлось бы установить минимальное число объектов MinPts, равным очень малому значению, чтобы учесть все объекты, принадлежащие одному и тому же кластеру. Однако это значение не будет характерно для соответствующего кластера, особенно при наличии шума.

Поэтому целесообразно потребовать, чтобы для каждого объекта *p* в кластере *C* существовал объект *q* в кластере *C* такой, чтобы объект *p* находился внутри Eps-окрестности объекта *q* и Eps-окрестность $N_{\mathrm{Eps}}(q)$ содержала не менее MinPts объектов.

Определение 2 (непосредственная (прямая) достижимость по плотности).

Объект *p* непосредственно достижим по плотности из объекта *q* с учетом значений параметров Eps и MinPts, если:

– $p \in N_{\mathrm{Eps}}(q)$;

– $|N_{\mathrm{Eps}}(q)| \geq$ MinPts (условие основного объекта).

Очевидно, что непосредственная достижимость по плотности симметрична для пар основных объектов. Однако, как правило, непосредственная достижимость по плотности не является симметричной, если задействованы один основной объект и один пограничный объект.

Определение 3 (достижимость по плотности). Объект *p* достижим по плотности из объекта *q* с учетом значений параметров Eps и MinPts, если существует цепочка объектов $p_1, ..., p_n$, $p_1 = q$, $p_n = p$ такая, что объект p_i+1 непосредственно достижим по плотности из объекта p_i.

Плотность-достижимость – это каноническое расширение непосредственной плотности-достижимости. Это отношение транзитивно, но не симметрично.

Хотя в целом это отношение не симметрично, очевидно, что оно симметрично для основных объектов.

Два пограничных объекта одного и того же кластера C, возможно, не достижимы друг из друга по плотности, потому что условие основного объекта может не выполняться для них обоих. Однако в кластере C должен быть основной объект, из которого оба пограничных объекта кластера C достижимы по плотности. Введение понятия плотности-связности позволяет охватить это отношение пограничных объектов.

Определение 4 (связность по плотности). Объект p является связанным по плотности с объектом q с учетом значений параметров `Eps` и `MinPts`, если существует объект o такой, что и объект p, и объект q достижимы по плотности из объекта o с учетом значений параметров `Eps` и `MinPts`. Плотность-связность является симметричным отношением. Для достижимых по плотности объектов отношение плотность-связность является рефлексивным.

Интуитивно кластер определяется как набор плотно связанных объектов, который является максимальным в смысле плотности-достижимости. Шум будет определяться относительно этого набора кластеров. Шум – это просто набор объектов в анализируемом наборе данных, не принадлежащих ни к одному из его кластеров.

Определение 5 (кластер). Пусть X – набор данных. Кластер C с учетом значений параметров `Eps` и `MinPts` – непустое подмножество набора X, удовлетворяющее следующим условиям:

– для любых объектов p и q: если $p \in C$ и объект q достижим по плотности из объекта p с учетом значений параметров `Eps` и `MinPts`, то $q \in C$ (*максимальность*);

– для любых объектов $p \in C$, $q \in C$: объект p связан по плотности с объектом q с учетом значений параметров `Eps` и `MinPts` (*связность*).

Определение 6 (шум). Пусть C_1, ..., C_k – кластеры набора данных X с учетом значений параметров $\texttt{Eps}_i$ и $\texttt{MinPts}_i$ ($i = \overline{1,k}$). Тогда шум – это множество объектов набора данных X, не принадлежащих ни одному кластеру C_i, $\texttt{noise} = \{p \in X \mid \forall i : p \notin C_i\}$.

Кластер C с учетом значений параметров `Eps` и `MinPts` содержат как минимум `MinPts` объектов следующим причинам. Поскольку кластер C содержит хотя бы один объект p, то объект p должен быть плотно связан сам с собой через некоторый объект o (который может совпадать с объектом p). Таким образом, по крайней мере объект o должен удовлетворять условию основного

объекта, и, следовательно, Eps-окрестность объекта *o* содержит по крайней мере MinPts объектов.

Учитывая значения параметров Eps и MinPts, можно обнаружить кластер в два этапа. Во-первых, следует выбрать произвольный объект из набора данных, удовлетворяющий условию основного объекта, в качестве начального объекта seed. Во-вторых, следует найти все объекты, достижимые по плотности из начального объекта seed, и получить кластер, содержащий начальный объект seed.

Лемма 1. Пусть p – объект из набора данных X и $|N_{\text{Eps}}(p)| \geq$ MinPts. Тогда множество O = {$o \mid o \in X$ и объект o достижим по плотности из p с учетом значений параметров Eps и MinPts} – кластер с учетом значений параметров Eps и MinPts.

Не очевидно, что кластер C с учетом значений параметров Eps и MinPts однозначно определяется любым из его основных объектов. Однако, каждый объект в кластере C достижим по плотности из любого из основных объектов кластера C, и, следовательно, кластер C содержит ровно те объекты, которые достижимы по плотности из произвольного основного объекта кластера C.

Лемма 2. Пусть C – кластер с учетом значений параметров Eps и MinPts. Пусть p – любой объект кластера C, для которого $|N_{\text{Eps}}(p)| \geq$ MinPts. Тогда кластер C совпадает с множеством O = {$o \mid$ объект o достижим по плотности из объекта p с учетом значений параметров Eps и MinPts}.

В идеале при работе с DBSCAN-алгоритмом необходимо знать значения параметров Eps_i и MinPts_i ($i = \overline{1,k}$) для каждого кластера и как минимум один объект из каждого кластера. Тогда бы можно было найти все объекты, достижимые по плотности из известного объекта кластера, используя правильные значения параметров Eps_i и MinPts_i ($i = \overline{1,k}$). Однако не существует простого способа получения этой информации заранее для всех кластеров набора данных. При этом имеется простая и эффективная эвристика для определения значений параметров Eps и MinPts «самого тонкого», то есть наименее плотного кластера в наборе данных. Параметры плотности «самого тонкого» кластера являются хорошими кандидатами на роль значений глобальных параметров DBSCAN-алгоритма для всех кластеров, так как определяют самую низкую плотность, которая не считается шумом.

Чтобы найти кластер, DBSCAN-алгоритм начинает с произвольного объекта x и находит все объекты, достижимые по плотности, из объекта x с учетом значений параметров Eps и MinPts. Если x является основным объектом, эти

действия приводят к построению кластера, если x – пограничный объект, ни один объект не может быть достигнут по плотности из объекта x и DBSCAN-алгоритм должен перейти к рассмотрению следующего не посещённого объекта набора данных.

DBSCAN-алгоритм может объединить два кластера по определению 5 в один, если два кластера разной плотности находятся «близко» друг к другу. Пусть расстояние между двумя наборами объектов S_1 и S_2 определяется как $dist(S_1, S_2) = min\{dist(p, q) \mid p \in S_1, q \in S_2\}$. Тогда два набора объектов, имеющие, по крайней мере, плотность самого неплотного кластера (самого «тонкого» кластера) будут отделены друг от друга, только если расстояние между двумя наборами больше, чем `Eps`. Следовательно, рекурсивный вызов DBSCAN-алгоритма может быть необходим для обнаруженных кластеров с более высоким значением параметра `MinPts`. Однако это не недостаток, потому что рекурсивное использование DBSCAN-алгоритма дает элегантный и очень эффективный базовый алгоритм. При этом рекурсивная кластеризация объектов кластера необходима только в условиях, которые могут быть легко обнаружены. Ниже на рисунках приведено авторское описание базовой версии DBSCAN-алгоритма [10].

```
DBSCAN (SetOfPoints, Eps, MinPts)

// SetOfPoints is UNCLASSIFIED
  ClusterId := nextId(NOISE);
  FOR i FROM 1 TO SetOfPoints.size DO
    Point := SetOfPoints.get(i);
    IF Point.ClId = UNCLASSIFIED THEN
      IF ExpandCluster(SetOfPoints, Point,
             ClusterId, Eps, MinPts) THEN
        ClusterId := nextId(ClusterId)
      END IF
    END IF
  END FOR
END; // DBSCAN
```

Рисунок 22. Базовая версия DBSCAN-алгоритма

На рис. 22 используются следующие обозначения: `SetOfPoints` – весь набор данных или выявленный кластер из предыдущего запуска алгоритма; `eps` и `MinPts` – глобальные плотностные параметры, определяемые либо вручную,

либо в соответствии со специальной эвристикой. Функция `SetOfPoints.get(i)` возвращает *i*-й объект из `SetOfPoints`. Самая важная функция, используемый в DBSCAN-алгоритме – `ExpandCluster` (рис. 23).

```
ExpandCluster(SetOfPoints, Point, ClId, Eps,
              MinPts) : Boolean;
  seeds:=SetOfPoints.regionQuery(Point,Eps);
  IF seeds.size<MinPts THEN // no core point
    SetOfPoint.changeClId(Point,NOISE);
    RETURN False;
  ELSE   // all points in seeds are density-
         // reachable from Point
    SetOfPoints.changeClIds(seeds,ClId);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
      currentP := seeds.first();
      result := SetOfPoints.regionQuery(currentP,
                                        Eps);
      IF result.size >= MinPts THEN
        FOR i FROM 1 TO result.size DO
          resultP := result.get(i);
          IF resultP.ClId
              IN {UNCLASSIFIED, NOISE} THEN
            IF resultP.ClId = UNCLASSIFIED THEN
              seeds.append(resultP);
            END IF;
            SetOfPoints.changeClId(resultP,ClId);
          END IF; // UNCLASSIFIED or NOISE
        END FOR;
      END IF; // result.size >= MinPts
      seeds.delete(currentP);
    END WHILE; // seeds <> Empty
    RETURN True;
  END IF
END; // ExpandCluster
```

Рисунок 23. Функция `ExpandCluster`

Вызов `SetOfPoints.regionQuery(Point, Eps)` возвращает `Eps`-окрестность объекта в `SetOfPoints` в виде списка объектов. Запросы к областям могут эффективно поддерживаться методами пространственного доступа, такими как R*-деревья [10], которые применяются для эффективной обработки различных типов пространственных запросов. Высота R*-дерева составляет **O**(*log n*) для набора данных из *n* объектов в худшем случае, а запрос с «маленькой» областью запроса проходит только ограниченное число путей в R*-дереве. Поскольку ожидается, что `Eps`-окрестности будут небольшими по сравнению с

размером всего пространства данных, средняя сложность времени выполнения запроса к одной области составляет $O(log\ n)$. Для каждого из n объектов набора данных выполняется не более одного запроса. Таким образом, средняя сложность времени выполнения DBSCAN-алгоритма составляет $O(n \cdot log\ n)$.

Метка кластера `ClId` (`clusterId`) объектов, помеченных как шумы (`NOISE`), может быть изменена позже, если эти объекты доступны по плотности из какого-либо другого объекта набора данных. Это может произойти для пограничных объектов кластера. Такие объекты не добавляются в `seeds`-список, потому что известно, что объект с `ClId`, соответствующим шуму (`NOISE`) не является основном объектом. Добавление этих объектов к `seeds` может привести только к дополнительным запросам, которые не дадут новых ответов.

Если два кластера C_1 и C_2 расположены очень близко друг к другу, может случиться так, что некоторый объект p принадлежит им обоим. Тогда объект x должен быть пограничным объектом в обоих кластерах, иначе кластер C_1 будет совпадать с кластером C_2. В этом случае объект p будет отнесен к кластеру, выявленному первым. За исключением этих редких случаев, результат работы DBSCAN-алгоритма не зависит от порядка посещения объектов набора данных.

При реализации DBSCAN-алгоритма осуществляется рекурсивный поиск всех объектов, связанных по плотности с анализируемым основным объектом и отнесение их тому же кластеру, что и основной объект. Объекты a и b называются связанными по плотности, если существует объект c, который имеет достаточное число объектов в своих соседях, и оба объекта a и b находятся в пределах радиуса `Eps`. Группирование объектов в кластер происходит по цепочке. Если объект b является соседом объекта c, объект c является соседом объекта d, объект d является соседом объекта e, а объект e является соседом объекта a, то объект b является соседом объекта a. При этом выполняется анализ всех непосещённых объектов в наборе данных. Те объекты, которые не принадлежат ни к одному кластеру, полагаются шумовыми.

На рис. 24 приведен пример выявления кластера. При этом `MinPts`. Объект `A` и другие объекты, помеченные красными круглыми маркерами являются основными, так как окружности радиуса `Eps`, окружающие каждый из этих объектов, содержат по меньшей мере 4 объекта (включая сам объект). Эти объекты образуют один кластер, так как все они достижимы друг из друга. Объекты `B` и `C` основными не являются, но достижимы из объекта `A` (через другие основные объекты). Это пограничные объекты. Они также принадлежат кластеру.

Объект `N` является шумовым объектом: он не является ни основным объектом, ни пограничным.

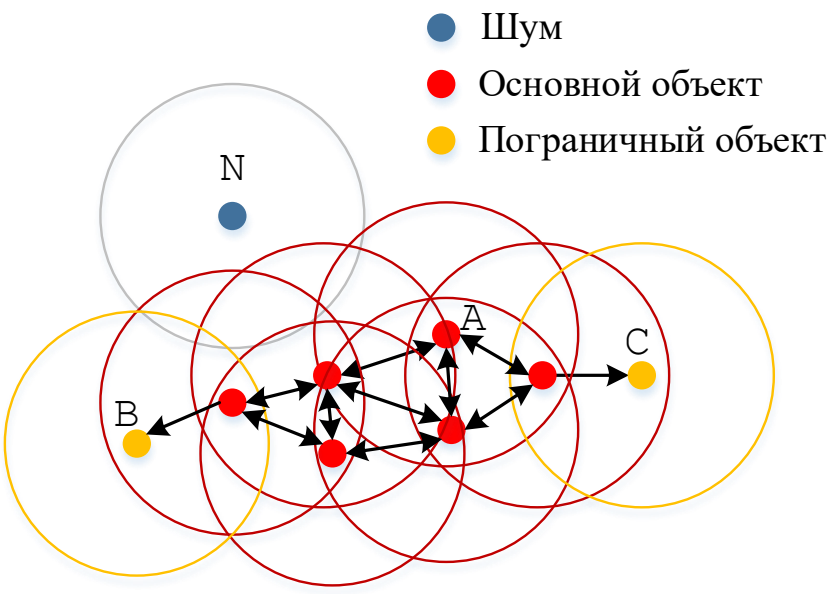


Рисунок 24. Выявление кластера DBSCAN-алгоритмом

Существенный интерес представляет использование эвристики для определения значений параметров `Eps` и `MinPts` самого «тонкого» кластера в наборе данных [10]. Эта эвристика основана на следующем наблюдении. Пусть d – расстояние от объекта p до его k-го ближайшего соседа, тогда d-окрестность объекта p содержит ровно k+1 объект почти для всех объектов p. d-окрестность объекта p содержит более k+1 объекта только в том случае, если несколько объектов находятся на одинаковом расстоянии d от объекта p, что весьма маловероятно. Кроме того, изменение числа k для объекта в кластере не приводит к большим изменениям d. Это происходит только в том случае, если k-е ближайшие соседи объекта p для k = 1, 2, 3,.. расположены примерно на прямой линии, что, вообще говоря, неверно для объекта в кластере. Для заданного числа k можно определить функцию *k-dist* на основе набора данных X, вычисляя для каждого объекта расстояние до его k-го ближайшего соседа. На основе результатов сортировки объектов набора данных в порядке убывания их значений *k-dist* можно построить график и получить некоторые подсказки относительно плотности распределения в наборе данных. Этот график называют *отсортированным графиком k-расстояний*. Если взять произвольный объект p, установить значение параметра `Eps` равным *k-dist*(p), а значение параметр `MinPts` равным k, то все объекты с равным или меньшим значением *k-dist* будут основными объектами. Если бы можно было найти пороговый объект с максимальным значением *k-dist* в самом «тонком»

кластере набора данных *X*, удалось бы найти искомые значения параметров. Пороговый объект – это первый объект в первой «долине» отсортированного графа *k-dist*. Все объекты с более высоким значением *k-dist* (слева от порога) считаются шумом, все остальные объекты (справа от порога) отнесены к некоторому кластеру. Обнаружить первую «долину» автоматически очень сложно, но её относительно легко увидеть в графическом представлении.

3.4. Оценка числа кластеров. Показатели качества кластеризации

При решении задачи кластеризации возможны ситуации, влияющие на результаты кластеризации и искажающие их.

