МИНОБРНАУКИ РОССИИ

РГУ нефти и газа (НИУ) имени И.М. Губкина

|  |  |
| --- | --- |
| Факультет | **Автоматики и Вычислительной техники** |
| Кафедра | **Информатики** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Оценка комиссии: |  | | | Рейтинг: |  |
| Подписи членов комиссии: | | | | | |
|  | |  | Вишневская Е. А. | | |
| (подпись) | |  | (фамилия, имя, отчество) | | |
|  | |  |  | | |
| (подпись) | |  | (фамилия, имя, отчество) | | |
|  | | | | | |
| (дата) | | | | | |
|  | |  |  | | |

**ДОМАШНЕЕ ЗАДАНИЕ №3**

|  |  |
| --- | --- |
| по дисциплине | Средства интеллектуального анализа данных и машинное |
| обучение | |

|  |  |
| --- | --- |
| на тему | Задача кластеризации |
|  | |
|  | |

|  |  |  |  |
| --- | --- | --- | --- |
| «К ЗАЩИТЕ» |  | ВЫПОЛНИЛ: |  |
|  |  | Студент группы | **АА-22-08** |
|  |  |  | (номер группы) |
| Доцент, к. ф. м. н.: Вишневская Е. А. |  | Сафуанов Артур Ришатович | |
| (должность, ученая степень; фамилия, и.о.) |  | (фамилия, имя, отчество) | |
|  |  |  | |
| (подпись) |  | (подпись) | |
|  |  |  | |
| (дата) |  | (дата) | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Москва, 20 | 25 |  |

Оглавление

[Введение 3](#_Toc167825323)

[Ход работы 4](#_Toc167825324)

[Список используемой литературы 3](#_Toc167825325)4

**Введение**

Кластеризация — один из ключевых методов машинного обучения без учителя, предназначенный для автоматического группирования данных на основе их схожести. Эта задача особенно полезна в ситуациях, когда отсутствуют явные метки, но требуется выявить скрытые структуры или закономерности в данных. Цель данной работы — изучить алгоритмы кластеризации, такие как K-средних, Mean-Shift и DBSCAN, а также научиться применять их на практике с использованием современных инструментов анализа данных. В ходе выполнения задания будут рассмотрены принципы работы алгоритмов, их настройка, оценка качества и визуализация результатов. Актуальность темы обусловлена широким применением кластеризации в маркетинге, биологии, обработке изображений и других областях, где требуется автоматическая группировка объектов.

**Ход работы**

Задание 1: изучить алгоритмы машинного обучения для решения задачи кластеризации.

Решение:

1. **Метод K-средних (K-Means)**
   * **Принцип работы**: Алгоритм делит данные на *K* кластеров, минимизируя внутрикластерную дисперсию. Начальные центры кластеров выбираются случайно, затем итеративно обновляются до сходимости.
   * **Особенности**: требует заранее заданного числа кластеров (*K*), чувствителен к выбросам и начальной инициализации.
2. **Mean-Shift**
   * **Принцип работы**: алгоритм "сдвигает" точки данных в направлении наибольшей плотности, пока они не сойдутся к модам распределения. Количество кластеров определяется автоматически.
   * **Особенности**: не требует задания числа кластеров, эффективен для данных с переменной плотностью, но вычислительно затратен.
3. **DBSCAN (Density-Based Spatial Clustering)**
   * **Принцип работы**: Кластеры формируются как области высокой плотности, разделенные областями низкой плотности. Алгоритм выделяет ядровые точки, соединяя их с соседями в пределах радиуса *ε*.
   * **Особенности**: Устойчив к выбросам, автоматически определяет число кластеров, хорошо работает с кластерами произвольной формы.

Задание 2: изучить соответствующие библиотеки и инструменты для анализа данных в Python.

Решение:

Для реализации задач кластеризации в Python используются следующие библиотеки:

* **scikit-learn**: основная библиотека для машинного обучения. Содержит реализации алгоритмов K-Means (KMeans), Mean-Shift (MeanShift) и DBSCAN (DBSCAN).
* **pandas**: для загрузки, обработки и анализа структурированных данных (DataFrame).
* **numpy**: для выполнения численных операций и работы с многомерными массивами.
* **matplotlib** и **seaborn**: для визуализации данных (построение графиков, диаграмм рассеяния, heatmap).
* **scipy**: для дополнительных операций с расстояниями и иерархической кластеризацией.

