Основы кластерного анализа.

Кластерный анализ – это многомерная статистическая процедура и технология анализа данных, основная цель которой – разделение множества объектов на группы (кластеры) так, чтобы объекты внутри каждого кластера были максимально схожи друг с другом, а объекты из разных кластеров существенно отличались. Это позволяет выявить естественную структуру данных без предварительного знания меток или классов – по сути, кластеризация является методом обучения без учителя.

Основные этапы кластерного анализа включают: выбор объектов для кластеризации и признаков (переменных), определение меры сходства или расстояния между объектами, применение алгоритмов кластеризации для объединения схожих объектов в кластеры, а также проверку достоверности и интерпретацию полученных групп. Важным условием является однородность – все объекты должны быть одной природы и иметь сопоставимый набор признаков.

Существуют разные подходы к кластеризации, среди которых выделяют две большие группы методов: иерархические (агломеративные и дивизимные) и неиерархические (например, метод k-средних). Иерархические методы строят дерево кластеров, постепенно объединяя или разбивая группы, а неиерархические разбивают данные на заранее заданное число кластеров.

Кластерный анализ широко применяется в самых разных областях: маркетинге (сегментация клиентов), экономике, биологии, медицине, психологии, социологии и многих других. Он помогает выделять однородные группы для дальнейшего анализа, оптимизации и принятия решений, а также выявлять скрытые закономерности в сложных данных.

Методы кластеризации: k-средних

Метод k-средних (k-means) – популярный алгоритм кластерного анализа, который используется для разбиения множества объектов на заранее заданное число кластеров k. Основная цель алгоритма – сформировать кластеры так, чтобы объекты внутри одного кластера были максимально похожи друг на друга, а объекты из разных кластеров – как можно более различны. Это достигается минимизацией суммы квадратов расстояний от объектов до центров их кластеров.

Алгоритм работает итеративно и включает следующие шаги, на примере определения точек, где оптимально будет разместить шаурмичные в жилом квартале (рис.1):

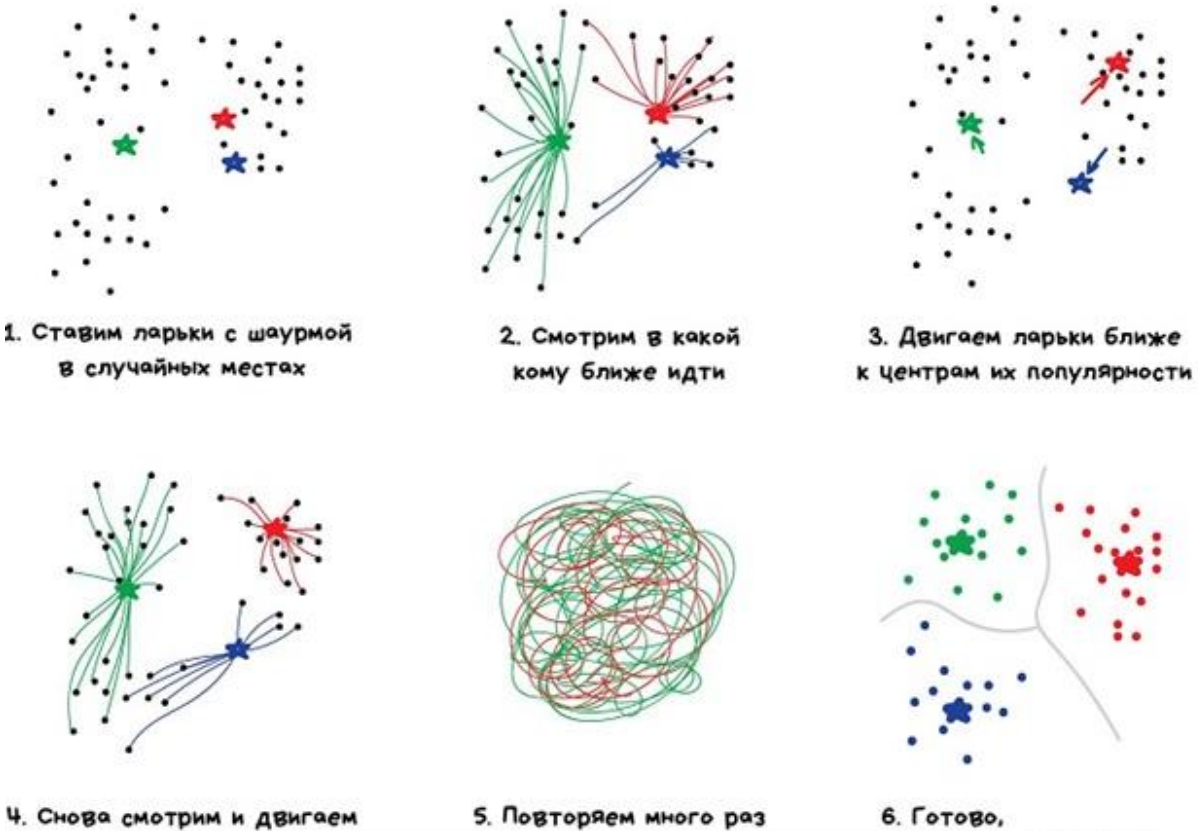

Рисунок 1 – Визуализация работы алгоритмы k-средних.

1. Изначально выбирается число кластеров k.
2. Случайным образом выбираются k начальных центров кластеров («центроидов») (рис.1 – п.1).
3. Каждому объекту назначается ближайший по выбранной метрике (обычно Евклидовой) центр кластера. (рис.1 – п.2)
4. Рассчитываются новые центроиды как средние значения признаков всех объектов, принадлежащих к каждому кластеру (рис.1 – п.3).
5. Процесс назначения объектов и пересчета центроидов повторяется до тех пор, пока центроиды не перестанут существенно изменяться, то есть не будет достигнута сходимость (рис.1 – п.5).