Эти ситуации могут возникнуть из-за: 1) ошибочно выделенных признаков исходных объектов (что часто бывает при выполнении векторного представления текстов); 2) неверно определенного метода сравнения объектов; 3) неверного выбора алгоритма кластеризации для конкретного набора данных.

Для уверенности в том, что кластерная структура корректна и пригодна для дальнейшего использования, необходимо, чтобы она отвечала требованиям, вытекающим из постановки задачи кластеризации, т.е. требованиям компактности и отделимости.

Компактность кластеров подразумевает, что объекты одного кластера должны быть как можно ближе друг к другу. Это свойство может быть выражено: через расстояния между объектами в кластере, через плотность объектов внутри кластера или же через объем, занимаемый кластером в многомерном пространстве.

Отделимость кластеров подразумевает, что расстояние между различными кластерами должно быть как можно больше. Это свойство может быть выражено через расстояние между ближайшими объектами разных кластеров, через расстояние между наиболее удаленными друг от друга объектами кластеров, через расстояние между центроидами кластеров (если они вычисляются алгоритмом).

Для алгоритмов кластеризации, в которых выделяются центроиды кластеров, может использоваться требование о концентрации. *Концентрация* кластеров подразумевает, что объекты каждого кластера должны быть сконцентрированы вокруг центроида своего кластера.

В настоящее время известны различные подходы к оценке качества кластеризации, основанные на применении различных показателей качества кластеризации [12 – 14]. При этом для оценки качества четкой кластеризации (ко-

гда не допускается пересечение кластеров) и нечеткой кластеризации (когда допускается пересечение кластеров) разработаны различные показатели.

Определение числа кластеров может быть выполнено, например, с использованием метода локтя *Elbow* [14].

Оценка качества четкой кластеризации может быть выполнена с использованием таких показателей качества, как: модифицированная Hubert Γ Statistic или ее нормализованная версия $\hat{\Gamma}$; индекс Calinski – Harabasz *CH*; индекс Данна *Dunn*; индекс Дэвида – Болдуина *DB*; индексы *CS* или *PS* validity; *SD* (Scatter – Distance), *SDbw* (Scatter – Density), *RMSSTD* (Root Mean Square Standard Deviation), *RS* (R Squared) индексы; индекс оценки кластерного силуэта *Silh*; индекс *MB* (Maulik – Bandoypadhyay), индекс плотности *CDbw*, общий гиперобъем *H* и др.

Считается, что лучшими по точности оценки являются такие показатели, как: индекс плотности *CDbw*, индекс оценки кластерного силуэта *Silh*, индекс Данна *Dunn* и индекс Дэвида – Болдуина *DB*.

Для оценки качества нечеткой кластеризации используются такие показатели качества кластеризации, как: показатели разбиения *PC* и *HZ*, энтропия разбиения *PE*, индекс Fukuyama – Sugeno *FS*, индекс Се – Бени *XB*, коэффициент нечеткости разбиения *FPC* (fuzzy partition coefficient), индекс отделимости-компактности *SC*, индекс плотности *nCS*, нечеткий общий гиперобъем *FH*, средняя плотность нечеткого разбиения *FAPD*, индекс плотности нечеткого разбиения *FPD*. Одни из них основаны на использовании только степеней принадлежности объектов центроидам кластеров, другие учитывают и степени принадлежности объектов центроидам кластеров, и геометрические свойства как самих объектов кластеризации, так и центроидов кластеров. Кроме того, можно использовать индекс кластерного силуэта *Silh*.

В случае, когда в наборе данных скрыты кластеры гиперсферической формы, в качестве показателя качества кластеризации целесообразно использовать индекс Се – Бени *XB* и его модификации, обеспечивающие получение адекватных результатов кластеризации и характеризующиеся невысокой вычислительной сложностью.

В случае, когда в наборе данных скрыты кластеры гиперэллипсоидной формы, в качестве показателей качества кластеризации следует использовать нечеткий общий гиперобъем *FH*, среднюю плотность разбиения *FAPD*, индекс плотности разбиения *FPD* и индекс плотности *nCS*, обеспечивающие получение адекватных результатов кластеризации, но характеризующиеся более высокой вычислительной сложностью.

Определение числа кластеров можно выполнить, сравнив значения показателя качества кластеризации при различных значениях числа кластеров: искомое число кластеров то, при котором показатель качества достигает своего экстремального значения (минимального или максимального – в зависимости от смысла, заложенного в показатель).

Выбор конкретного показателя качества кластеризации определяется целями выполняемой кластеризации.

Следует отметить, что не существует универсального показателя для оценки качества кластеризации: для любого показателя существует такой набор данных, на котором его оценка качества неверна. В связи с этим для получения обоснованных результатов кластеризации целесообразно использовать не один показатель качества, а их совокупность.

Ниже приведены сведения о некоторых показателях качества кластеризации.

1. Общий гиперобъем *H*.

В качестве показателя качества четкого разбиения рекомендуется использовать общий гиперобъем H найденных кластеров, который должен быть минимизирован:

$$H = \sum_{j=1}^{k} (\det(R_j))^{\frac{1}{2}} \to \min, \tag{29}$$

$$R_j = \frac{1}{n_j} \cdot \sum_{i=1}^{n} (x_i - v_j) \cdot (x_i - v_j)^T, \tag{30}$$

где R_j – ковариационная матрица j-го кластера; n_j – число объектов, отнесенных к j-му кластеру; v_j – вектор координат центроида j-го кластера; x_i – вектор координат (оценок по критериям) i-го объекта; $\det(R_j)$ – определитель ковариационной матрицы j-го кластера; n – число объектов; k – число кластеров; $i = \overline{1,n}$, $j = \overline{1,k}$.

2. Индекс Се-Бени *XB*.

В качестве показателя качества нечеткого разбиения рекомендуется использовать индекс Се-Бени *XB*, который должен быть минимизирован:

$$XB = \frac{J(U,V)}{n \cdot \min\limits_{k \neq t} d(v_k, v_t)} \to \min, \tag{31}$$

где $J(U,V)$ – целевая функция, вычисляемая по формуле (25).

Индекс Се-Бени учитывает, как нечеткие степени принадлежности объектов центроидам кластеров, так и геометрическое расположение центроидов кластеров и объектов, что в большинстве случаев позволяет получить адекватные результаты кластеризации. В результате работы алгоритма нечётких *c*-

средних центроиды искомых кластеров «стягиваются» к скоплениям объектов, обеспечивая при этом минимальное сходство между объектами, принадлежащими разным кластерам.

Считается, что качество кластеризации высокое, если *XB*<1.

3. Коэффициент нечеткости разбиения *FPC*.

Коэффициент нечеткости разбиения *FPC* – еще один активно используемый показатель качества нечеткой кластеризации.

Коэффициент нечеткости разбиения *FPC* должен быть максимизирован:

$$FPC = \frac{1}{n}\sum_{j=1}^{c}\sum_{i=1}^{n}(u_j(x_i))^m \to \max, \qquad (32)$$

где m – фаззификатор; n – число объектов; c – число кластеров; $u_j(x_i)$ – степень принадлежности i-го объекта j-му кластеру.

Коэффициент нечеткости разбиения *FPC* учитывает при оценке качества кластеризации только значения функции принадлежности.

Коэффициент нечеткости разбиения *FPC* принимает значения в диапазоне от 0 до 1. Чем больше значение коэффициента нечеткости разбиения *FPC*, тем выше качество разбиения набора данных на кластеры.

4. Индекс кластерного силуэта *Silh*.

Индекс кластерного силуэта ***Silh*** выполняет учет значений силуэта для каждого объекта кластеризации, поэтому можно оценить как качество отнесения отдельного объекта кластеру, так и качество кластеризации в целом.

Пусть в результате кластеризации объекту x_i $(i=\overline{1,n})$ назначен кластер c_j. Для определения силуэта для каждого объекта x_i $(i=\overline{1,n})$ необходимо вычислить: $a(x_i)$ – среднее расстояние от объекта x_i до других объектов кластера c_j, которому принадлежит объект x_i; $\gamma_\rho(x_i)$ – среднее расстояние от объекта x_i до объектов из кластера c_ρ $(j \neq \rho = \overline{1,c})$, которому не принадлежит объект x_i; $b(x_i) = \min\limits_{\rho \neq j} \gamma_\rho(x_i)$ – показатель различия объекта x_i с объектами ближайшего кластера:

$$a(x_i) = \frac{1}{|c_j|}\sum_{x_l \in c_r} d(x_i, x_l) \ (x_i \in c_j); \qquad (33)$$

$$b(x_i) = \min_{\rho \neq j} \gamma_\rho(x_i) = \min\left\{\frac{1}{|c_j|}\sum_{x_l \in c_r} d(x_i, x_l),\ j \neq \rho = \overline{1,c}\right\}. \qquad (34)$$

Тогда индекс силуэта $sil(x_i)$ каждого объекта x_i $(i=\overline{1,n})$ определяется как:

$$sil(x_i) = \frac{a(x_i) - b(x_i)}{\max(a(x_i), b(x_i))}. \quad (35)$$

При вычислении расстояния между объектами могут использоваться различные метрики, но обычно применяется метрика евклидова расстояния (2).

Значение индекса силуэта лежит в диапазоне $[-1;1]$. Чем ближе значение индекса силуэта $sil(x_i)$ к единице, тем лучше для объекта x_i определена кластерная принадлежность. Отрицательное значение индекса силуэта $sil(x_i)$ свидетельствует о том, что объект x_i плохо кластеризован (кластер для него определен неверно). Значение индекса силуэта $sil(x_i)$, близкое к нулю, означает, что объекту x_i с одинаковой уверенностью могут быть назначены несколько кластеров. Для кластера, состоящего из одного объекта, индекс силуэта считается равным нулю.

Кластерные силуэты индексы силуэта $sil(x_i)$, отсортированные по возрастанию значений внутри каждого кластера. На рис. 25 изображены примеры кластерных силуэтов. По рис. 25 (слева) можно судить о хорошем качестве разбиения объектов на 2 кластера. По рис. 25 (справа) видно, что в кластерах с номерами 0 и 1 есть не очень удачно кластеризованные объекты.

Оценка силуэта для всей кластерной структуры достигается усреднением индексов силуэта по всем объектам:

$$Silh = \frac{1}{n}\sum_{i=1}^{n} sil(x_i). \quad (36)$$

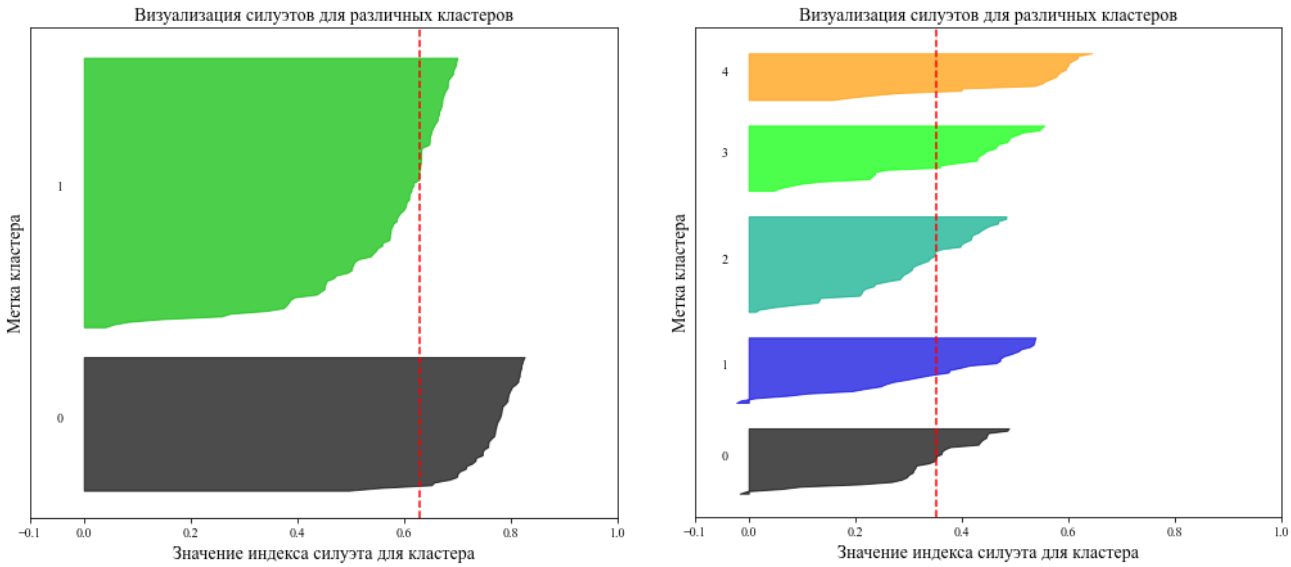


Рисунок 25. Кластерные силуэты:
слева – разделение на 2 кластера,
справа – разделение на 5 кластеров

Лучшее разбиение характеризуется максимально близким к единице значением индекса кластерного силуэта *Silh*, что достигается, когда расстояние между объектами внутри кластеров мало, а расстояние между объектами соседних кластеров велико.

Для использования того или иного показателя качества кластеризации при оценке оптимального числа кластеров необходимо:

– применить алгоритм кластеризации для разных значений числа кластеров;

– для каждого числа кластеров вычислить значение показателя качества кластеризации;

– выбрать в качестве искомого числа кластеров то, при котором достигается экстремум показателя качества кластеризации.

При оценке оптимального числа кластеров может быть использован метод локтя *Elbow*.

Метод локтя *Elbow* основан на расчете суммы квадратов внутрикластерных расстояний – *WCSS* (Within Clusters Sum of Squares). При этом сначала для каждого кластера в отдельности рассчитывается сумма квадратов расстояний от объектов, принадлежащих кластеру, до центроида этого кластера, а затем находится общая сумма этих сумм.

Сумма квадратов расстояний внутри кластера является мерой изменчивости объектов в кластере. Как правило, кластер с небольшой суммой квадратов более компактен, чем кластер с большой суммой квадратов. Кластеры с более высокими значениями демонстрируют большую вариативность объектов внутри кластера.

Сумма квадратов расстояний внутри кластера зависит от числа объектов в нём. По мере увеличения числа объектов сумма квадратов становится больше. Следовательно, сумма квадратов расстояний внутри кластера часто не может быть напрямую использована при анализе кластеров с разным числом объектов. Чтобы сравнить внутрикластерную изменчивость разных кластеров, целесообразно использовать среднее расстояние от объектов, принадлежащих кластеру, до центроида этого кластера.

Для использования метода локтя *Elbow* при оценке числа кластеров необходимо:

– применить алгоритм кластеризации для разных значений числа кластеров j ($j = \overline{2,k}$, k – максимальное число кластеров);

– для каждого числа кластеров вычислить общую сумму квадратов внутрикластерных расстояний – *WCSS*;

– построить графическую зависимость для *WCSS* при разных значениях числа кластеров;

– выполнить анализ графической зависимости для *WCSS*: расположение изгиба (локтя) на участке обычно рассматривается как индикатор соответствующего числа кластеров.

При использовании метода локтя *Elbow* можно вычислить разности $\Delta_{j-1,j}$ между значениями *WCSS* для каждой пары (j – 1, j), где j – число кластеров ($j = \overline{3,k}$), чтобы численно оценить «резкий скачок» для значений *WCSS*. Для значений разностей $\Delta_{j-1,j}$ можно также построить графическую зависимость, что позволит получить визуальное подтверждение для решения по оценке числа кластеров.

3.5. Программная реализация *k*-means-алгоритма на языке Python

Для работы с *k*-means-алгоритмом можно воспользоваться различными программными библиотеками.

В частности, можно использовать ту же программную библиотеку, которая была рекомендована для работы с алгоритмами иерархической агломеративной кластеризации следует воспользоваться программной библиотекой. Эта библиотека доступна по ссылке:

https://pypi.org/project/scipy/.

Для установки программной библиотеки необходимо выполнить команду:

```
pip install scipy
```

Подробная документация по функции, реализующей *k*-means-алгоритм, приведена по ссылке [15]:

https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.vq.kmeans.html#scipy.cluster.vq.kmeans

Кроме того, можно использовать программную библиотеку `scikit-learn`, предоставляющую различные инструменты для прогнозного анализа данных, в том числе, инструменты машинного обучения. Эта библиотека доступна по ссылке:

https://scikit-learn.org/stable/

Для установки программной библиотеки необходимо выполнить команду:

```
pip install -U scikit-learn
```

Подробная документация по программной реализации *k*-means-алгоритма приведена по ссылке [16]:

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans

Ниже представлена информация по классу `KMeans`, реализующему *k*-means-алгоритм.