Задание 3: подобрать или сгенерировать данные с подходящей структурой для решения задачи кластеризации.

Решение:

В качестве данных были взяты несколько датасетов, которые содержат специфичную форму, чтобы подчеркнуть эффективность алгоритмов.

Импортируем их:

from sklearn.datasets import make\_moons, make\_blobs



Начнём с просмотра набора данных – капли, где чётко разделены 3 кластера:

X, y = make\_blobs(n\_samples=200,

n\_features=2,

centers=3,

cluster\_std=0.5,

shuffle=True,

random\_state=42)

plt.scatter(X[:, 0], X[:, 1],

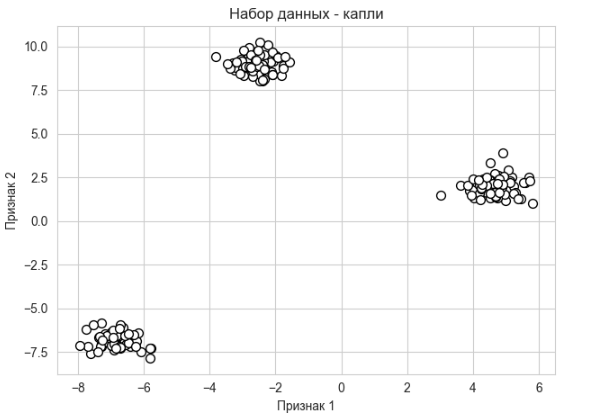
c='white', marker='o', edgecolor='black', s=50)

plt.xlabel('Признак 1')

plt.ylabel('Признак 2')

plt.title('Набор данных - капли')

plt.tight\_layout()



Теперь посмотрим на набор данных – полумесяцы, с которым как раз таки обычно плохо справляются основные алгоритмы кластеризации:

X1, y1 = make\_moons(n\_samples=200, noise=0.02, random\_state=42)

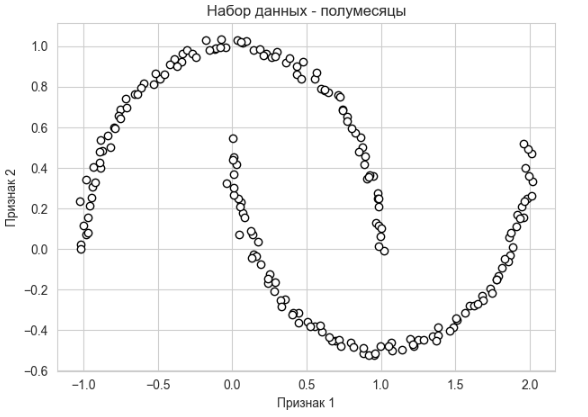
plt.scatter(X1[:, 0], X1[:, 1], c='white', edgecolor='black')

plt.xlabel('Признак 1')

plt.ylabel('Признак 2')

plt.title('Набор данных - полумесяцы')

plt.tight\_layout()



Задание 4: построить модели на основе указанных методов, используя соответствующие библиотеки Python.

Решение:

Проанализируем все алгоритму по порядку и без оптимизации, этим мы займёмся потом, а сейчас главное посмотреть на работу моделей.

**Kmeans на каплях:**

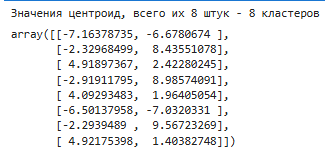
kmeans\_base\_blobs = KMeans()

preds = kmeans\_base\_blobs.fit\_predict(X)

Выведем количество получившихся центроид:

print('Значения центроид, всего их 8 штук - 8 кластеров')

kmeans\_base\_blobs.cluster\_centers\_



Проведём визуализацию обученной модели:

plt.scatter(X[:, 0], X[:, 1],

c='white', marker='o', edgecolor='black', s=50)

plt.scatter(kmeans\_base\_blobs.cluster\_centers\_[:, 0],

kmeans\_base\_blobs.cluster\_centers\_[:, 1],

s=150, marker='8',

c='lightgreen', edgecolor='black',

label='Центроиды')