Метод k-средних прост для реализации, быстр по сравнению с другими методами кластеризации и хорошо работает в задачах с хорошо разделимыми кластерами. Однако требует заранее задать число кластеров, что иногда сложно. Для выбора оптимального k применяют, например, метод локтя – анализ графика внутрикластерной дисперсии при разных значениях k (рис.2).

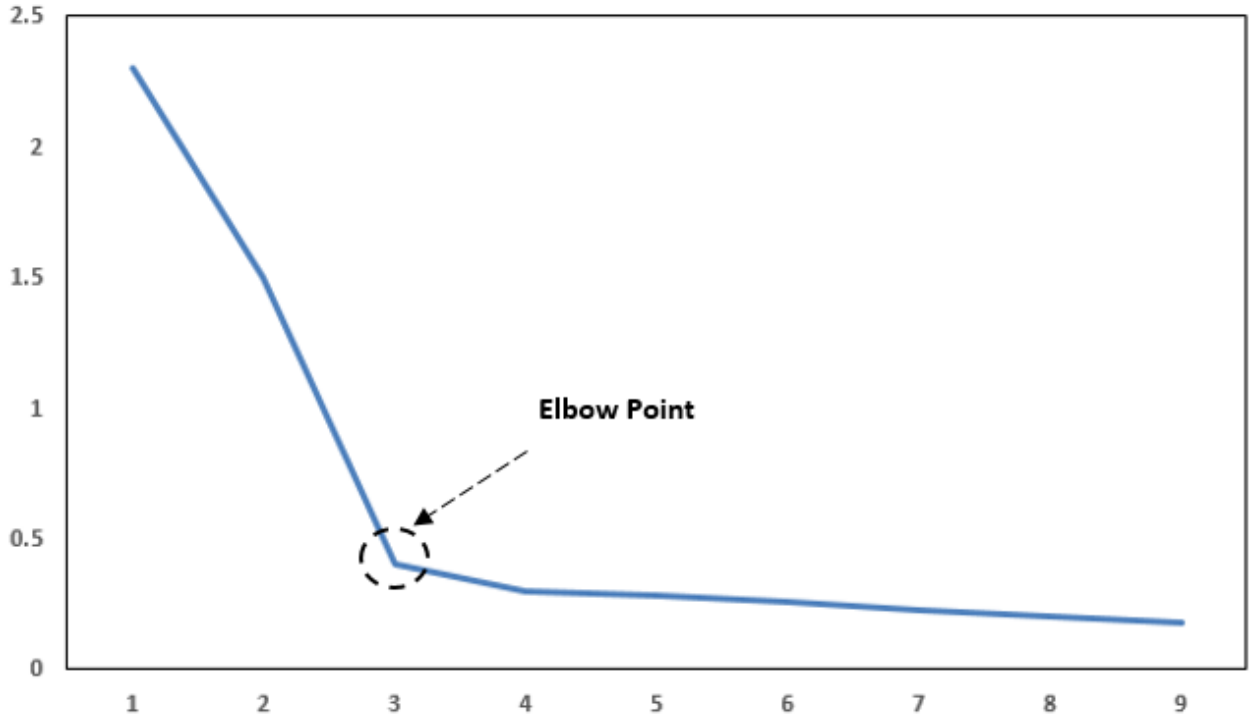

Рисунок 2 – Метод локтя

В итоге метод k-средних представляет собой эффективный инструмент неспециализированного обучения без учителя для выявления структур в данных, широко используемый в маркетинге, биологии, экономике и многих других областях, где требуется выделение однородных групп в больших наборах данных.

Пример. Реализовать метод k-means с помощью Python, использовать набор данных о строительстве жилья в Калифорнии для создания экономических сегментов в различных районах Калифорнии.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.cluster import KMeans
data = pd.read_csv("housing.csv")
print(data.columns)
# для кластеризации возьмем данные по среднему доходу семьи, широте и долготе
data = data.loc[:, ["median_income", "latitude", "longitude"]]
print(data.head())

#  реализуем алгоритм кластеризации K-средних с помощью Python
kmeans = KMeans(n_clusters=6)
data["Cluster"] = kmeans.fit_predict(data)
data["Cluster"] = data["Cluster"].astype("int")
print(data.head())

# кластеры, идентифицированные алгоритмом с помощью диаграммы рассеяния
plt.style.use("seaborn")
plt.rc("figure", autolayout=True)
```