Синтаксис:

class sklearn.cluster.KMeans(*n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='auto'*)

Входные параметры:

n_clusters – число кластеров, которые нужно сформировать, а также число центроидов, которые нужно сгенерировать;

init – метод инициализации:

'k-means++' («умная» инициализация центроидов кластеров для ускорения сходимости: центроиды выбираются таким образом, чтобы получить доказуемую верхнюю границу *WCSS* (Within Clusters Sum of Squares) – суммы квадратов внутрикластерных расстояний);

'random' (случайный выбор *n_clusters* объектов из набора данных для инициализации центроидов кластеров);

n_init – число запусков алгоритма *k*-средних с разными инициализациями центроидов, при этом итоговым результатом кластеризации будет лучший результат, полученный в ходе *n_init* последовательных запусков с точки зрения показателя инерции;

max_iter – максимальное число итераций алгоритма *k*-средних за один запуск;

tol – относительное допустимое отклонение для нормы Фробениуса при оценке различия в координатах центроидов кластеров на двух последовательных итерациях для объявления сходимости;

verbose – параметр, отвечающий за режим детализации;

random_state – параметр, отвечающий за генерацию случайных чисел для инициализации центроидов (*None* или некоторое целое число, позволяющее обеспечить воспроизводимость результатов эксперимента за счёт детерминированности);

copy_x – параметр, определяющий, будет ли создаваться резервная копия набора данных перед тем, как будут вычисляться расстояния между объектами (при вычислении расстояний целесообразно сначала центрировать данные, если параметр *copy_x* имеет значение *True* (по умолчанию), исходные данные не изменяются, если параметр *copy_x* имеет значение *False*, исходные данные

изменяются и трансформируются обратно до того, как будет выведен результат, но при этом могут появиться небольшие числовые различия из-за выполнения операций вычитания и сложения; при этом есть ряд ситуаций, связанных с типом данных, при которых создается копия, даже если `copy_x=False`);

`algorithm` – используемая версия *k*-means-алгоритма:

`'auto'` (в настоящий момент – алгоритм `'elkan'`, который может быть в будущем заменен на другой для обеспечения лучшей эвристики);

`'full'` (классический алгоритм в EM-стиле);

`'elkan'` (алгоритм, наиболее эффективный для данных с четко определенными кластерами за счет использования неравенства треугольника, но требующий больших объемов памяти из-за необходимости выделения дополнительного массива размером `n_samples x n_clusters` (`n_samples`=*n*, `n_clusters`=*k*));

Значения входных параметров класса `KMeans`, используемые по умолчанию, указаны при описании его синтаксиса.

«*», фигурирующая в списке входных параметров, показывает, что параметр, стоящей перед ней можно передавать как по значению, так и по имени, а параметры, стоящие после нее, можно передавать только по имени.

Выходные параметры:

`cluster_centers_` – матрица размером `n_clusters x n_features` (`n_clusters`=*k*, `n_features`=*p*), содержащая координаты центроидов кластеров;

`labels_` – вектор длиной `n_samples`, где `n_samples` – число объектов в наборе данных (`n_samples`=*n*), содержащий метки кластеров для объектов;

`inertia_` – общая сумма квадратов расстояний от объектов до центроида ближайшего к ним кластера, взвешенная по весам объектов, если они предоставлены;

`n_iter_` – число итераций;

`n_features_in_` – число признаков во время обучения;

`feature_names_in_` – вектор названий признаков длиной `n_features_in_` во время обучения, определяемый, если в наборе данных названия признаков являются строками.

Параметр `inertia_`, содержащий значение общей суммы квадратов расстояний от объектов до центроида ближайшего к ним кластера, может быть использован при применении метода локтя *Elbow* при оценке оптимального числа кластеров.

Методы

1. `fit(X, y=None, sample_weight=None).`

Реализует *k*-means-алгоритм.

Входные параметры:

X – матрица размером `n_samples x n_features` (`n_samples`=*n*, `n_features`=*p*), содержащая набор данных, на основе которого происходит обучение;

y – параметр, который не используется, присутствует здесь для согласованности API по соглашению;

`sample_weight` – массив длиной `n_samples` (`n_samples`=*n*), содержащий веса для каждого объекта набора данных (если `None`, всем объектам присваивается одинаковый вес).

Значения входных параметров метода `fit`, используемые по умолчанию, указаны при описании его синтаксиса.

Выходной параметр:

`self` – обученный кластеризатор.

2. `fit_predict(X, y=None, sample_weight=None)`

Вычисляет центроиды кластеров и предсказывает метку кластера для каждого объекта из набора данных *X*.

Входные параметры:

X – матрица размером `n_samples x n_features` (`n_samples`=*n*, `n_features`=*p*), содержащая набор данных;

y – параметр, который не используется, присутствует здесь для согласованности API по соглашению;

`sample_weight` – массив длиной `n_samples` (`n_samples`=*n*), содержащий веса для каждого объекта набора данных (если `None`, всем объектам присваивается одинаковый вес).

Значения входных параметров метода `fit_predict`, используемые по умолчанию, указаны при описании его синтаксиса.

Выходной параметр:

`labels` – вектор размером `n_samples`, содержащий метки кластеров объектов набора данных.

3. `fit_transform(X, y=None, sample_weight=None)`

Вычисляет результаты кластеризации и преобразует набор данных *X* в пространство кластерных расстояний.

Входные параметры:

X – матрица размером *n_samples x n_features* (*n_samples*=*n*, *n_features*=*p*), содержащая набор данных;

y – параметр, который не используется, присутствует здесь для согласованности API по соглашению;

sample_weight – массив длиной *n_samples* (*n_samples*=*n*), содержащий веса для каждого объекта набора данных (если *None*, всем объектам присваивается одинаковый вес).

Значения входных параметров метода fit_transform, используемые по умолчанию, указаны при описании его синтаксиса.

Выходной параметр:

X_new – матрица размером *n_samples x n_clusters*, содержащий матрицу *X*, переведенную в новое пространство.

4. get_params(*deep=True*)

Выводит параметры кластеризатора.

Входной параметр:

deep – параметр, отвечающий за возврат параметров кластеризатора (вывод осуществляется, если *deep=True*).

Значение входного параметра метода get_params, используемое по умолчанию, указано при описании его синтаксиса.

Выходной параметр:

params – словарь, в котором имена параметров сопоставлены с их значениями.

5. predict(*X, sample_weight=None*)

Предсказывает ближайший кластер, к которому принадлежит каждый объект из набора данных *X*.

Входные параметры:

X – матрица размером *n_samples x n_features* (*n_samples*=*n*, *n_features*=*p*), содержащая набор данных;

sample_weight – массив длиной *n_samples* (*n_samples*=*n*), содержащий веса для каждого объекта набора данных (если *None*, всем объектам присваивается одинаковый вес).

Значения входных параметров метода predict, используемые по умолчанию, указаны при описании его синтаксиса.

Выходной параметр:

labels – вектор размером *n_samples*, содержащий метки кластеров объектов набора данных.

6. score(*X, y=None, sample_weight=None*)

Вычисляет значение суммы, взятой со знаком минус, для квадратов расстояний от объектов до центроида ближайшего к ним кластера, взвешенной по весам объектов, если они предоставлены (значение выходного параметра *inertia_*, взятое со знаком минус).

Входные параметры:

X – матрица размером *n_samples x n_features* (*n_samples*=*n*, *n_features*=*p*), содержащая набор данных, на основе которого происходит обучение;

y – параметр, который не используется, присутствует здесь для согласованности API по соглашению;

sample_weight – массив длиной *n_samples* (*n_samples*=*n*), содержащий веса для каждого объекта набора данных (если *None*, всем объектам присваивается одинаковый вес).

Значения входных параметров метода score, используемые по умолчанию, указаны при описании его синтаксиса.

Выходной параметр:

score – сумма, взятая со знаком минус, для квадратов расстояний от объектов до центроида ближайшего к ним кластера, взвешенной по весам объектов, если они предоставлены (*score= - inertia_*).

7. set_params(params)**

Устанавливает параметры кластеризатора.

Входной параметр:

**params – словарь с параметрами кластеризатора.

Выходной параметр:

self – экземпляр кластеризатора.

8. transform(*X*)

Преобразует *X* в пространство кластерного расстояния.

Входной параметр:

X – матрица размером *n_samples x n_features* (*n_samples*=*n*, *n_features*=*p*), содержащая набор данных, на основе которого происходит обучение.

Выходной параметр:

X_new – матрица размером *n_samples x n_clusters*, содержащий матрицу *X*, переведенную в новое пространство.

Альтернативно можно использовать еще одну программную реализацию *k*-means-алгоритма – Mini-Batch *k*-Means-алгоритм.

Подробная документация по программной реализации Mini-Batch *k*-Means-алгоритм приведена по ссылке:

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html.

Этот алгоритм реализует кластеризацию с использованием выборок (мини-пакетов) из исходного набора данных. Размер таких выборок – `batch_size`, является одним из входных параметров Mini-Batch *k*-Means-алгоритма.

3.6. Программная реализация fuzzy-*c*-means-алгоритма на языке Python

Для работы с fuzzy-*c*-means-алгоритмом можно воспользоваться различными программными библиотеками.

В частности, можно использовать программную библиотеку, предложенную по ссылке:

https://pypi.org/project/scikit-fuzzy/.

SciKit-Fuzzy – библиотека, реализующая алгоритмы на основе теории нечетких множеств и нечеткой логики.

Для установки программной библиотеки необходимо выполнить команду:

```
pip install scikit-fuzzy
```

Подробная документация по библиотеке приведена по ссылке [17]:

https://scikit-fuzzy.readthedocs.io/en/latest/api/skfuzzy.html.

Информация по программной реализации fuzzy-*c*-means-алгоритма приведена по ссылке:

https://github.com/scikit-fuzzy/scikit-fuzzy/blob/master/skfuzzy/cluster/_cmeans.py.

Кроме того, можно использовать программную библиотеку, предложенную по ссылке [18]:

https://pypi.org/project/fuzzy-c-means/.

Для установки программной библиотеки необходимо выполнить команду:

```
pip install fuzzy-c-means
```

Информация по программной реализации fuzzy-*c*-means-алгоритма приведена по ссылке:

https://github.com/omadson/fuzzy-c-means.

В целом принципы работы fuzzy-*c*-means-алгоритмом аналогичны принципам работы с *k*-means-алгоритмом.

Для работы с fuzzy-*c*-means-алгоритмом используется класс `FCM`.

Входные параметры:

n_clusters – число кластеров, которые нужно сформировать, а также число центроидов, которые нужно сгенерировать (по умолчанию *n_clusters*=5);

max_iter – максимальное число итераций алгоритма нечетких *c*-средних за один запуск (по умолчанию *max_iter* =150);

max_iter – фаззификатор (по умолчанию *m* =2);

error – относительное допустимое отклонение при оценке различия в нечетких разбиениях объектов на кластеры на двух последовательных итерациях для объявления сходимости;

random_state – параметр, отвечающий за случайный выбор *n_clusters* объектов из набора данных для инициализации центроидов кластеров;

trained – индикатор, определяющий факт обученности.

Выходные параметры:

centers – массив, содержащий координаты центроидов кластеров;

u – матрица, содержащая результаты нечеткого разбиения объектов на кластеры;

labels – вектор, содержащий метки кластеров объектов.

Метод *fit* обеспечивает обучение без учителя на основе анализируемого набора данных с использованием алгоритма нечетких *c*-средних.

3.7. Программная реализация DBSCAN-алгоритма на языке Python

Для работы с DBSCAN-алгоритмом можно использовать программную библиотеку, предложенную по ссылке:

https://scikit-learn.org/stable/.

Для установки программной библиотеки необходимо выполнить команду:

```
pip install -U scikit-learn
```

Подробная документация по программной реализации DBSCAN-алгоритма приведена по ссылке [19]:

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html.

DBSCAN-алгоритм находит основные объекты в областях высокой плотности и строит на их основе кластеры [20]. При этом DBSCAN-алгоритм

наиболее эффективен для наборов данных, содержащих кластеры одинаковой плотности.

Ниже представлена информация по классу DBSCAN, реализующему DBSCAN-алгоритм.

Синтаксис:

```
class sklearn.cluster.DBSCAN(eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None)
```

Входные параметры:

eps – максимальное расстояние между двумя объектами, при котором один считается соседом другого (считается самым важным параметром в алгоритме);

min_samples – число объектов (или общий вес) в окрестности объекта, которую следует рассматривать как основной объект (в это число входит и сам основной объект);

metric – метрика, используемая при расчете расстояния между объектами набора данных (если метрика является строкой или вызывается, она должна быть одной из опций, разрешенных `sklearn.metrics.pairwise_distances` для метрики; если метрика «предварительно вычислена», предполагается, что *X* является матрицей расстояний и должен быть квадратным. *X* может быть глоссарием, в этом случае только «ненулевые» объекты могут считаться соседями);

metric_params – словарь, содержащий дополнительные ключевые аргументы для функции метрики;

algorithm (*'auto', 'ball_tree', 'kd_tree', 'brute'*) – алгоритм, который будет использоваться модулем `NearestNeighbors` для вычисления расстояний между объектами и поиска ближайших соседей;

leaf_size – размер листа, передаваемый в алгоритм BallTree или cKDTree и способный повлиять на скорость построения и запроса, а также на объем памяти, необходимой для хранения дерева;

p – степень метрики Минковского, используемая для вычисления расстояния между объектами;

n_jobs – число параллельных заданий для запуска.

Значения входных параметров класса `DBSCAN`, используемые по умолчанию, указаны при описании его синтаксиса.

Выходные параметры:

*core_sample_indices_* – вектор длиной *n_core_samples*, содержащий индексы основных объектов;

*components_* – вектор длиной *n_core_samples*, содержащий копии каждого основных объектов, найденного в результате обучения;

*labels_* – метки кластеров для каждого объекта в наборе данных, указанном для `fit()`, при этом шумовые объекты имеют метку «–1»;

*n_features_in_* – число признаков во время обучения;

*feature_names_in_* – вектор названий признаков длиной *n_features_in_* во время обучения, определяемый, если в наборе данных названия признаков являются строками.

Методы

`fit(X, y=None, sample_weight=None).`

`fit_predict(X, y=None, sample_weight=None).`

`get_params(deep=True).`

`set_params(**params).`

Все вышеуказанные методы имеют те же по смыслу входные и выходные параметры, что и одноименные методы алгоритма k-средних.

3.8. Программная реализация показателей качества кластеризации

Для работы с показателями качества кластеризации можно воспользоваться их программными реализациями, предложенными по ссылке [21]:

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics.

На рис. 26 приведен перечень показателей качества кластеризации, программная реализация которых имеется в библиотеке `scikit-learn`, с их описанием на английском языке.

Некоторые из этих показателей позволяют оценить качество кластеризации при наличии истинных меток кластеров. В частности, к таким показателям относятся:

– Adjusted Rand index (*ARI*, скорректированный индекс Рэнда, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html#sklearn.metrics.adjusted_rand_score);

– Homogeneity metric (метрика однородности, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.homogeneity_score.html#sklearn.metrics.homogeneity_score);

– Completeness metric (метрика полноты, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.completeness_score.html#sklearn.metrics.completeness_score).

metrics.adjusted_mutual_info_score(...[, ...])	Adjusted Mutual Information between two clusterings.
metrics.adjusted_rand_score(labels_true, ...)	Rand index adjusted for chance.
metrics.calinski_harabasz_score(X, labels)	Compute the Calinski and Harabasz score.
metrics.davies_bouldin_score(X, labels)	Compute the Davies-Bouldin score.
metrics.completeness_score(labels_true, ...)	Completeness metric of a cluster labeling given a ground truth.
metrics.cluster.contingency_matrix(...[, ...])	Build a contingency matrix describing the relationship between labels.
metrics.cluster.pair_confusion_matrix(...)	Pair confusion matrix arising from two clusterings.
metrics.fowlkes_mallows_score(labels_true, ...)	Measure the similarity of two clusterings of a set of points.
metrics.homogeneity_completeness_v_measure(...)	Compute the homogeneity and completeness and V-Measure scores at once.
metrics.homogeneity_score(labels_true, ...)	Homogeneity metric of a cluster labeling given a ground truth.
metrics.mutual_info_score(labels_true, ...)	Mutual Information between two clusterings.
metrics.normalized_mutual_info_score(...[, ...])	Normalized Mutual Information between two clusterings.
metrics.rand_score(labels_true, labels_pred)	Rand index.
metrics.silhouette_score(X, labels, *[, ...])	Compute the mean Silhouette Coefficient of all samples.
metrics.silhouette_samples(X, labels, *[, ...])	Compute the Silhouette Coefficient for each sample.
metrics.v_measure_score(labels_true, ...[, beta])	V-measure cluster labeling given a ground truth.