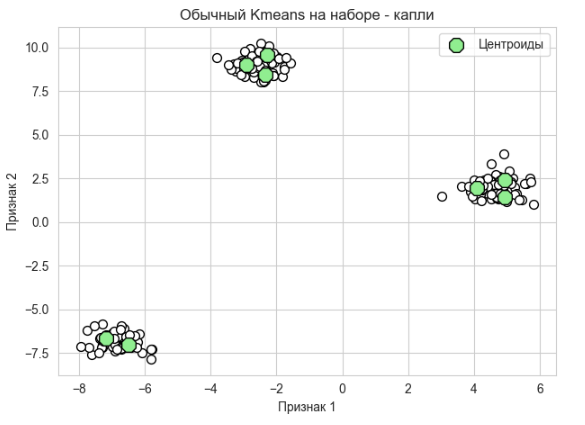
plt.xlabel('Признак 1')

plt.ylabel('Признак 2')

plt.legend(scatterpoints=1)

plt.title('Обычный Kmeans на наборе - капли')

plt.tight\_layout()



Как видим, Kmeans очень плохо разделил кластеры, так как по умолчанию n\_clusters = 8.

**Kmeans на полумесяцах:**

kmeans\_base\_moons = KMeans()

preds = kmeans\_base\_moons.fit\_predict(X1)

plt.scatter(X1[:, 0], X1[:, 1], c='white', edgecolor='black')

plt.scatter(kmeans\_base\_moons.cluster\_centers\_[:, 0],

kmeans\_base\_moons.cluster\_centers\_[:, 1],

s=150, marker='8',

c='lightgreen', edgecolor='black',

label='Центроиды')

plt.xlabel('Признак 1')

plt.ylabel('Признак 2')

plt.legend(scatterpoints=1)

plt.title('Обычный Kmeans на наборе - полумесяцы')

plt.tight\_layout()



Здесь тоже самое, плохая кластеризация за счёт большого количества центроид.

**DBSCAN на каплях:**

Этот метод основывается не на вычислениях расстояния между точками, а на плотности данных, поэтому он должен лучше отработать на втором наборе данных даже без настройки.

dbscan\_base\_blobs = DBSCAN()

preds\_db = dbscan\_base\_blobs.fit\_predict(X)

plt.scatter(X[preds\_db == 0, 0],

X[preds\_db == 0, 1],

s=50, c='lightgreen',

marker='s', edgecolor='black',

label='Кластер 1')

plt.scatter(X[preds\_db == 1, 0],

X[preds\_db == 1, 1],

s=50, c='orange',

marker='o', edgecolor='black',

label='Кластер 2')

plt.scatter(X[preds\_db == 2, 0],

X[preds\_db == 2, 1],

s=50, c='lightblue',

marker='v', edgecolor='black',

label='Кластер 3')

plt.scatter(X[preds\_db == -1, 0],

X[preds\_db == -1, 1],

s=50, c='red',

marker='8', edgecolor='black',

label='Выбросы')

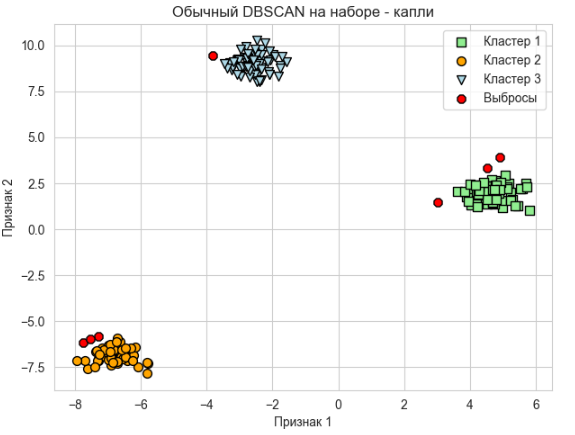
plt.xlabel('Признак 1')

plt.ylabel('Признак 2')

plt.legend()

plt.tight\_layout()

plt.title('Обычный DBSCAN на наборе - капли');



Помимо того, что DBSCAN отлично отделил кластеры, так он ещё и имеет особенность находить выбросы, которые не входят ни в один из кластеров. На практике этот метод чаще всего используют ради этого.