```
plt.rc("axes", labelweight='bold', labelsize='large', titleweight='bold',
titlesize=14, titlepad=10)
sns.relplot(x="longitude", y="latitude", hue="Cluster", data=data, height=6)
plt.show()
```

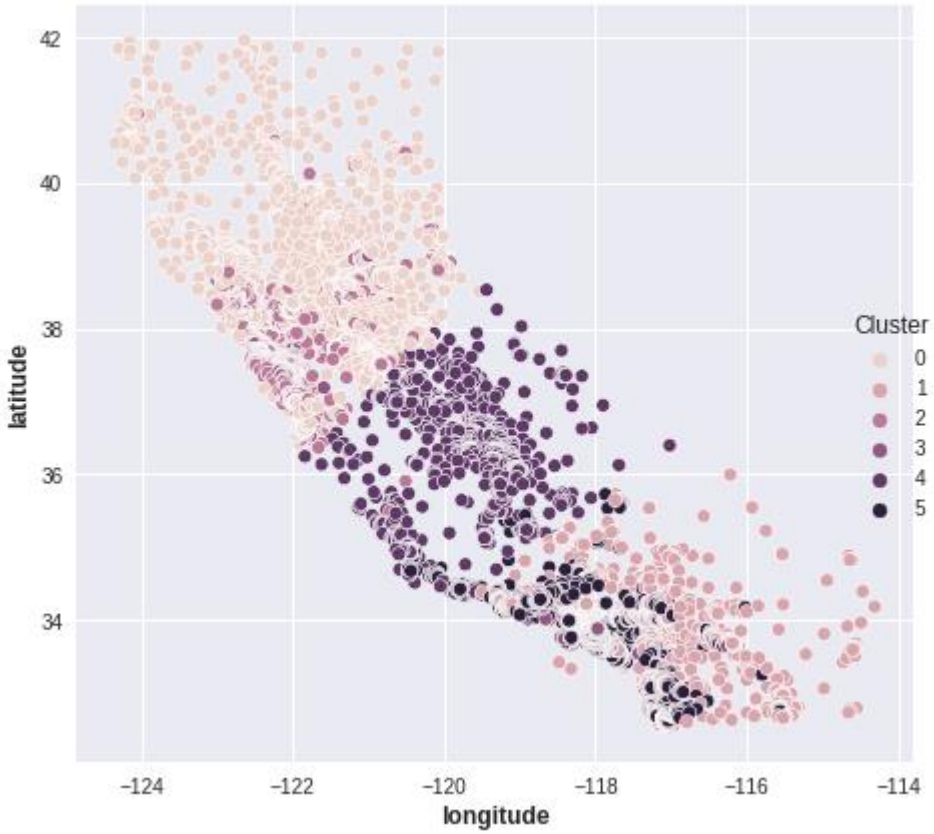

Рисунок 3 – Визуализация кластеров при помощи метода k-means

График разброса выше показывает географическое распределение кластеров. Алгоритм создал отдельные сегменты для области с высоким доходом.

Метод нечеткой кластеризации с-средних

Метод нечеткой кластеризации с-средних (Fuzzy C-Means, FCM) – это усовершенствованный вариант классического метода k-средних, который позволяет каждому объекту данных принадлежать одновременно нескольким кластерам с определённой степенью принадлежности. Вместо жёсткого распределения объектов по кластерам, FCM присваивает каждой точке набора данных вероятностные или весовые коэффициенты, отражающие, насколько объект относится к тому или иному кластеру.

Основная идея алгоритма заключается в итеративном минимизировании целевой функции, которая взвешивает расстояния между точками данных и центрами кластеров по степеням принадлежности объектов. На каждом шаге алгоритма пересчитываются центры кластеров как средневзвешенные координаты объектов, а затем обновляются степени принадлежности этих

объектов к каждому из кластеров. Процесс повторяется до сходимости – пока изменения в степенях принадлежности не станут меньше заданного порога.

Ключевые параметры метода – это число кластеров *c*, а также параметр нечёткости *m*>1>, который регулирует «размытие» границ кластеров: при увеличении *m* объекты принимают более равномерное распределение принадлежности к кластерам, при приближении m к 1 алгоритм приближается к жёсткому разбиению. Метод хорошо подходит для задач, где объекты могут одновременно частично принадлежать разным группам, например в обработке изображений, сегментации данных или анализе сложных многомерных структур.

Таким образом, нечеткая кластеризация с-средних – это гибкий способ группировки данных, когда требуется учитывать неоднозначность принадлежности объектов. В отличие от классического k-средних, она позволяет выделять пересекающиеся или размытые кластеры, что часто улучшает интерпретируемость и качество кластеризации в реальных задачах.

Основные формулы алгоритма FCM

Пусть имеется набор данных

$X=\{x_1, x_2,\ldots, x_n\}$, где n – количество объектов.

Необходимо разбить данные на с кластеров с центрами $V=\{v_1, v_2,\ldots, v_c\}$ в пространстве признаков размерности m.

Чем минимизируется целевая функция:

$$J_{FCM} = \sum_{i=}^{c}\sum_{j=1}^{n} u_{ij}^{m} d_{ij}^{2}$$

где: u_{ij} – степень принадлежности объекта x_j к кластеру i (число от 0 до 1), при этом для каждого объекта сумма принадлежностей равна 1:

$\sum_{i=1}^{c} u_{ij} = 1$,

m>1 – параметр "нечёткости" (обычно 2),

$d_{ij}=\|x_j - v_i\|$ – расстояние между объектом и центром кластера i, как правило, Евклидово.

Пример. Кластеризовать данные при помощи метода с-средних. На рисунке 4 показан результат работы кластеризатора.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

X, _ = make_blobs(n_samples=300, centers=3, cluster_std=1.5, random_state=42)

def fuzzy_c_means(X, c=3, m=2, max_iter=100, error=1e-5):
    N = len(X)
```

```
    u = np.random.rand(N, c)
    u /= u.sum(axis=1, keepdims=True)
    for _ in range(max_iter):
        um = u ** m
        centers = (um.T @ X) / um.sum(axis=0)[:, None]
        dist = np.linalg.norm(X[:, None] - centers, axis=2)
        tmp = dist[:, :, None] / dist[:, None, :]
        u_new = 1 / (tmp ** (2 / (m - 1))).sum(axis=2)
        if np.linalg.norm(u_new - u) < error:
            break
        u = u_new
    return centers, u

centers, u = fuzzy_c_means(X)
labels = np.argmax(u, axis=1)
plt.scatter(X[:,0], X[:,1], c=labels, cmap='viridis')
plt.scatter(centers[:,0], centers[:,1], c='red', marker='X')
plt.show()
```