Рисунок 26. Перечень показателей качества кластеризации

3.9. Пример кластерного анализа с применением *k*-means-алгоритма

Рассмотрим возможности итерационного кластерного анализа с применением *k*-means-алгоритма на примере набора данных «Ирисы Фишера» (The Iris Dataset).

На рис. 27 приведен фрагмент программного кода, в котором реализован расчет значений показателей *WCSS* и *Silh* при разном числе кластеров с целью выбора оптимального числа кластеров. При этом набор данных был предварительно подвергнут масштабированию с применением `MinMaxScaler` (как и в примере с иерархическим кластерным анализом).

Поиск оптимального числа кластеров и максимального значения индекса кластерного силуэта *Silh* может быть реализован за пределами цикла с помощью команд `Silh.index(max(Silh))+2` и `max(Silh)` соответственно.

На рис. 28 приведен фрагмент программного кода, в котором реализован вывод графической зависимости для значений индекса кластерного силуэта *Silh*, вычисленных при разном числе кластеров.

Аналогичный программный код может быть использован для вывода графической зависимости для значений общей суммы квадратов внутрикластерных расстояний *WCSS*, вычисленных при разном числе кластеров.

На рисунках 29 и 30 приведены графические зависимости, построенные для значений индекса кластерного силуэта *Silh* и общей суммы квадратов внутрикластерных расстояний *WCSS*, вычисленных при разном числе кластеров.

На рис. 31 и 32 приведены скриншоты, отражающие вывод значений индекса кластерного силуэта *Silh* и общей суммы квадратов внутрикластерных расстояний *WCSS*, вычисленных при разном числе кластеров.

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
# Расчет показателей WCSS и Silh при разном числе кластеров
WCSS = []
Silh= []
for i in range(2, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++',
                    max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(scaled_data)
    df['cluster_labels'] =kmeans.fit_predict(scaled_data)
    WCSS.append(kmeans.inertia_)
    Silh.append(silhouette_score(scaled_data, df['cluster_labels'],
                         metric='euclidean'))
```

Рисунок 27. Фрагмент программного кода, в котором реализован расчет показателей WCSS и Silh при разном числе кластеров с целью выбора оптимального числа кластеров

```
# Построение графической зависимости для silhouette score
fig = plt.figure(figsize=(5,5))
plt.plot(range(2, 11), Silh)
plt.title('Индекс кластерного силуэта (Silhouette score)')
plt.xlabel('Число кластеров')
plt.ylabel('Silhouette score')
plt.show()
```

Рисунок 28. Фрагмент программного кода, в котором реализован вывод графической зависимости для значений индекса кластерного силуэта Silh, вычисленных при разном числе кластеров

Рисунок 29. Графическая зависимость, построенная для значений индекса кластерного силуэта Silh, вычисленных при разном числе кластеров

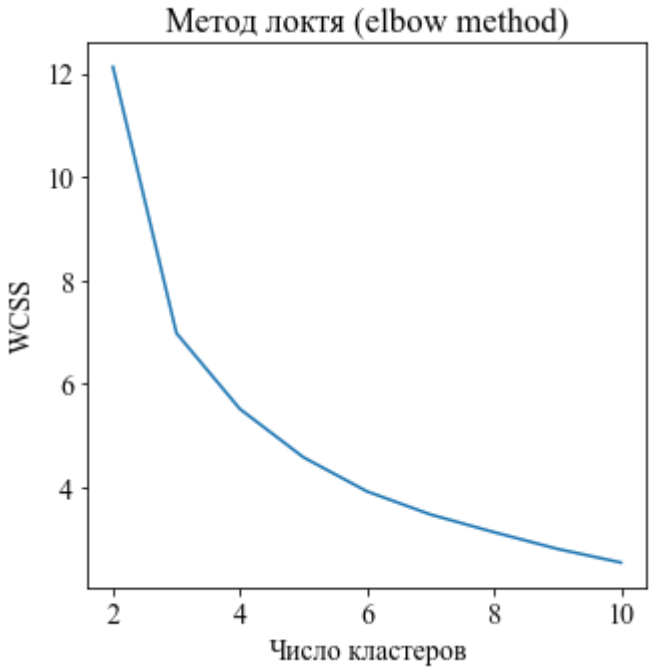


Рисунок 30. Графическая зависимость, построенная для значений общей суммы квадратов внутрикластерных расстояний WCSS, вычисленных при разном числе кластеров

```
Silh
```

```
[0.6300471284354711,
 0.5047687565398589,
 0.44506548804598545,
 0.3474171297756554,
 0.3524618716009237,
 0.34552509197295467,
 0.33580111344286906,
 0.3225035221285963,
 0.32269988264604843]
```

Рисунок 31. Скриншот, отражающий вывод значений индекса кластерного силуэта Silh, вычисленных при разном числе кластеров

```
WCSS
```

```
[12.12779075053819,
 6.982216473785234,
 5.516933472040375,
 4.583940799311901,
 3.924825639027848,
 3.4784804492666934,
 3.136454773254017,
 2.8130466276074837,
 2.5495218922756093]
```

Рисунок 32. Скриншот, отражающий вывод значений общей суммы квадратов внутрикластерных расстояний WCSS, вычисленных при разном числе кластеров

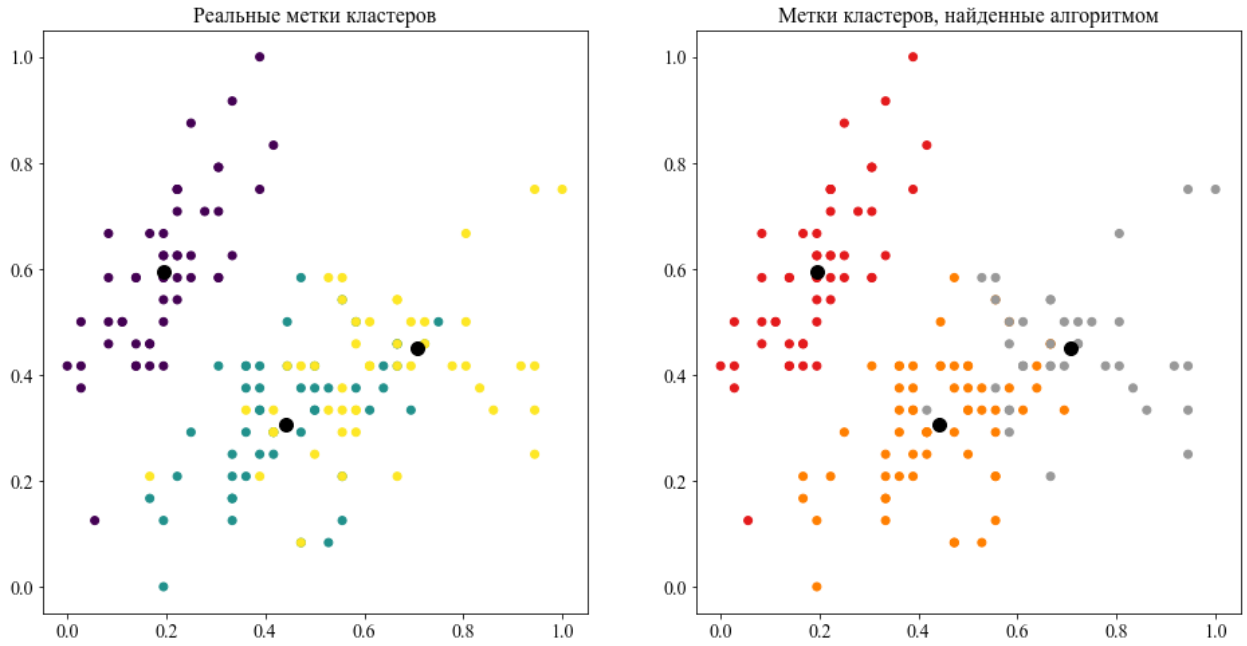


Рисунок 33. Результаты визуализации набора данных по первому и второму признаку при числе кластеров, равном 3

```
df['target'] = iris.target
df['cluster_labels'] =y_kmeans
fig, axes = plt.subplots(1, 2, figsize=(16,8))
axes[0].scatter(scaled_data[:,0], scaled_data[:,1],
                c=df['target'])
# Вывод центроидов кластеров
axes[0].scatter(kmeans.cluster_centers_[:, 0],
                kmeans.cluster_centers_[:,1],
                s = 100, c = 'black', label = 'Centroids')
axes[0].set_title('Реальные метки кластеров', fontsize=16)

axes[1].scatter(scaled_data[:,0], scaled_data[:,1],
                c=df['cluster_labels'], cmap=plt.cm.Set1)
# Вывод центроидов кластеров
axes[1].scatter(kmeans.cluster_centers_[:, 0],
                kmeans.cluster_centers_[:,1],
                s = 100, c = 'black', label = 'Centroids')
axes[1].set_title('Метки кластеров, найденные алгоритмом',
                  fontsize=16)
```

Рисунок 34. Фрагмент программного кода, позволяющий выполнить визуализацию набора данных по первому и второму признаку с выводом центроидов кластеров

Анализ значений индекса кластерного силуэта *Silh* и общей суммы квадратов внутрикластерных расстояний *WCSS*, вычисленных при разном числе кластеров, и графических зависимостей, построенных на их основе, позволяет сделать следующие выводы.

При использовании индекса кластерного силуэта *Silh*, который должен быть максимизирован, в качестве оптимального числа кластеров следует выбрать число, равное 2, при котором значение индекса кластерного силуэта *Silh* равно 0.630 (рис. 31).

При использовании метода локтя *elbow* в качестве оптимального числа кластеров следует выбрать число, равное 3. В этом случае нужно ориентироваться на максимальное изменение общей суммы квадратов внутрикластерных расстояний *WCSS* при увеличении числа кластеров на 1. На рис. 30 максимальному изменению общей суммы квадратов внутрикластерных расстояний *WCSS* соответствует «локоть» (максимальный изгиб на графике).

На рис. 33 представлены результаты визуализации набора данных по первому и второму признаку (в двумерном пространстве) при числе кластеров, равном 3. Кроме того, выполнена визуализация найденных центроидов кластеров, отмеченных на рисунках черными круглыми маркерами. Число кластеров, равное 3, было выбрано потому, что априори известно число классов (кластеров) объектов, скрытых в наборе данных.

На рис. 34 приведен фрагмент программного кода, который реализует возможность визуализации набора данных по первому и второму признаку (в двумерном пространстве) с выводом центроидов кластеров.

3.10. Пример кластерного анализа с применением fuzzy-*c*-means-алгоритма

Рассмотрим возможности итерационного кластерного анализа с применением fuzzy-*c*-алгоритма на примере набора данных «Ирисы Фишера» (The Iris Dataset).

На рис. 35 приведен фрагмент программного кода, в котором реализован расчет индекса кластерного силуэта *Silh* при разном числе кластеров с целью выбора оптимального числа кластеров. При этом набор данных был предварительно подвергнут масштабированию с применением `MinMaxScaler`.

Поиск оптимального числа кластеров и максимального значения индекса кластерного силуэта *Silh* может быть реализован за пределами цикла с помощью команд `Silh.index(max(Silh))+2` и `max(Silh)` соответственно.

Следует отметить, что в используемой программной реализации fuzzy-*c*-алгоритма выполняется всего один запуск алгоритма. Поэтому целесообразно самостоятельно реализовать некоторое число запусков (например, 10) при разных значениях для параметра `random_state` с целью выбора лучшего решения (с точки зрения используемого показателя качества кластеризации) по результатам нескольких запусков.

```
from fcmeans import FCM
from sklearn.metrics import silhouette_score
# Поиск оптимального числа кластеров
Silh= []
for i in range(2, 11):
    fcm=FCM(n_clusters=i, m=2, max_iter = 300, random_state=0)
    fcm.fit(scaled_data)
    df['cluster_labels'] =fcm.predict(scaled_data)
    Silh.append(silhouette_score(scaled_data, df['cluster_labels'],
                          metric='euclidean'))
```

Рисунок 35. Фрагмент программного кода, в котором реализован расчет индекса кластерного силуэта Silh при разном числе кластеров с целью выбора оптимального числа кластеров

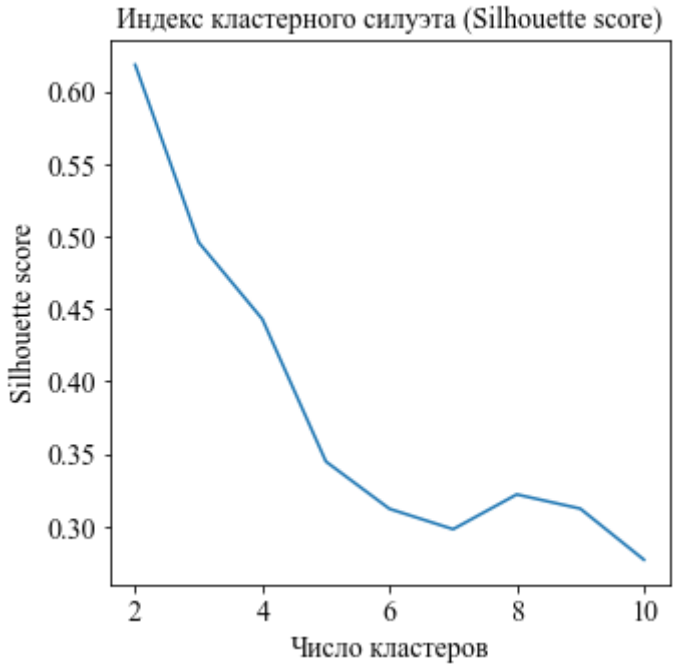


Рисунок 36. Графическая зависимость, построенная для значений индекса кластерного силуэта Silh, вычисленных при разном числе кластеров

На рис.36 приведена графическая зависимость, построенная для значений индекса кластерного силуэта *Silh*, вычисленных при разном числе кластеров.

На рис. 37 приведен скриншот, отражающий вывод значений индекса кластерного силуэта *Silh*, вычисленных при разном числе кластеров.

Анализ значений индекса кластерного силуэта *Silh*, вычисленных при разном числе кластеров, и графической зависимости, построенной на их основе, позволяет сделать следующие выводы.

При использовании индекса кластерного силуэта *Silh*, который должен быть максимизирован, в качестве оптимального числа кластеров следует выбрать число, равное 2, при котором значение индекса кластерного силуэта *Silh* равно 0.618 (рис. 37).

```
Silh
```

```
[0.6184028854416708,
 0.4959228897731569,
 0.44302555258225823,
 0.34478650994076065,
 0.31218066051756665,
 0.2981507745934186,
 0.3221265139074485,
 0.3123249473925689,
 0.2771117166653904]
```

Рисунок 37. Скриншот, отражающий вывод значений индекса кластерного силуэта Silh, вычисленных при разном числе кластеров

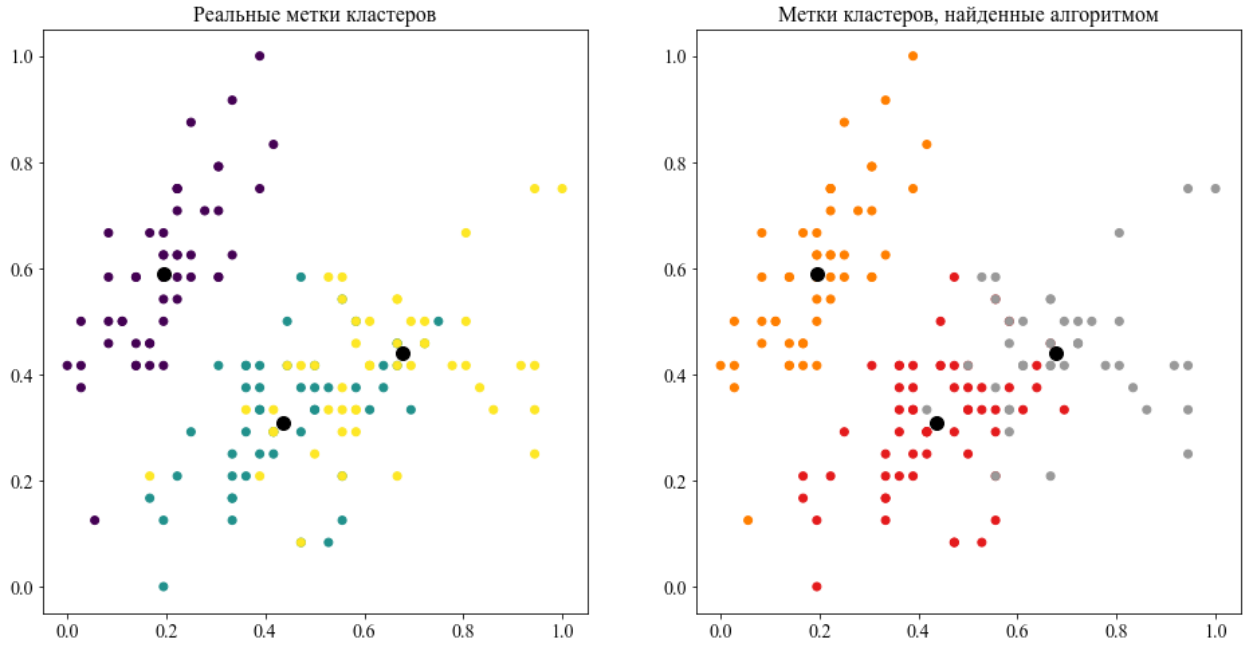


Рисунок 38. Результаты визуализации набора данных по первому и второму признаку при числе кластеров, равном 3

Следует отметить, что значение индекса кластерного силуэта *Silh* для fuzzy-*c*-алгоритма оказалось меньше, чем для *k*-means-алгоритма (0.618 против 0.630). Однако для *k*-means-алгоритма было выполнено 10 запусков, а для fuzzy-*c*-алгоритма – всего один.

Целесообразно провести эксперименты и сравнить полученные результаты кластеризации при одинаковом числе запусков алгоритмов.

На рис. 38 представлены результаты визуализации набора данных по первому и второму признаку (в двумерном пространстве) при числе кластеров, равном 3. Кроме того, выполнена визуализация найденных центроидов кластеров, отмеченных на рисунках черными круглыми маркерами. Число кластеров, равное 3, было выбрано потому, что априори известно число классов (кластеров) объектов, скрытых в наборе данных.

3.11. Пример кластерного анализа с применением DBSCAN-алгоритма

Рассмотрим возможности итерационного кластерного анализа с применением DBSCAN-алгоритма на примере набора данных «Ирисы Фишера» (The Iris Dataset).

При использовании этого алгоритма число кластеров в списке входных параметров в явном виде не указывается. При этом должны быть определены значения 2 глобальных параметра – `Eps` и `MinPts`. В простейшем случае можно организовать перебор значений этих параметров по сетке, а можно воспользоваться какими-либо эвристиками для их определения, например, эвристиками, предложенными авторами алгоритма.

Поскольку число признаков в наборе данных «Ирисы Фишера» равно 4 (g=4), то `MinPts` можно определить, как `MinPts`=g+1=5. Вычислим для каждого объекта расстояние до его 5-го ближайшего соседа, построим матрицу расстояний и на ее основе определим значения для параметра `Eps`.

На рис.39 приведен фрагмент программного кода, позволяющего вычислить для каждого объекта расстояние до его 5-го ближайшего соседа и выполнить визуализацию для результатов расчета.

```
from math import sqrt
```

```
def calculate_kn_distance(X,k):
    kn_distance = []
    for i in range(len(X)):
        eucl_dist = []
        for j in range(len(X)):
            eucl_dist.append(
                math.sqrt(
                    ((X[i,0] - X[j,0]) ** 2) +
                    ((X[i,1] - X[j,1]) ** 2)))

        eucl_dist.sort()
        kn_distance.append(eucl_dist[k])

    return kn_distance
```

```
eps_dist = calculate_kn_distance(scaled_data,5)
plt.hist(eps_dist,bins=30)
plt.ylabel('Число объектов', size=14)
plt.xlabel('Eps расстояние', size=14)
```

Рисунок 39. Фрагмент программного кода, позволяющего вычислить для каждого объекта расстояние до его 5-го ближайшего соседа

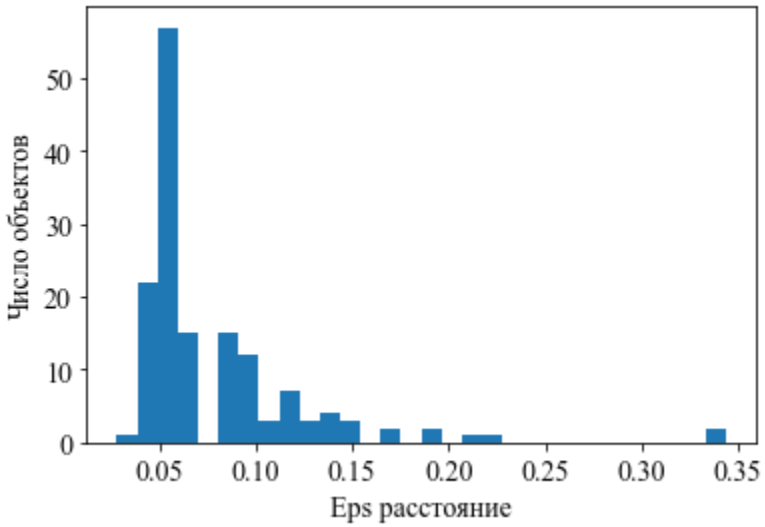


Рисунок 40. Гистограмма, построенная по результатам расчета расстояний до 5-го ближайшего соседа для каждого объекта из набора данных

```
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
db = DBSCAN(eps=0.25, min_samples=5).fit(scaled_data)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

# число кластеров в метках без учета шума, если он имеется
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

labels_true=df['target']
print("Число кластеров: %d" % n_clusters_)
print("Число шумовых объектов: %d" % n_noise_)
print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(labels_true, labels))
print(
    "Adjusted Mutual Information: %0.3f"
   % metrics.adjusted_mutual_info_score(labels_true, labels))
print("Silhouette score: %0.3f" % metrics.silhouette_score(scaled_data, labels))
```

Рисунок 41. Фрагмент программного кода, в котором реализована кластеризация набора данных при `Eps`=0.25 *и* `MinPts`=5.

```
Число кластеров: 2
Число шумовых объектов: 2
Homogeneity: 0.575
Completeness: 0.906
V-measure: 0.704
Adjusted Rand Index: 0.556
Adjusted Mutual Information: 0.699
Silhouette score: 0.577
```

Рисунок 42. Скриншот, отражающий вывод информации о числе кластеров, числе шумовых объектов, а также о значениях различных показателей качества кластеризации
(`Eps`=0.25 и `MinPts`=5)

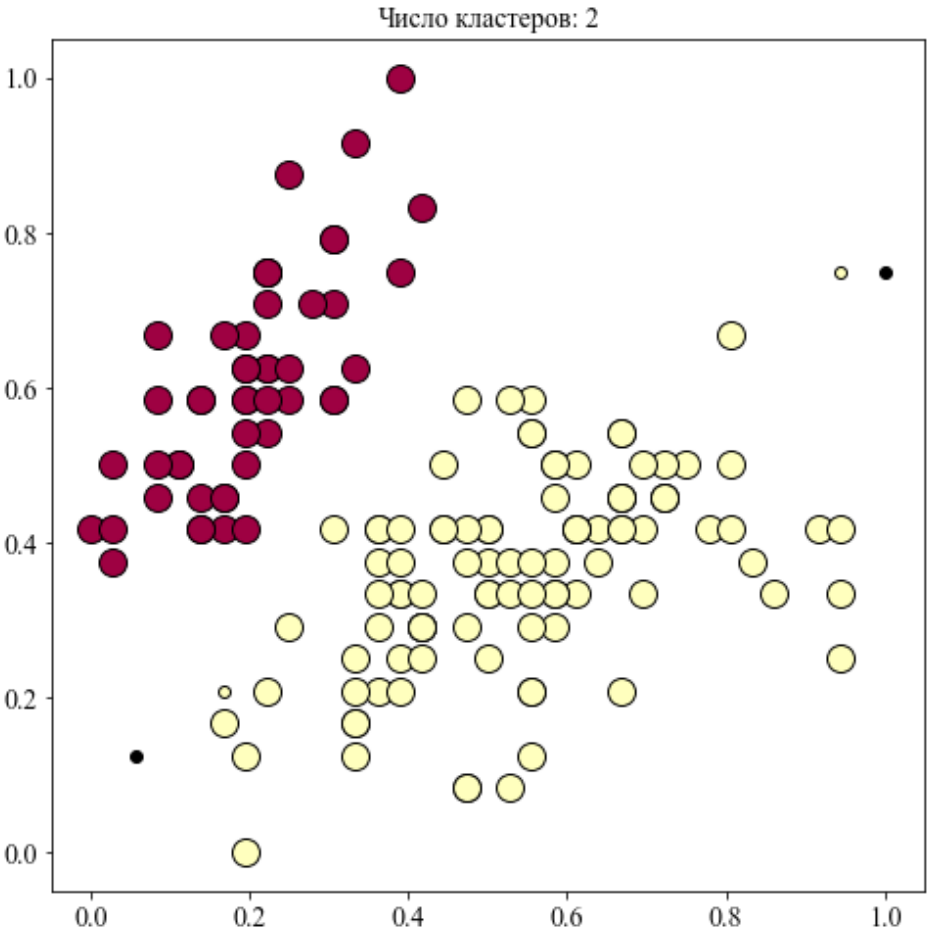


Рисунок 43. Результаты визуализации набора данных по первому и второму признаку при числе кластеров, равном 2
(`Eps`=0.25 *и* `MinPts`=5)

На рис.40 приведена гистограмма, построенная по результатам расчета расстояний до 5-го ближайшего соседа для каждого объекта из набора данных. Анализ гистограммы позволяет сделать следующие выводы: большинство объектов лежит на расстоянии, которое не больше, чем 0.25, поэтому значение для параметра `Eps` можно определить равным 0.25.

На рис. 41 приведен фрагмент программного кода, в котором реализована кластеризация набора данных при `Eps`=0.25 и `MinPts`=5. При этом выполнен оценка качества кластеризации как с применением индекса кластерного силуэта, который не учитывает информацию об истинных метках объектов, так и ряд показателей качества кластеризации, которые используют информацию об истинных метках объектов.

На рис. 42 приведен скриншот, отражающий вывод информации числе кластеров, числе шумовых объектов, а также о значениях различных показателей качества кластеризации.

При `Eps`=0.25 и `MinPts`=5 число кластеров оказалось равным 2, число шумовых объектов – 2.

```
# визуализация
plt.figure(figsize=(8,8))
# задание цвета для маркеров, обозначающих объекты разных кластеров
unique_labels = set(labels)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # черный цвет используется для шумовых объектов
        col = [0, 0, 0, 1]
    class_member_mask = labels == k
    xy = scaled_data[class_member_mask & core_samples_mask]
    plt.plot(
        xy[:, 0],
        xy[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=14
    )

    xy = scaled_data[class_member_mask & ~core_samples_mask]
    plt.plot(
        xy[:, 0],
        xy[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=6,
    )

plt.title("Число кластеров: %d" % n_clusters_, size=14)
plt.show()
```

Рисунок 44. Фрагмент программного кода, который обеспечивает визуализацию набора данных по первому и второму признаку