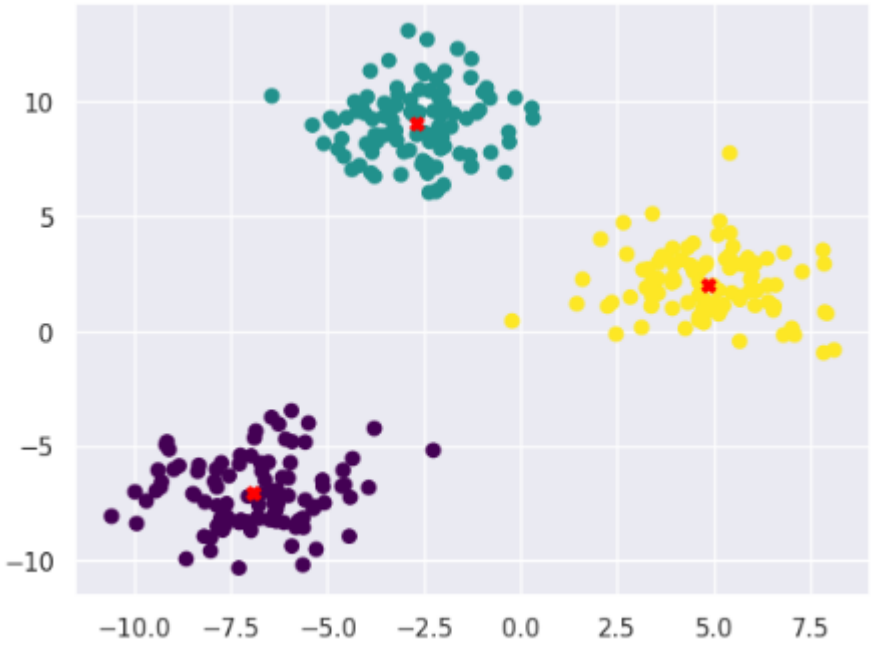

Рисунок 4 – Результат кластеризации методом с-средних

Иерархическая кластеризация

Иерархическая кластеризация – это набор алгоритмов, направленных на построение иерархии вложенных кластеров в виде древовидной структуры (дендрограммы). Суть метода в том, что все объекты изначально рассматриваются либо как отдельные кластеры (агломеративный подход), либо все объекты сразу объединены в один большой кластер (дивизивный подход), и далее происходит последовательное объединение или деление кластеров.

Агломеративная иерархическая кластеризация начинается с каждого объекта в отдельном кластере. На каждом шаге объединяются два наиболее похожих кластера, после чего расстояния между новыми кластерами

пересчитываются. Процесс повторяется до тех пор, пока все объекты не окажутся в одном кластере или не будет достигнуто желаемое число кластеров.

Дивизивная иерархическая кластеризация работает наоборот: начинается с одного большого кластера, включающего все объекты, и последовательно разделяет его на меньшие, пока не будут выделены отдельные объекты или заданное число кластеров.

Для оценки сходства между кластерами применяются различные метрики расстояния (например, Евклидово) и методы агрегации расстояний:

– расстояние ближайших соседей (single linkage),
– расстояние полной связи (complete linkage),
– среднее расстояние (average linkage) и др.

Результатом иерархической кластеризации часто служит дендрограмма – дерево, отражающее структуру вложенности кластеров и расстояния между ними, что помогает анализировать множество кластерных разбиений на разных уровнях.

Таким образом, иерархическая кластеризация позволяет выявлять многослойную структуру данных и визуально интерпретировать отношения между объектами и группами. Она широко применяется в биологии (например, для систематики), маркетинге, психологии и других областях, где важна глубокая кластеризация иерархий.

Пример. рассмотрим пример иерархической кластеризации для Wine Dataset. В этом примере мы сначала визуализируем данные с помощью точечного графика, а затем применим иерархическую кластеризацию и построим дендрограмму (рис.5-7).

Датасет Wine состоит из 178 образцов вина, описанных 13 химическими признаками. Он включает три класса вин.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.decomposition import PCA
from scipy.cluster.hierarchy import dendrogram, linkage
# Загрузка датасета Wine
wine = load_wine()
X = wine.data
y = wine.target
import pandas as pd
df = pd.DataFrame(data=wine.data, columns=wine.feature_names)
df.head()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0

Рисунок 5 – Исходные данные для примера

```
# Применение PCA для уменьшения размерности до 2D для визуализации
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

# Построение точечного графика
plt.figure(figsize=(8, 5))
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y, cmap='viridis',
marker='o', edgecolor='k')
plt.title('Точечный график Wine Dataset (PCA)')
plt.xlabel('Первая главная компонента')
plt.ylabel('Вторая главная компонента')
plt.colorbar(label='Класс вина')
plt.show()
```