```
Число кластеров: 2
Число шумовых объектов: 0
Homogeneity: 0.579
Completeness: 1.000
V-measure: 0.734
Adjusted Rand Index: 0.568
Adjusted Mutual Information: 0.732
Silhouette score: 0.630
```

Рисунок 45. Скриншот, отражающий вывод информации о числе кластеров, числе шумовых объектов, а также о значениях различных показателей качества кластеризации (`Eps`=0.35 *и* `MinPts`=5)

На рис. 43 представлены результаты визуализации набора данных по первому и второму признаку (в двумерном пространстве) при числе кластеров, равном 2, при `Eps`=0.35 и `MinPts`=5. Маленькими чёрными круглыми маркерами помечены шумовые объекты. Большими маркерами двух цветов помечены основные объекты. Маленькими маркерами, цвет которых отличен от чёрного, помечены пограничные объекты.

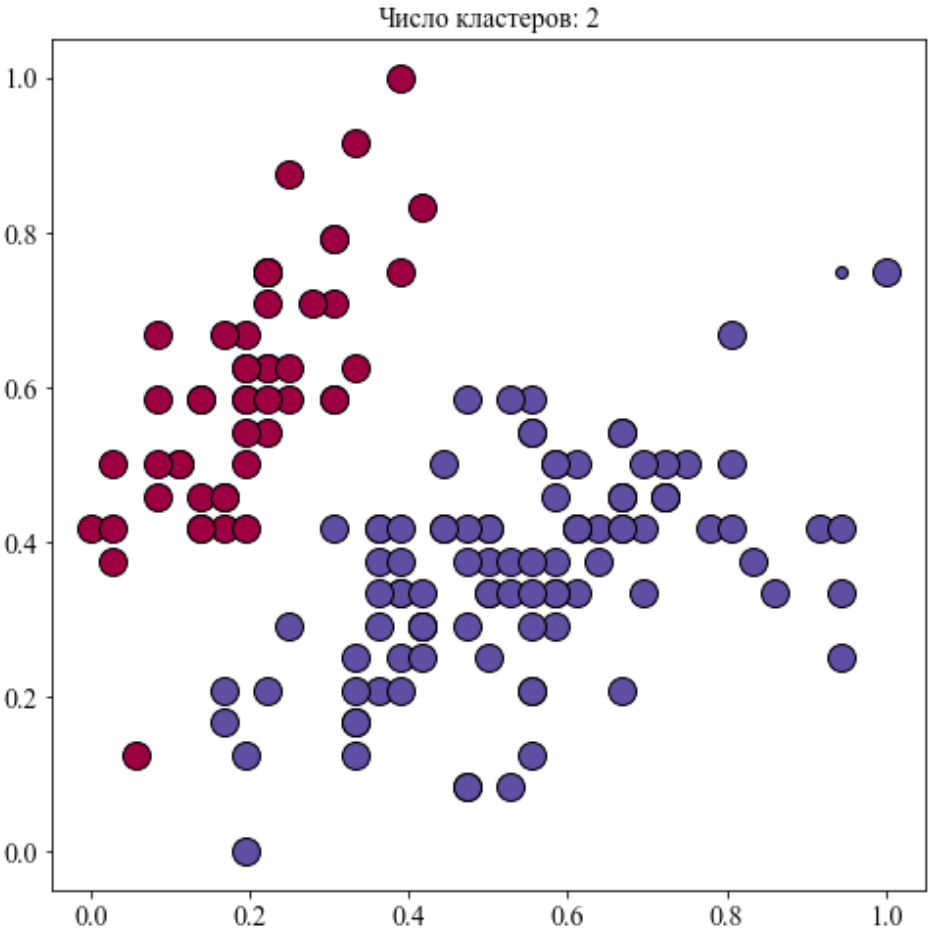


Рисунок 46. Результаты визуализации набора данных по первому и второму признаку при числе кластеров, равном 2
(`Eps`=0.35 и `MinPts`=5)

На рис. 44 приведен фрагмент программного кода, который обеспечивает визуализацию набора данных по первому и второму признаку (в двумерном пространстве).

Для сравнения DBSCAN-алгоритм был реализован при `Eps`=0.35, что позволяет охватить все случаи расчета расстояний до 5-го ближайшего соседа для каждого объекта из набора данных (рис. 40), и `MinPts`=5 число кластеров оказалось равным 2, число шумовых объектов также равно 2.

На рис. 45 приведен скриншот, отражающий вывод информации числе кластеров, числе шумовых объектов, а также о значениях различных показателей качества кластеризации.

При `Eps`=0.35 и `MinPts`=5 число кластеров также оказалось равным 2, а число шумовых объектов оказалось равным 0.

На рис. 46 представлены результаты визуализации набора данных по первому и второму признаку (в двумерном пространстве) при числе кластеров, равном 2, при `Eps`=0.35 и `MinPts`=5.

При `Eps`=0.35 и `MinPts`=5 улучшились значения большинства показателей качества кластеризации. При том значение индекса кластерного силуэта увеличилось и оказалось равным значению индекса кластерного силуэта, полученному при применении *k*-means-алгоритма при числе кластеров, равном 2.

Следует отметить, что неадекватный выбор значений глобальных параметров `Eps` и `MinPts` может привести к тому, что значительная часть объектов в анализируемом наборе данных будет отнесена к шумовым объектам, в результате чего результаты кластеризации не будут отражать реальную структуру кластеров, скрытых в наборе данных.

4. Сложности и проблемы, которые могут возникнуть при выполнении кластерного анализа

Постановка задачи кластеризации сложна и неоднозначна, так как:

– число кластеров в общем случае неизвестно;

– выбор как меры «похожести» или близости признаков объектов набора данных между собой, так и показателя качества кластеризации всегда носит субъективный характер.

Разные алгоритмы кластеризации могут давать различные результаты кластеризации на одних и тех же наборах данных [22, 23].

По следующей ссылке

https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html

можно ознакомиться с примерами применения различных алгоритмов кластеризации к так называемым «игрушечным» наборам данных («toy datasets») в двухмерном пространстве и увидеть различия в результатах кластеризации с применением этих алгоритмов [23].

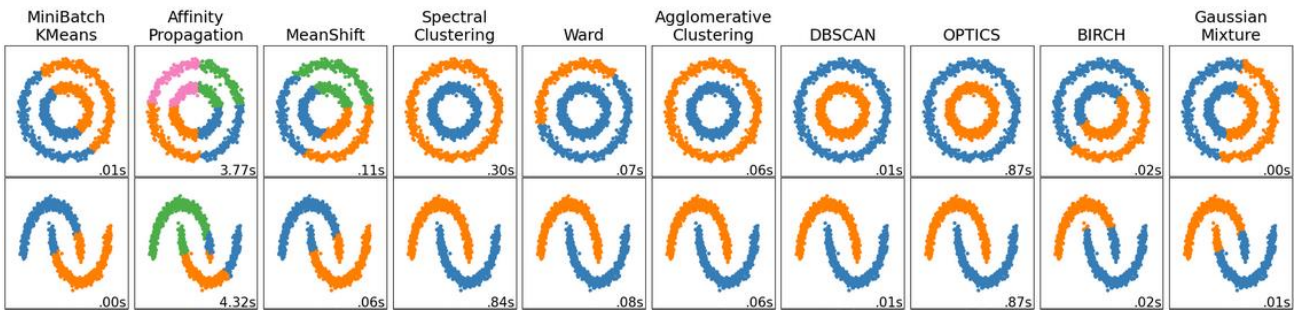


Рисунок 47. Результаты кластеризации 2 наборов данных с применением различных алгоритмов кластеризации

На рис. 47 приведены результаты кластеризации 2 «игрушечных» наборов данных с применением различных алгоритмов кластеризации с указанием времени работы алгоритма. При этом для каждого алгоритма была выполнена настройка значений его параметров с целью получения оптимальных результатов кластеризации. Хотя визуально можно легко выделить кластеры, скрытые в этих наборах данных, не все алгоритмы успешно справились с задачей кластеризации.

Очевидно, что для многомерных наборов данных визуализация в исходном пространстве признаков невозможна. При этом применение различных показателей качества кластеризации не всегда позволяет принять правильные решения о результатах кластеризации как ввиду специфики самих алгоритмов кластеризации, так и ввиду специфики показателей качества кластеризации.

Следует отметить, что для визуализации многомерных наборов данных можно использовать алгоритмы нелинейного снижения размерности, в частности, t-sne- и UMAP-алгоритмы, программные реализации которых предложены соответственно по ссылкам:

https://pypi.org/project/tsne/,

https://pypi.org/project/umap-learn/.

Подробная документация по программным реализациям t-sne- и UMAP-алгоритмов доступна соответственно по ссылкам:

https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html,

https://pypi.org/project/umap-learn/#description.

Показатели качества кластеризации, например, индекс кластерного силуэта *Silh*, оценивают качество кластеризации без использования истинных меток кластеров для объектов: эти метки считаются неизвестными в задаче обучения без учителя.

Метки принадлежности объектов (метки классов) известны в задаче классификации, которая является задачей обучения с учителем. Такие метки используется в процессе обучения для оценки качества разрабатываемого классификатора. Метки классов могут быть назначены объектам, например, человеком-экспертом или каким-либо алгоритмом.

4.1. Проблема выбора показателя качества кластеризации

Показатели качества кластеризации делают все возможное, чтобы предложить (с учетом заложенных в них математических принципов) правильное число кластеров, но могут давать неверные решения, если используются без контекста (например, без учета информации о возможной форме кластеров).

Приведем 2 примера, подтверждающие вышесказанное.

Рассмотрим в двухмерном пространстве работу алгоритма *k*-средних, который очень хорошо решает задачу кластеризации наборов данных, в которых скрыты кластеры сферической формы, при использовании метрики евклидова расстояния (2), а также работу DBSCAN-алгоритма, который учитывает плотность расположения объектов в пространстве признаков и не предъявляет к форме кластеров таких жёстких требований, как алгоритм *k*-средних. Пусть DBSCAN-алгоритм также работает с метрикой евклидова расстояния (2).

```
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons
from sklearn.datasets import make_circles
from sklearn.metrics import silhouette_score
from sklearn.metrics import adjusted_rand_score
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

make_moons

```
features, true_labels = make_moons(n_samples=250, noise=0.05, random_state=42)
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
```

```
# создание экземпляров алгоритмов k-means и DBSCAN
kmeans = KMeans(n_clusters=2)
dbscan = DBSCAN(eps=0.3)

# обучение
kmeans.fit(scaled_features)
dbscan.fit(scaled_features)

# вычисление индекса кластерного силуэта
kmeans_silhouette = silhouette_score(scaled_features, kmeans.labels_).round(2)
dbscan_silhouette = silhouette_score(scaled_features, dbscan.labels_).round (2)
```

```
kmeans_silhouette
```

```
0.5
```

```
dbscan_silhouette
```

```
0.38
```

Рисунок 48. Программный код, в котором решается задача кластеризации «игрушечного» набора данных `make_moons`

В ходе экспериментов можно будет увидеть, что алгоритм *k*-средних плохо справляется с решением задачи кластеризации наборов данных, в которых

скрыты кластеры *несферической* формы, при использовании метрики евклидова расстояния (2), хотя правильное решение легко определяется визуально.

Пусть для оценки качества кластеризации используется индекс кластерного силуэта *Silh*. Вычислим значения индекс кластерного силуэта *Silh* для результатов кластеризации анализируемого набора данных, полученных с применением алгоритма *k*-средних и DBSCAN-алгоритма, и сравним их.

На рис. 48 приведен пример программного кода, в котором решается задача кластеризации «игрушечного» набора данных `make_moons`. При этом выполнено вычисление значений индекс кластерного силуэта *Silh*. Считается, что чем больше значение индекса кластерного силуэта *Silh*, тем лучше качество кластеризации.

В рассматриваемом примере значение индекса кластерного силуэта *Silh* больше для результатов кластеризации с применением алгоритма *k*-средних: оно равно 0.5, в то время как для результатов кластеризации с применением DBSCAN-алгоритма значение индекса кластерного силуэта *Silh* равно 0.38.

Получается, что алгоритм *k*-средних позволил получить результаты кластеризации большее высокого качества. Проверим полученные выводы, выполнив визуализацию результатов кластеризации в двухмерном пространстве (рис. 49). Так ли это?

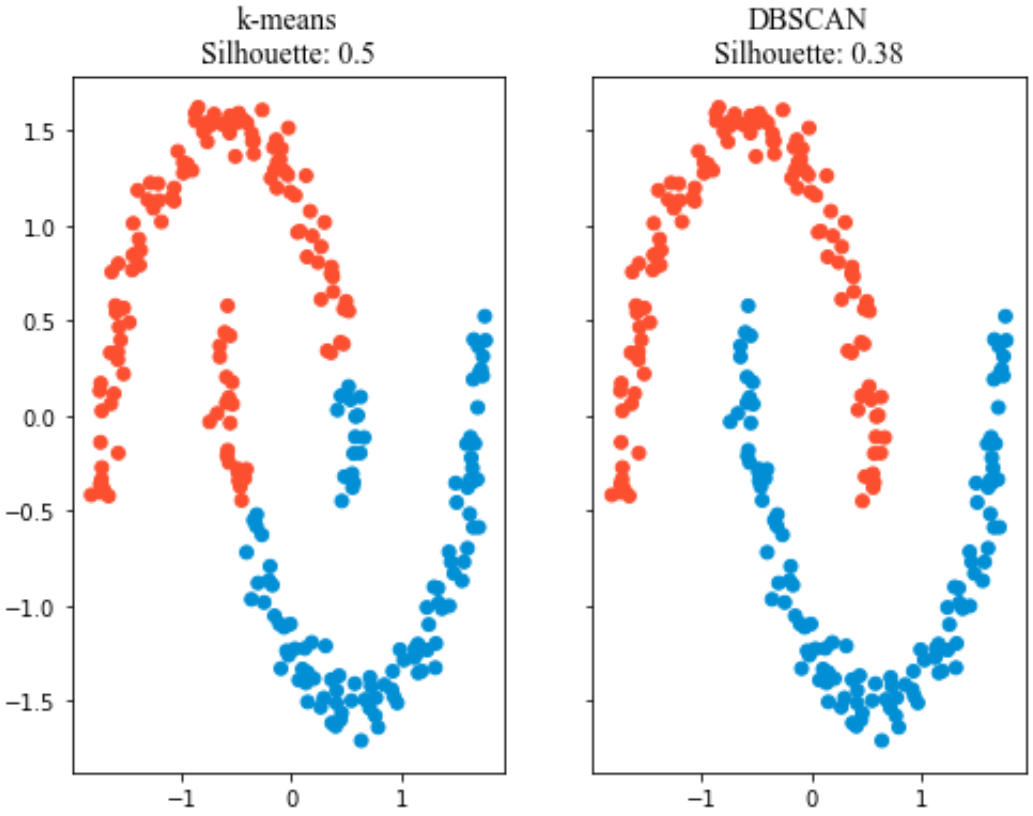


Рисунок 49. Визуализация результатов кластеризации набора данных `make_moons`

```
# визуализация результатов кластеризации
fig, (ax1, ax2) = plt.subplots(
     1, 2, figsize=(8, 6), sharex=True, sharey=True)
fte_colors = {0: "#008fd5", 1: "#fc4f30"}

# визуализация для k-means-алгоритма
km_colors = [fte_colors[label] for label in kmeans.labels_]
ax1.scatter(scaled_features[:, 0], scaled_features[:, 1], c=km_colors)
ax1.set_title("k-means\nSilhouette: {kmeans_silhouette}",
    fontdict={"fontsize": 14, "fontname": 'Times New Roman'})

# визуализация для DBSCAN-алгоритма
db_colors = [fte_colors[label] for label in dbscan.labels_]
ax2.scatter(scaled_features[:, 0], scaled_features[:, 1], c=db_colors)
ax2.set_title(f"DBSCAN\nSilhouette: {dbscan_silhouette}"
    , fontdict={"fontsize": 14, "fontname": 'Times New Roman'})
plt.show()
```

Рисунок 50. Программный код, в котором решается в котором решается задача визуализации результатов кластеризации «игрушечного» набора данных `make_moons`

Анализ результатов кластеризации позволяет отвергнуть сформулированное предположение о том, что алгоритм *k*-средних позволил получить результаты кластеризации большее высокого качества! В рассматриваемом примере большее значение индекса кластерного силуэта *Silh*, равное 0.5, для результатов кластеризации с применением алгоритма *k*-средних приводит к принятию ошибочного решения. Оказывается, что именно DBSCAN-алгоритм находит более естественные кластеры в рассматриваемом примере, поскольку этот алгоритм учитывает плотность расположения объектов в пространстве признаков и не предъявляет к форме кластеров жёстких требований.

На рис. 50 приведен пример программного кода, в котором решается задача визуализации результатов кластеризации «игрушечного» набора данных `make_moons` с применением алгоритма *k*-средних и DBSCAN-алгоритма.

Полученные по результатам визуализации результатов кластеризации выводы, свидетельствуют о необходимости поиска более эффективного метода сравнения качества кластеризации с применением алгоритма *k*-средних и DBSCAN-алгоритма.

Так как в рассматриваемом примере метки объектов известны (`true_labels` в программном коде на рис. 47), то можно использовать метрику кластеризации, которая учитывает такие метки при оценке качества кластеризации. В частности, можно использовать скорректированный индекс Рэнда (Adjusted Rand Index**,** *ARI*), предложенный В.М. Рэндом (W.M. Rand). Информация о программной реализации этого индекса доступна по ссылке:

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html#sklearn.metrics.adjusted_rand_score.

В отличие от индекса кластерного силуэта *Silh*, индекс *ARI* использует истинные метки кластеров для измерения сходства между истинными и предсказанными алгоритмом кластеризации метками.

Скорректированный индекс Рэнда основан на индексе Рэнда, который вычисляет меру подобия между двумя кластеризациями, рассматривая все пары объектов и подсчитывая пары, которые отнесены в одни и те же или в разные кластеры в предсказанных и истинных кластеризациях. При этом в скорректированном индексе Рэнда выполнена корректировка на предмет случайности в соответствии с формулой:

$$ARI = \frac{RI - Expected\_RI}{\max(RI) - Expected\_RI},$$

которая может быть переписана в виде:

$$ARI = \frac{\sum_{ij}\binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}\right] / \binom{n}{2}},$$

где n – число объектов в наборе данных; n_{ij}, a_i, b_j – величины из таблицы сопряженности (n_{ij} – число объектов с истинной меткой кластера *j*, отнесенных к кластеру с меткой *i* алгоритмом кластеризации; a_i – число объектов, отнесенных к кластеру с меткой *i* алгоритмом кластеризации; b_j – число объектов с истинной меткой кластера *j*).

Индекс *ARI* принимает значения в диапазоне [– 1, 1]. Положительное значение индекса *ARI* указывает на то, что правильно сгруппировано в кластеры большее число объектов, чем можно было бы ожидать при случайном группировании, тогда как отрицательное значение индекса *ARI* указывает на то, что правильно сгруппировано в кластеры меньше объектов, чем можно было бы ожидать при случайном группировании, а значение индекса *ARI*, близкое к 0, соответствует случайному группированию объектов в кластеры.

В частности, значение индекса *ARI*, близкое к 1, указывает на идеально сформированные кластеры, когда истинная и предсказанная кластеризации идентичны (с точностью до перестановки).

На рис. 51 приведен фрагмент программного кода с результатами вычисления индекса *ARI* для результатов кластеризации «игрушечного» набора данных `make_moons` с применением алгоритма *k*-средних и DBSCAN-алгоритма.

Анализ значений индекса *ARI* для результатов кластеризации «игрушечного» набора данных `make_moons` с применением алгоритма *k*-средних и

DBSCAN-алгоритма позволяет сделать выводы о том, что DBSCAN-алгоритм обеспечивает лучшие (идеальные) результаты кластеризации «игрушечного» набора данных `make_moons` (по сравнению с результатами с применением алгоритма *k*-средних), т.к. для алгоритма *k*-средних и DBSCAN-алгоритма значение индекса *ARI* равно соответственно 0.47 и 1.0.

```
ari_kmeans = adjusted_rand_score(true_labels, kmeans.labels_)
ari_dbscan = adjusted_rand_score(true_labels, dbscan.labels_)
```

```
round(ari_kmeans, 2)
```

```
0.47
```

```
round(ari_dbscan, 2)
```

```
1.0
```

Рисунок 51. Фрагмент программного кода с результатами вычисления индекса ARI для результатов кластеризации «игрушечного» набора данных `make_moons` *с применением алгоритма k-средних и DBSCAN-алгоритма*

make_circles

```
features, true_labels = make_circles(n_samples=1500, factor=0.5, noise=0.05, random_state=42)
```

```
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
```

```
# создание экземпляров алгоритмов k-means и DBSCAN
kmeans = KMeans(n_clusters=2)
dbscan = DBSCAN(eps=0.3)

# обучение
kmeans.fit(scaled_features)
dbscan.fit(scaled_features)

# вычисление индекса кластерного силуэта
kmeans_silhouette = silhouette_score(scaled_features, kmeans.labels_).round(2)
dbscan_silhouette = silhouette_score(scaled_features, dbscan.labels_).round (2)
```

```
kmeans_silhouette
```

```
0.35
```

```
dbscan_silhouette
```

```
0.11
```

Рисунок 52. Программный код, в котором решается задача кластеризации «игрушечного» набора данных `make_circles`

На рис. 52 приведен пример программного кода, в котором решается задача кластеризации «игрушечного» набора данных `make_circles`. При этом выполнено вычисление значений индекс кластерного силуэта *Silh*.