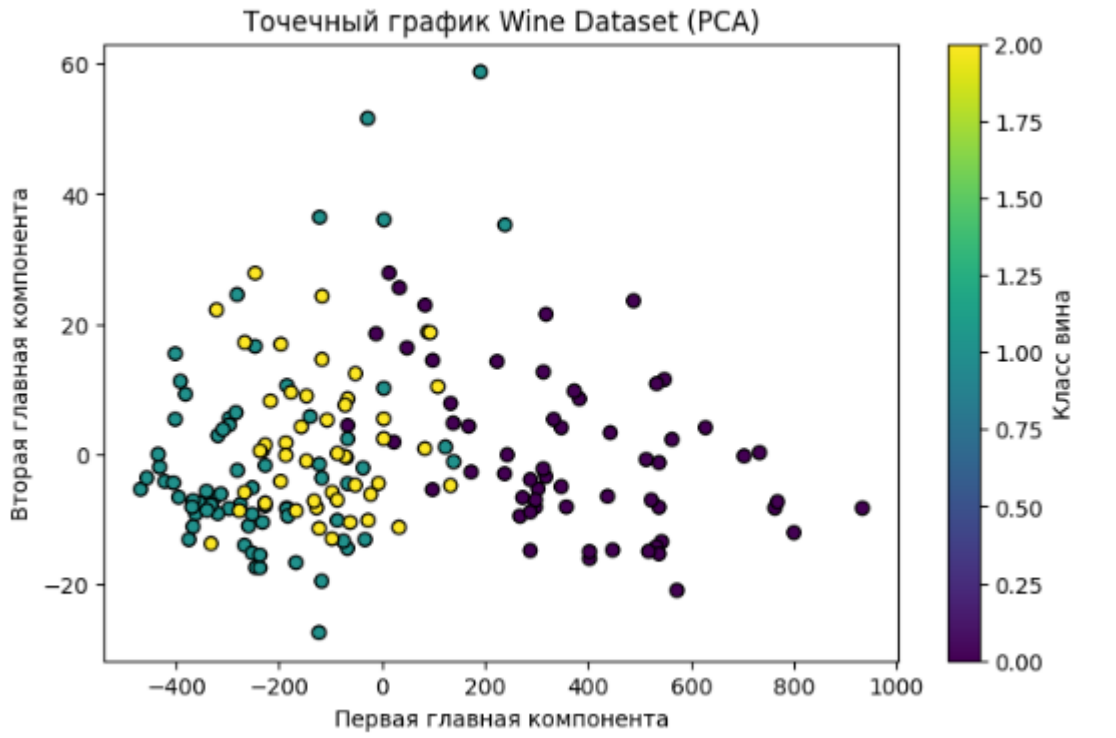

Рисунок 6 – Визуализация данных после понижения размерности

```
# Выполнение иерархической кластеризации
Z = linkage(X, method='ward')
# Построение дендрограммы
plt.figure(figsize=(12, 8))
dendrogram(Z, labels=wine.target_names[y])
plt.title('Дендрограмма для Wine Dataset')
plt.xlabel('Объекты')
plt.ylabel('Расстояние')
plt.show()
```

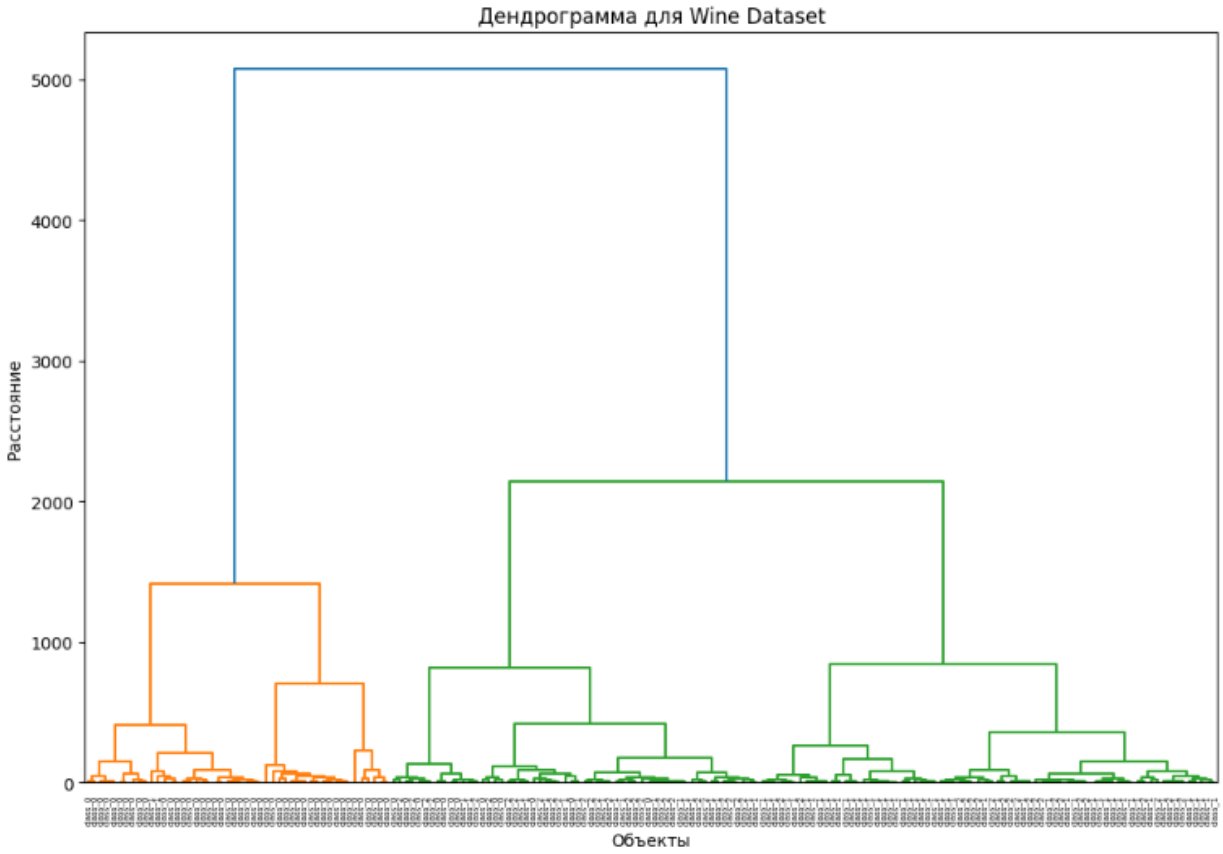

Рисунок 7 – Дендрограмма

По количеству длинных хвостов можно сделать предварительный вывод о количестве кластеров в наборе данных

DBSCAN

DBSCAN – это алгоритм кластеризации по плотности, который выделяет кластеры как области с высокой плотностью точек и отделяет их от шумов. Он использует два основных параметра: радиус ε, определяющий соседство точки, и минимальное число точек MinPts в этом радиусе, чтобы точка считалась ядром кластера (рис.8).

Точки с количеством соседей не меньше MinPts считаются ядрами, соседние с ядром точки – граничными, остальные – шумом. Этот метод не требует заранее задавать число кластеров, способен находить кластеры произвольной формы и автоматически выделяет выбросы.

DBSCAN хорошо подходит для данных с естественными плотными группами и шумовыми выбросами. В отличие от иерархической кластеризации, DBSCAN устойчив к шуму и не строит иерархию, а сразу формирует конечное разбиение.

Для успешного применения важно правильно подобрать параметры ε и MinPts, что может потребовать опытной настройки или анализа структуры данных.

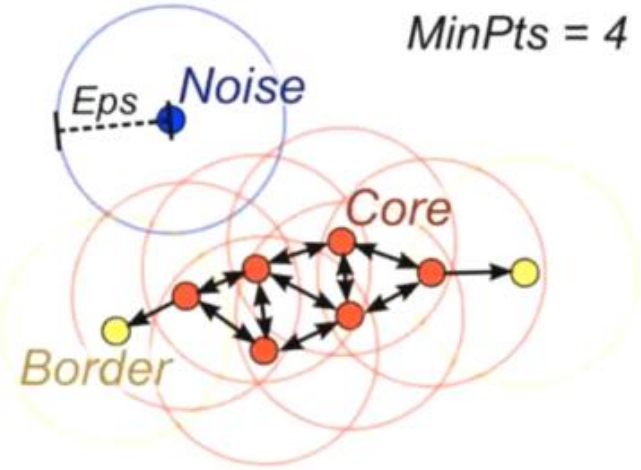

Рисунок 8 - Алгоритм DBSCAN

Пример. Вернемся к примеру для иерархической кластеризации и посмотрим как с этим набором данных справится DBSCAN (рис.9).

```
from sklearn.cluster import DBSCAN

# Применение DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)#3, 2
clusters = dbscan.fit_predict(X)

# Печать количества кластеров и шумов
n_clusters = len(set(clusters)) - (1 if -1 in clusters else 0)
n_noise = list(clusters).count(-1)
print(f'Количество кластеров: {n_clusters}')
print(f'Количество шумов: {n_noise}')

# Визуализация кластеров
plt.figure(figsize=(8, 6))
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=clusters, cmap='viridis',
marker='o', edgecolor='k')
plt.title('Кластеры Wine Dataset с использованием DBSCAN (PCA)')
plt.xlabel('Первая главная компонента')
plt.ylabel('Вторая главная компонента')
plt.colorbar(label='Кластер')
plt.show()
```