В рассматриваемом примере значение индекса кластерного силуэта *Silh* больше для результатов кластеризации с применением алгоритма *k*-средних: оно равно 0.35, в то время как для результатов кластеризации с применением DBSCAN-алгоритма значение индекса кластерного силуэта *Silh* равно 0.11.

Опять получается, что алгоритм *k*-средних позволил получить результаты кластеризации большее высокого качества. Проверим полученные выводы, выполнив визуализацию результатов кластеризации в двухмерном пространстве (рис. 53). Так ли это?

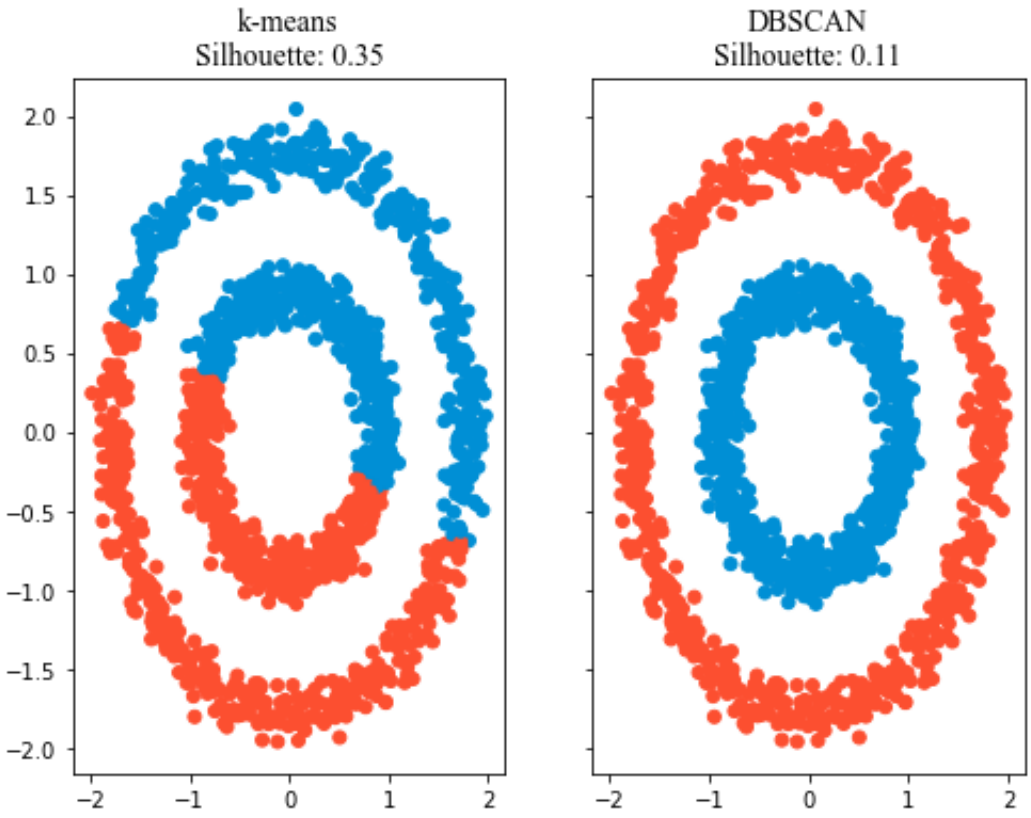


Рисунок 53. Визуализация результатов кластеризации набора данных `make_circles`

Анализ результатов кластеризации опять позволяет отвергнуть сформулированное предположение о том, что алгоритм *k*-средних позволил получить результаты кластеризации большее высокого качества! В рассматриваемом примере большее значение индекса кластерного силуэта *Silh*, равное 0.35, для результатов кластеризации с применением алгоритма *k*-средних приводит к принятию ошибочного решения. Оказывается, что именно DBSCAN-алгоритм находит более естественные кластеры в рассматриваемом примере, поскольку этот алгоритм учитывает плотность расположения объектов в пространстве признаков и не предъявляет к форме кластеров жёстких требований.

```
ari_kmeans = adjusted_rand_score(true_labels, kmeans.labels_)
ari_dbscan = adjusted_rand_score(true_labels, dbscan.labels_)
```

```
round(ari_kmeans, 2)
```

```
-0.0
```

```
round(ari_dbscan, 2)
```

```
1.0
```

Рисунок 54. Фрагмент программного кода с результатами вычисления индекса ARI для результатов кластеризации «игрушечного» набора данных `make_circles` *с применением алгоритма k-средних и DBSCAN-алгоритма*

Решение задачи визуализации результатов кластеризации «игрушечного» набора данных `make_circles` с применением алгоритма *k*-средних и DBSCAN-алгоритма было получено с применением того же программного кода, что и для набора данных `make_moons` (рис. 50).

Полученные по результатам визуализации результатов кластеризации выводы, свидетельствуют о необходимости применения более эффективного метода сравнения качества кластеризации с применением алгоритма *k*-средних и DBSCAN-алгоритма.

Так как в рассматриваемом примере метки объектов известны (`true_labels` в программном коде на рисунке 52), то опять можно использовать скорректированный индекс Рэнда (Adjusted Rand Index, *ARI*).

На рис. 54 приведен фрагмент программного кода с результатами вычисления индекса *ARI* для результатов кластеризации «игрушечного» набора данных `make_circles` с применением алгоритма *k*-средних и DBSCAN-алгоритма.

Анализ значений индекса *ARI* для результатов кластеризации «игрушечного» набора данных `make_circles` с применением алгоритма *k*-средних и DBSCAN-алгоритма позволяет сделать выводы о том, что DBSCAN-алгоритм обеспечивает лучшие (идеальные) результаты кластеризации «игрушечного» набора данных `make_circles` (по сравнению с результатами с применением алгоритма *k*-средних, которые можно считать случайными), т.к. для алгоритма *k*-средних и DBSCAN-алгоритма значение индекса *ARI* равно соответственно ~0.0 и 1.0.

Таким образом, по результатам экспериментов можно сделать следующие выводы: выбор алгоритма кластеризации и показателя качества кластеризации должен выполняться с учетом специфики объектов данных в анализируемом наборе данных.

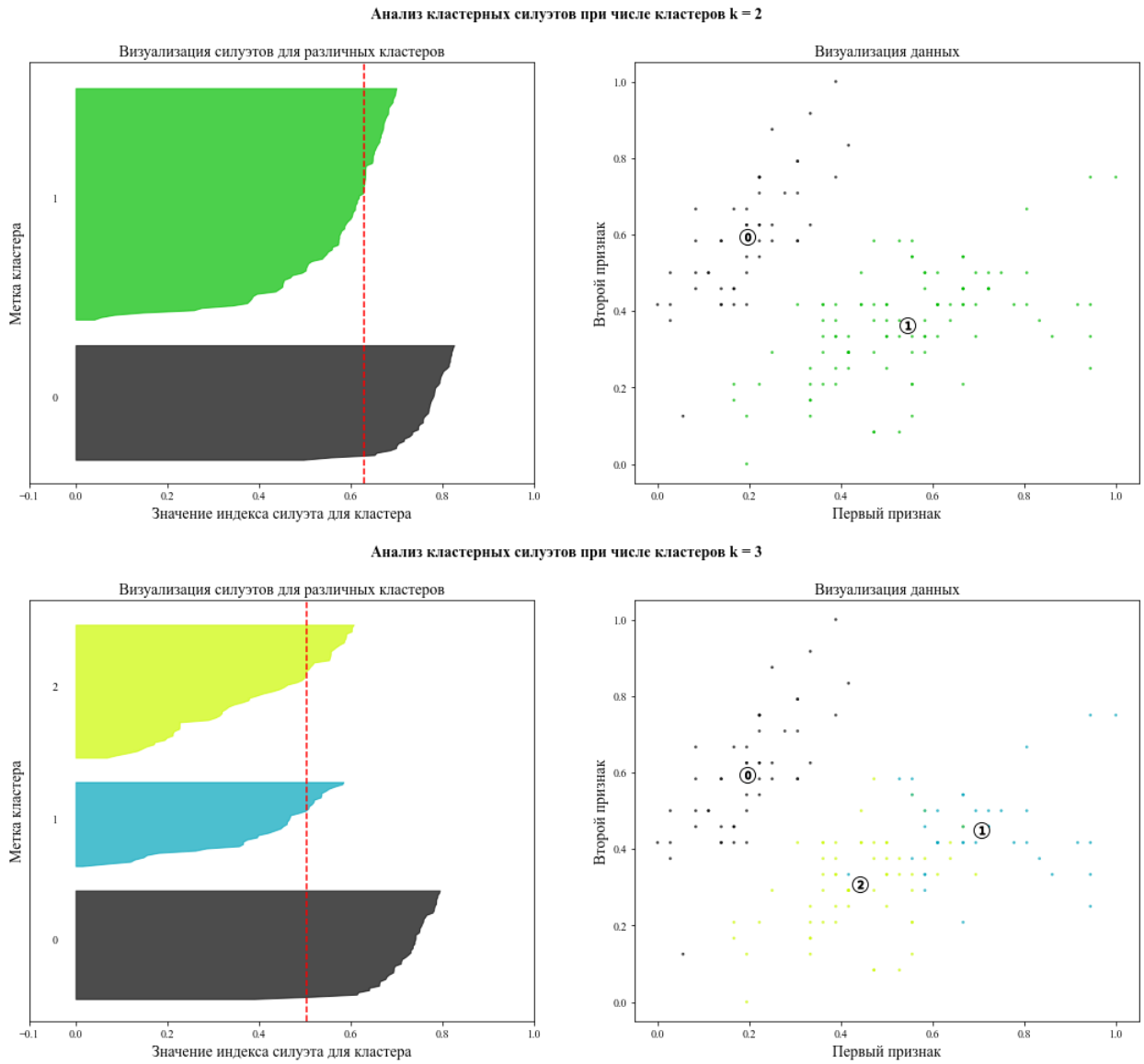


Рисунок 55. Результаты визуализации кластерного силуэта Silh при числе кластеров, соответственно равном 2 и 3, для k-means-алгоритма (Silh = 0.630 для числа кластеров k = 2; Silh = 0.505 для числа кластеров k = 3)

При использовании индекса кластерного силуэта *Silh* бывает полезно выполнить визуализацию силуэта, чтобы оценить, насколько близко каждый объект в одном кластере находится к объектам в соседних кластерах, и, таким образом, визуально оценить число кластеров. Следует отметить, что значения индекс кластерного силуэта *Silh* принадлежат диапазону [– 1, 1].

На рис. 55 и 56 представлены результаты визуализации кластерного силуэта *Silh* при числе кластеров, соответственно равном 2, 3 и 4, 5, для *k*-means-алгоритма. Вертикальная красная линия на рис. 55 и 56 отображает среднее значение силуэтов по всем кластерам, т.е. значение индекса кластерного силуэта *Silh.*

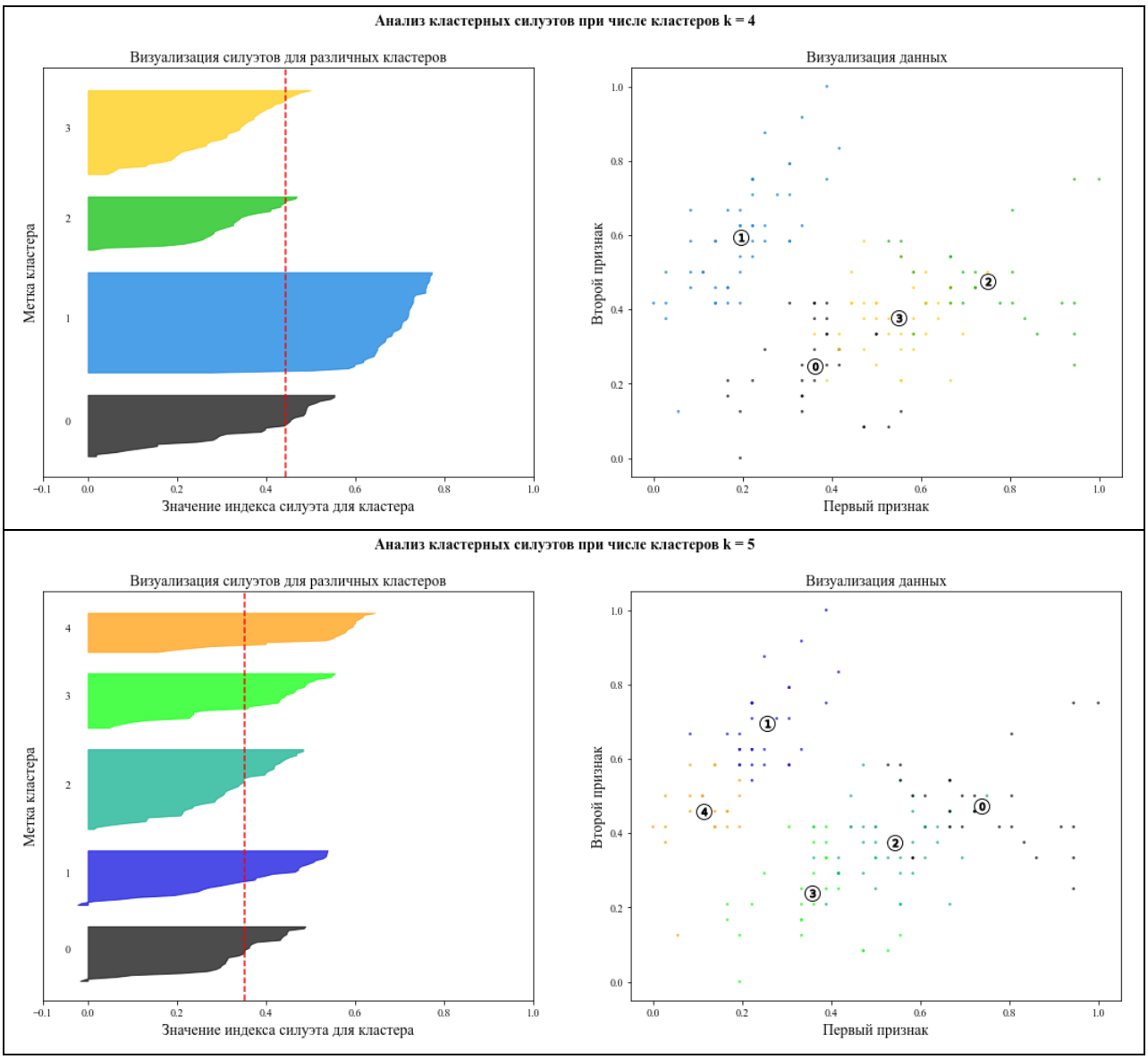


Рисунок 56. Результаты визуализации кластерного силуэта Silh при числе кластеров, соответственно равном 4 и 5, для k-means-алгоритма (Silh = 0.444 для числа кластеров k = 4; Silh = 0.353 для числа кластеров k = 5)

Программный код, который можно использовать для визуализации, доступен по ссылке [24]:

https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html#sphx-glr-auto-examples-cluster-plot-kmeans-silhouette-analysis-py.

Значение коэффициента силуэта объекта, близкое к 1, указывает на то, что объект находится далеко от соседних кластеров. Значение коэффициента силуэта объекта, близкое к 0, указывает на то, что объект находится на границе между двумя соседними кластерами или очень близко к ней, а отрицательное значение коэффициента силуэта объекта указывает на то, что объект мог быть отнесен к неправильному кластеру.

В рассматриваемом примере можно сделать следующие выводы. При числе кластеров, равном 5, имеются отрицательные значения коэффициента силуэта для некоторых объектов, а значение *Silh* – небольшое (*Silh* = 0.353). Поэтому число кластеров, равное 5, не следует использовать для кластеризации анализируемого набора данных. При числе кластеров, равном 4, один из кластеров существенно больше по объему, чем 3 другие, в которых большинство объектов имеют значения коэффициента силуэта существенно меньшие, чем среднее значение по набору данных – *Silh* (*Silh* = 0.444).