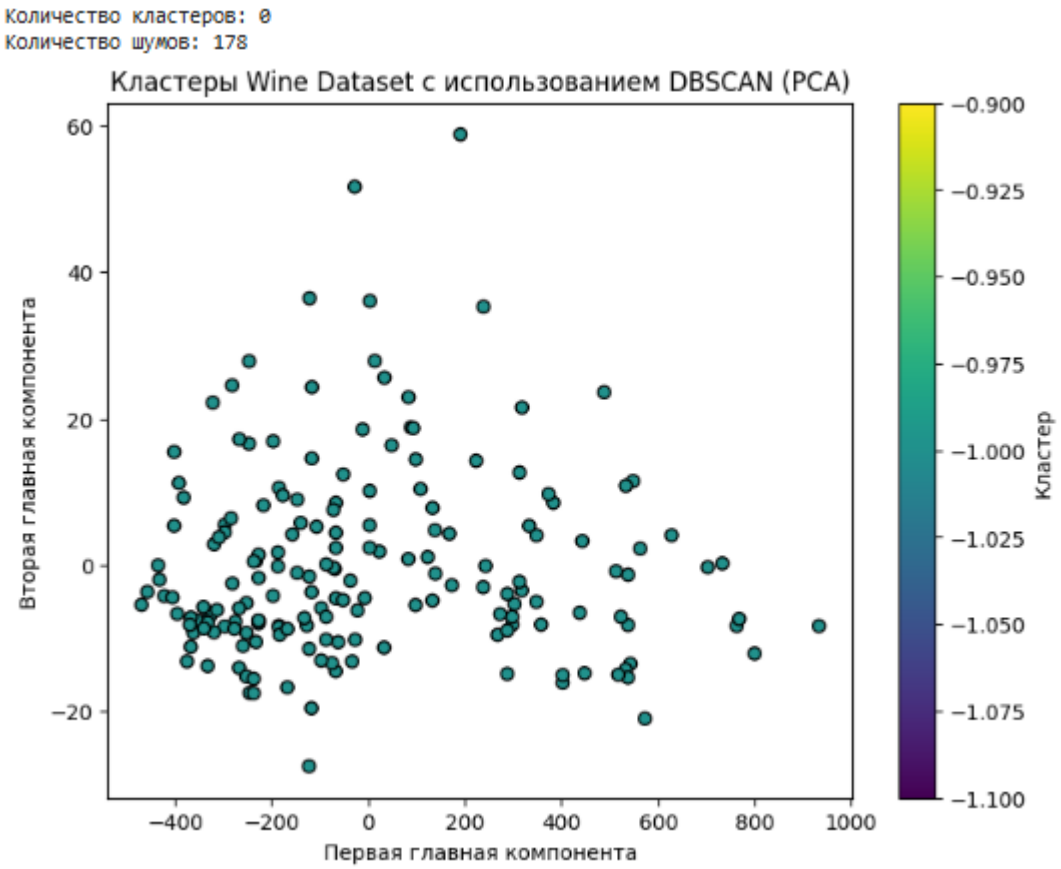

Рисунок 9 – Результат работы кластеризатора DBSCAN

Как видно, DBSCAN не подходит для такого набора данных, где точки равномерно разбросаны друг от друга. Как говорилось ранее, DBSCAN подходит для данных с естественными плотными группами. DBSCAN хорошо справляется с данными, содержащими шум или выбросы. Он классифицирует точки, которые не принадлежат ни к одному кластеру, как "шум" (метка -1), что позволяет игнорировать аномальные значения.

DBSCAN может быть эффективен для кластеризации данных, где кластеры имеют высокую плотность, а между ними есть области с низкой плотностью. Это делает его идеальным для:

– данных о местоположении: данные о местоположении пользователей, где плотные группы пользователей могут представлять популярные места.

– данных о событиях: данные о сейсмических событиях, где кластеры могут представлять зоны повышенной активности.

Так для известного набора данных moons DBSCAN подходит идеально (рис.10).