При числе кластеров, равном 2, достигается наибольшее значение индекса кластерного силуэта *Silh* (*Silh* = 0.630), однако один из кластеров примерно в 2 раза больше по объему, чем другой.

При числе кластеров, равном 3, достигается значение индекса кластерного силуэта *Silh* (*Silh* = 0.505), при этом кластеры уже более близки друг к другу по объему.

Как известно, по факту в анализируемом наборе данных скрыты 3 кластера.

Ранее уже отмечалось, что при выборе оптимального числа кластеров не следует опираться на значения только одного какого-либо показателя качества кластеризации.

Следует отметить, что не все программные реализации алгоритмов кластерного анализа обеспечивают вывод значений общей суммы квадратов расстояний от объектов до центроида ближайшего к ним кластера (внутрикластерных расстояний) – *WCSS* (Within Clusters Sum of Squares), поэтому применение метода локтя *elbow* сопряжено с выполнением дополнительных вычислений. В рассмотренных программных реализациях только для *k*-means-алгоритма реализовано вычисление значения выходного параметра `inertia_`, который может быть использован непосредственно в методе локтя *elbow*. Кроме того, многие алгоритмы (например, DBSCAN-алгоритм) не предполагают вычисление центроидов кластеров. В связи с этим наибольшее применение в последнее время находит индекс кластерного силуэта *Silh*.

4.2. Общие выводы по результатам экспериментов с набором данных «Ирисы Фишера» (The Iris Dataset)

Для набора данных «Ирисы Фишера» (The Iris Dataset) с применением 4 алгоритмов кластеризации были получены разбиения объектов на кластеры. Целесообразно сравнить полученные результаты.

Предварительно приведем результаты вычисления значений индекса кластерного силуэта при разном числе кластеров при реализации иерархического кластерного анализа.

```
Silh = []
for i in range(2, 11):
    df['cluster_labels']=fcluster(distance_matrix,
                                  i, criterion='maxclust')
    score = silhouette_score(scaled_data, df['cluster_labels'],
                             metric='euclidean')
    Silh.append(score)
```

Рисунок 57. Фрагмент программного кода, в котором реализован расчет индекса кластерного силуэта Silh при разном числе кластеров с целью выбора оптимального числа кластеров при реализации иерархического кластерного анализа

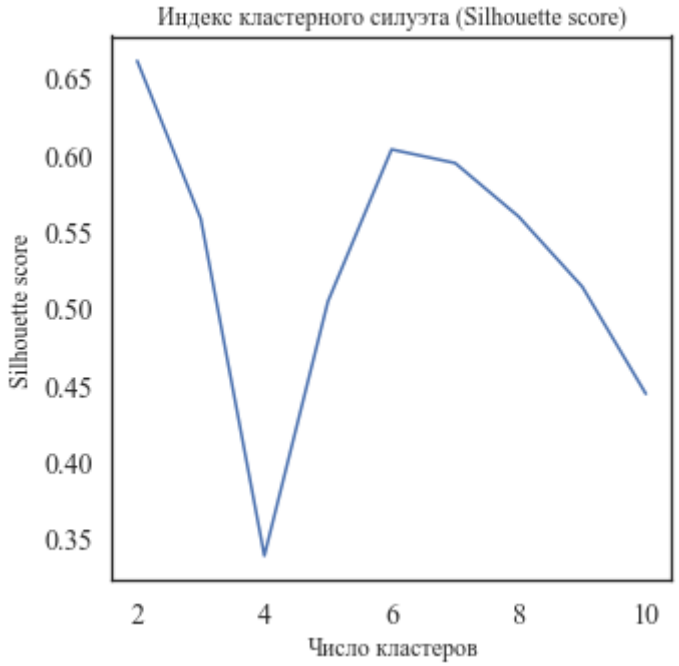


Рисунок 58. Графическая зависимость, построенная для значений индекса кластерного силуэта Silh, вычисленных при разном числе кластеров

```
silh
```

```
[0.6609240927892531,
 0.5581062605014832,
 0.33860809462550684,
 0.5041973608867151,
 0.6032064241374829,
 0.5943362929816636,
 0.5594382733236937,
 0.5137654557832417,
 0.4439884058683494]
```

Рисунок 59. Скриншот, отражающий вывод значений индекса кластерного силуэта Silh, вычисленных при разном числе кластеров

На рис. 57 приведен фрагмент программного кода, в котором реализован расчет индекса кластерного силуэта *Silh* при разном числе кластеров с целью выбора оптимального числа кластеров при реализации иерархического кла-

стерного анализа. При этом набор данных был предварительно подвергнут масштабированию с применением `MinMaxScaler` (п. 2.3), а при вычислении расстояний между кластерами использовался метод Уорда (`'ward'`).

Поиск оптимального числа кластеров и максимального значения индекса кластерного силуэта *Silh* может быть реализован за пределами цикла с помощью команд `Silh.index(max(Silh))+2` и `max(Silh)` соответственно.

На рис. 58 приведена графическая зависимость, построенная для значений индекса кластерного силуэта *Silh*, вычисленных при разном числе кластеров.

На рис. 59 приведен скриншот, отражающий вывод значений индекса кластерного силуэта *Silh* вычисленных при разном числе кластеров.

Анализ значений индекса кластерного силуэта *Silh*, вычисленных при разном числе кластеров, и графической зависимости, построенной на их основе, позволяет сделать следующие выводы.

При использовании индекса кластерного силуэта *Silh*, который должен быть максимизирован, в качестве оптимального числа кластеров следует выбрать число, равное 2, при котором значение индекса кластерного силуэта *Silh* равно 0.661 (рисунок 59).

Сравнивая результаты решения задачи кластеризации набора данных «Ирисы Фишера» (The Iris Dataset) с применением 4 алгоритмов кластеризации, можно сделать следующие выводы.

Во всех четырех экспериментах с точки зрения индекса кластерного силуэта *Silh* оптимальным является число кластеров, равное 2, в то время как реальное число кластеров, скрытых в наборе данных, равно 3. При этом максимальное значение индекса кластерного силуэта было получено при реализации иерархического кластерного анализа с использованием метода Уорда для вычисления расстояний между кластерами.

В заключение необходимо еще раз отметить, что алгоритмы иерархической кластеризации обычно применяют при небольшом числе объектов (такие алгоритмы удобно использовать для визуализации результатов кластеризации в виде дендрограмм), итерационные алгоритмы кластеризации обычно используют при большом числе объектов.

ВАРИАНТЫ И ЗАДАНИЯ

Необходимо выполнить задания по кластеризации предлагаемых наборов данных в соответствии с предложенным вариантом.

Выполнить кластеризацию наборов данных, приведенных в табл. 2 и 3, считая, что метки кластеров неизвестны, используя программные реализации иерархических алгоритмов кластеризации (п. 2.2) и итерационных алгоритмов кластеризации (п. 3.5, 3.6, 3.7).

Определить оптимальное число кластеров, рассмотрев варианты кластеризации при числе кластеров от 2 до 10.

Оценить качество кластеризации, используя различные показатели качества кластеризации в случае, когда *неизвестны* истинные метки классов (кластеров объектов), в том числе, индекс кластерного силуэта *Silh* и показатель на основе метода локтя *elbow*.

Оценить качество кластеризации, используя различные показатели качества кластеризации в случае, когда *известны* истинные метки классов (кластеров объектов), в том числе, скорректированный индекс Рэнда *ARI*.

Выбрать лучший алгоритм кластеризации для своего набора данных.

Выполнить визуализацию результатов кластеризации с помощью алгоритмов t-sne и UMAP, являющихся алгоритмами нелинейного снижения размерности, при различных сочетаниях значений их параметров: изобразить объекты разных кластеров маркерами разного цвета. Отметить центроиды кластеров, если алгоритм кластеризации их вычисляет.

Сделать выводы о принципах группирования объектов в кластеры и выполнить профилирование кластеров.

Информацию по работе с языком Python можно получить в [25, 26].

Варианты

1. Наборы данных с *известными* метками классов (кластеров) объектов.

Таблица 2. Варианты наборов данных

Вариант	Источник набора данных
1	http://archive.ics.uci.edu/ml/datasets/Post-Operative+Patient
2	http://archive.ics.uci.edu/ml/datasets/Primary+Tumor
3	http://archive.ics.uci.edu/ml/datasets/Soybean+%28Large%29
4	http://archive.ics.uci.edu/ml/datasets/Low+Resolution+Spectrometer
5	http://archive.ics.uci.edu/ml/datasets/Ionosphere
6	http://archive.ics.uci.edu/ml/datasets/Horse+Colic
7	http://archive.ics.uci.edu/ml/datasets/Hepatitis
8	http://archive.ics.uci.edu/ml/datasets/Heart+Disease
9	http://archive.ics.uci.edu/ml/datasets/Hayes-Roth
10	http://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival
11	http://archive.ics.uci.edu/ml/datasets/Glass+Identification
12	http://archive.ics.uci.edu/ml/datasets/Flags
13	http://archive.ics.uci.edu/ml/datasets/Ecoli
14	http://archive.ics.uci.edu/ml/datasets/Echocardiogram
15	http://archive.ics.uci.edu/ml/datasets/Dermatology
16	http://archive.ics.uci.edu/ml/datasets/Credit+Approval
17	http://archive.ics.uci.edu/ml/datasets/Pittsburgh+Bridges
18	http://archive.ics.uci.edu/ml/datasets/Spambase
19	http://archive.ics.uci.edu/ml/datasets/University
20	http://archive.ics.uci.edu/ml/datasets/statlog+(australian+credit+approval)

2. Наборы данных с *неизвестными* метками классов (кластеров) объектов.

Таблица 3. Варианты наборов данных

Вариант	Источник набора данных
1	https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities
2	https://archive.ics.uci.edu/ml/datasets/Breath+Metabolomics
3	https://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaIoT
4	https://archive.ics.uci.edu/ml/datasets/DrivFace
5	https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014
6	https://archive.ics.uci.edu/ml/datasets/Epileptic+Seizure+Recognition
7	https://archive.ics.uci.edu/ml/datasets/FMA%3A+A+Dataset+For+Music+Analysis
8	https://archive.ics.uci.edu/ml/datasets/Gas+Sensor+Array+Drift+Dataset+at+Different+Concentrations
9	https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq
10	https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones
11	https://archive.ics.uci.edu/ml/datasets/Kitsune+Network+Attack+Dataset
12	https://archive.ics.uci.edu/ml/datasets/MEx
13	https://archive.ics.uci.edu/ml/datasets/Mturk+User-Perceived+Clusters+over+Images
14	https://archive.ics.uci.edu/ml/datasets/Repeat+Consumption+Matrices
15	https://archive.ics.uci.edu/ml/datasets/Reuter_50_50
16	https://www.kaggle.com/arjunbhasin2013/ccdata
17	https://www.kaggle.com/mrisdal/open-exoplanet-catalogue
18	https://www.kaggle.com/imdevskp/corona-virus-report
19	https://www.kaggle.com/roshansharma/mall-customers-clustering-analysis
20	https://www.kaggle.com/karnikakapoor/customer-segmentation-clustering

СПИСОК ЛИТЕРАТУРЫ

1. Паклин Н., Орешков В. Бизнес-аналитика: от данных к знаниям: Учебное пособие. 2-е изд., испр. СПб.: Питер. 2013. 704 с.
2. Clustering in Machine Learning [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/clustering-in-machine-learning/?ref=lbp, свободный (дата обращения 12.02.2022).
3. Демидова Л.А., Степанов М.А. Подход к решению задачи выявления структурных трансформаций в группах временных рядов // Cloud of Science. 2019. Т. 6. № 2. С. 201 – 226.
4. SciPy – Cluster Hierarchy Dendrogram [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/scipy-cluster-hierarchy-dendrogram/?ref=gcse, свободный (дата обращения 12.02.2022).
5. 6.3. Preprocessing data [Электронный ресурс]. – Режим доступа: https://scikit-learn.org/stable/modules/preprocessing.html, свободный (дата обращения 12.02.2022).
6. ГЛАВА 10. Кластерный анализ. 10.1 Алгоритмы кластеризации, основанные на разделении [Электронный ресурс]. – Режим доступа: https://ranalytics.github.io/data-mining/101-Partitioning-Algos.html, свободный (дата обращения 12.02.2022).
7. MacQueen J. Some methods for classification and analysis of multivariate observations // Berkeley Symposium on Mathematical Statistics and Probability, 1967. – P. 281 – 297.
8. K means Clustering – Introduction [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/k-means-clustering-introduction/?ref=lbp, свободный (дата обращения 12.02.2022).
9. Bezdek J.C., Ehrlich R., Full W. FCM: The fuzzy c-means clustering algorithm // Computers & Geosciences. 1984. Vol. 10 (2–3). P. 191 – 203.
10. Ester M., Kriegel H.-P., Sander J., Xu X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise // KDD'96: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. August 1996. P. 226–231.
11. DBSCAN Clustering in ML | Density based clustering [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/, свободный (дата обращения 12.02.2022).
12. Cluster Validaty and Cluster Number Selection [Электронный ресурс]. – Режим доступа: https://chih-ling-hsu.github.io/2018/05/28/cluster-number, свободный (дата обращения 12.02.2022).

13. Оценка качества в задаче кластеризации [Электронный ресурс]. – Режим доступа: http://neerc.ifmo.ru/wiki/index.php?title=%D0%9E%D1%86%D0%B5%D0%BD%D0%BA%D0%B0_%D0%BA%D0%B0%D1%87%D0%B5%D1%81%D1%82%D0%B2%D0%B0_%D0%B2_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B5_%D0%BA%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8, свободный (дата обращения 12.02.2022).

14. Knee/Elbow Point Detection [Электронный ресурс]. – Режим доступа: https://www.kaggle.com/kevinarvai/knee-elbow-point-detection, свободный (дата обращения 12.02.2022).

15. scipy.cluster.vq.kmeans [Электронный ресурс]. – Режим доступа: https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.vq.kmeans.html#scipy.cluster.vq.kmeans, свободный (дата обращения 12.02.2022).

16. sklearn.cluster.KMeans [Электронный ресурс]. – Режим доступа: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html, свободный (дата обращения 12.02.2022).

17. Fuzzy c-means clustering [Электронный ресурс]. – Режим доступа: https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_cmeans.html, свободный (дата обращения 12.02.2022).

18. fuzzy-c-means 1.6.3 [Электронный ресурс]. – Режим доступа: https://pypi.org/project/fuzzy-c-means/, свободный (дата обращения 12.02.2022).

19. sklearn.cluster.DBSCAN [Электронный ресурс]. – Режим доступа: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html, свободный (дата обращения 12.02.2022).

20. Demo of DBSCAN clustering algorithm [Электронный ресурс]. – Режим доступа: https://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html#sphx-glr-auto-examples-cluster-plot-dbscan-py, свободный (дата обращения 12.02.2022).

21. API Reference [Электронный ресурс]. – Режим доступа: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics, свободный (дата обращения 12.02.2022).

22. K-means, DBSCAN, GMM, Agglomerative clustering — Mastering the popular models in a segmentation problem [Электронный ресурс]. – Режим доступа: https://towardsdatascience.com/k-means-dbscan-gmm-agglomerative-clustering-mastering-the-popular-models-in-a-segmentation-c891a3818e29, свободный (дата обращения 12.02.2022).

23. Comparing different clustering algorithms on toy datasets [Электронный ресурс]. – Режим доступа: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html#sphx-glr-auto-examples-cluster-plot-kmeans-silhouette-analysis-py, свободный (дата обращения 12.02.2022).

24. Selecting the number of clusters with silhouette analysis on KMeans clustering [Электронный ресурс]. – Режим доступа: https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html#sphx-glr-auto-examples-cluster-plot-cluster-comparison-py, свободный (дата обращения 12.02.2022).

25. Лутц М. Изучаем Python. М.: Диалектика. 2020. Том. 1. 832 с.; Том. 2. 720 с.

26. Доусон М. Программируем на Python. СПб.: Питер. 2020. 416 с.

Сведения об авторах

Демидова Лилия Анатольевна, доктор технических наук, профессор, профессор кафедры корпоративных информационных систем Института информационных технологий РТУ МИРЭА.