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
```

```
from sklearn.cluster import DBSCAN

# Генерация синтетических данных
X, y = make_moons(n_samples=300, noise=0.1, random_state=42)

# Применение DBSCAN
dbscan = DBSCAN(eps=0.2 , min_samples=5)
clusters = dbscan.fit_predict(X)

# Визуализация результатов
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=clusters, cmap='viridis', marker='o',
edgecolor='k')
plt.title('Кластеры, обнаруженные с помощью DBSCAN (make_moons)')
plt.xlabel('Признак 1')
plt.ylabel('Признак 2')
plt.colorbar(label='Кластер')
plt.show()
```

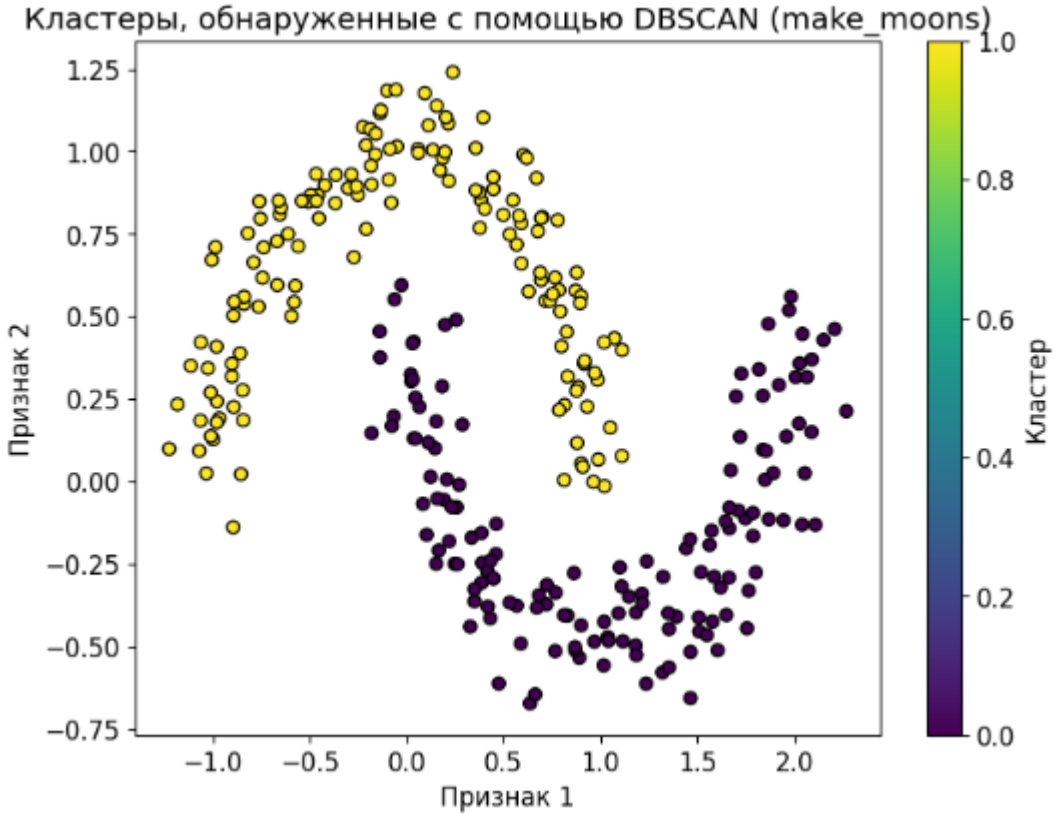

Рисунок 10 – Результат работа кластеризатора DBSCAN для датасета moons

Постановка задачи кластеризации и выбор числа кластеров.

Постановка задачи кластеризации состоит в разбиении конечной обучающей выборки объектов на подмножества (кластеры), так чтобы внутри каждого кластера объекты были максимально похожими друг на друга по выбранной метрике (функции расстояния), а объекты из разных кластеров – существенно отличались. Это задача обучения без учителя, так как обычно заранее не известны метки объектов и даже число кластеров. Алгоритм

кластеризации сопоставляет каждому объекту идентификатор кластера, формируя тем самым группировку по сходству характеристик.

Основной сложностью постановки является выбор числа кластеров k, которое является гиперпараметром большинства алгоритмов, например, метода k-средних. Число кластеров выбирается либо эмпирически, либо с помощью специальных критериев качества кластеризации, таких как внутрикластерное расстояние (средняя дисперсия внутри кластера), силуэт, индекс Дэвиса-Боулдина и др. От правильного выбора k зависит качество и интерпретируемость результатов кластеризации.

Для корректной постановки задачи важно определить признаки (характеристики объектов), по которым будет оцениваться их схожесть, а также выбрать подходящую метрику расстояния, например,

Евклидово расстояние

$$\rho(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Манхэттенское расстояние

$$\rho(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

.

Предварительно данные часто нормализуют, чтобы разные признаки были сопоставимы по масштабу. Кроме того, важными этапами являются подготовка и очистка данных, поскольку качество вводных данных напрямую влияет на результаты кластеризации.

Итого, задача кластеризации формулируется так: разбить множество объектов на k кластеров, минимизируя внутрикластерные расстояния и максимизируя межкластерные, при этом определить k можно разными способами исходя из задачи и структуры данных. Результаты кластеризации требуют экспертной интерпретации и проверки качества, поскольку идеального единственного решения не существует из-за субъективности выбора параметров и метода.

Оценка качества кластеризации.

Существует большое количество инструментов оценки качества кластеризации, они разделяются на внутренние (основанные только на свойствах выборки и кластеров) и внешние (использующие данные об истинном распределении объектов по кластерам, если оно известно).

Внутренние метрики

Внутрикластерное расстояние (также называется компактностью кластеров, cluster cohesion):

$$\sum_{k=1}^{K}\sum_{i=1}^{l}[a(x_i)=k]\rho(x_i,c_k)$$

где K - количество кластеров,

c_k - центр кластера.

Этот функционал нужно минимизировать, так как в идеальном случае все объекты в одном кластере одинаковы, и расстояние между ними равно нулю.

Межкластерное расстояние (отделимость кластеров, cluster separation):

$$\sum_{i,j=1}^{l}[a(x_i)\neq a(x_j)]\rho(x_i,x_j)$$

Этот функционал наоборот нужно максимизировать, так как объекты из разных кластеров должны максимально различаться, то есть иметь максимальное расстояние между собой.

Часто используются те же формулы, но включающие не расстояние ρ, а его квадрат, получая квадратичное *внутрикластерное* и *межкластерное расстояние*:

$$\sum_{k=1}^{K}\sum_{i=1}^{l}[a(x_i)=k]\rho^2(x_i,c_k),$$

$$\sum_{i,j=1}^{l}[a(x_i)\neq a(x_j)]\rho^2(x_i,x_j).$$

- *Среднее внутрикластерное расстояние* (среднее расстояние внутри каждого кластера, просуммированное по всем кластерам) и *среднее межкластерное расстояние* (минимизируется и максимизируется, соответственно, по аналогии с двумя первыми функционалами):

$$\sum_{k=1}^{K}\frac{1}{|k|}\sum_{i=1}^{l}[a(x_i)=k]\rho(x_i,c_k),$$

$$\frac{1}{K}\sum_{i,j=1}^{l}[a(x_i)\neq a(x_j)]\rho(x_i,x_j),$$

где |k| - количество элементов в кластере под номером k.

- По аналогии с квадратичным внутрикластерным и межкластерным расстоянием - *среднее квадратичное внутрикластерное* и *межкластерное расстояние*.

$$\sum_{k=1}^{K} \frac{1}{|k|} \sum_{i=1}^{l} [a(x_i) = k] \rho^2(x_i, c_k),$$

$$\frac{1}{K} \sum_{i,j=1}^{l} [a(x_i) \neq a(x_j)] \rho^2(x_i, x_j).$$

Силуэт (Silhouette) является мерой того, насколько объект похож на свой собственный кластер (сплоченность) по сравнению с другими кластерами (разделение).

$$s = \frac{1}{n} \sum \frac{b - a}{max(a, b)},$$

где a – среднее расстояние от данного объекта до объектов из того же кластера,

b – среднее расстояние от данного объекта до объектов из ближайшего кластера (отличного от того, в котором лежит сам объект)

Внешние метрики

Эти метрики используются, если есть дополнительные знания о кластеризуемой выборке, например, известно истинное распределение по кластерам. Задачу можно рассматривать как задачу многоклассовой классификации с использованием соответствующих метрик. В этом случае примерами могут быть:

Гомогенность (Homogenity) измеряет, насколько каждый кластер состоит из объектов одного класса. Определяются с использованием функций энтропии K – результат кластеризации, C – истинное разбиение выборки на классы

$$h = 1 - \frac{H(C|K)}{H(C)}$$

$$H(C|K) = -\sum_{j=1}^{m} \sum_{i=1}^{n} p(c_i, k_j) \log \frac{p(c_i, k_j)}{p(k_i)}$$

$$H(C) = -\sum_{i=1}^{n} p(c_i) \log p(c_i)$$

Полнота (Completeness) измеряет, насколько объекты одного класса относятся к одному кластеру.

$$c = 1 - \frac{H(K|C)}{H(K)}$$

$$H(K|C) = -\sum_{j=1}^{m}\sum_{i=1}^{n} p(k_i, c_j) \log \frac{p(k_i, c_j)}{p(c_i)}$$

$$H(K) = -\sum_{i=1}^{n} p(k_i) \log p(k_i)$$

V-мера (V-measure). Для учёта гомогенности и полноты одновременно вводится V-мера, как их среднее гармоническое:

$$v = 2\frac{hc}{h + c}$$

Принимают значения в диапазоне [0,1].