**DBSCAN на полумесяцах:**

dbscan\_base\_moons = DBSCAN()

preds\_db = dbscan\_base\_moons.fit\_predict(X1)

plt.scatter(X1[preds\_db == 0, 0], X1[preds\_db == 0, 1],

c='lightblue', marker='o', s=40,

edgecolor='black',

label='Кластер 1')

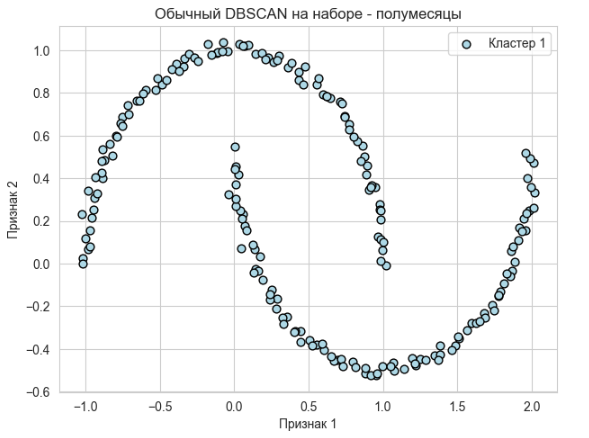
plt.xlabel('Признак 1')

plt.ylabel('Признак 2')

plt.legend()

plt.tight\_layout()

plt.title('Обычный DBSCAN на наборе - полумесяцы');



На удивление, здесь он определил 1 кластер, но это из-за того, что по умолчанию в алгоритме стоит большой радиус рассматриваемых точек. Далее это будет поправлено.

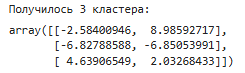
**Mean-Shift на каплях:**

ms\_base\_blob = MeanShift()

preds\_ms = ms\_base\_blob.fit\_predict(X)

print('Получилось 3 кластера:')

ms\_base\_blob.cluster\_centers\_



Здесь он выделил 3 центра, как и полагается.

plt.scatter(X[preds\_ms == 0, 0],

X[preds\_ms == 0, 1],

s=50, c='lightgreen',

marker='s', edgecolor='black',

label='Кластер 1')

plt.scatter(X[preds\_ms == 1, 0],

X[preds\_ms == 1, 1],

s=50, c='orange',

marker='o', edgecolor='black',

label='Кластер 2')

plt.scatter(X[preds\_ms == 2, 0],

X[preds\_ms == 2, 1],

s=50, c='lightblue',

marker='v', edgecolor='black',

label='Кластер 3')

plt.scatter(ms\_base\_blob.cluster\_centers\_[:, 0],

ms\_base\_blob.cluster\_centers\_[:, 1],

s=150, marker='8',

c='red', edgecolor='black',

label='Центроиды')

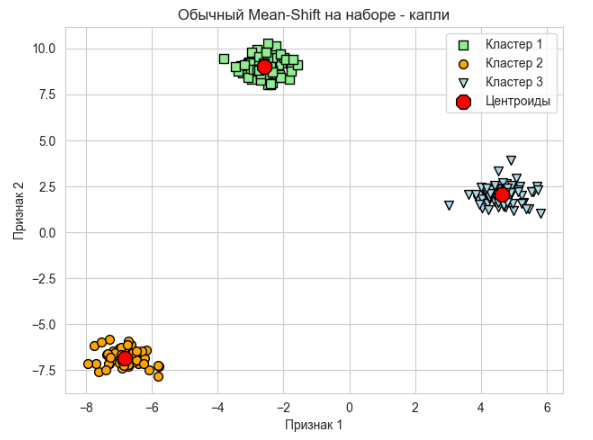
plt.xlabel('Признак 1')

plt.ylabel('Признак 2')

plt.legend()

plt.tight\_layout()

plt.title('Обычный Mean-Shift на наборе - капли');



Здесь уже произведена кластеризация с выделением явных центроидов, отличная работа. Этот метод заточен на то, чтобы смещать свои центры в процессе обучения к более плотным областям.

**Mean-Shift на полумесяцах:**

ms\_base\_moons = MeanShift()

preds\_ms = ms\_base\_moons.fit\_predict(X1)

print('Получилось 2 кластера:')

ms\_base\_moons.cluster\_centers\_



По полумесяцам уже получилось 2 кластера, что наталкивает на правильную кластеризацию, но посмотрим на график.

plt.scatter(X1[preds\_ms == 0, 0],

X1[preds\_ms == 0, 1],

s=50, c='lightblue',

marker='s', edgecolor='black',

label='Кластер 1')

plt.scatter(X1[preds\_ms == 1, 0],

X1[preds\_ms == 1, 1],

s=50, c='red',

marker='o', edgecolor='black',

label='Кластер 2')

plt.scatter(ms\_base\_moons.cluster\_centers\_[:, 0],

ms\_base\_moons.cluster\_centers\_[:, 1],

s=150, marker='8',

c='lightgreen', edgecolor='black',

label='Центроиды')

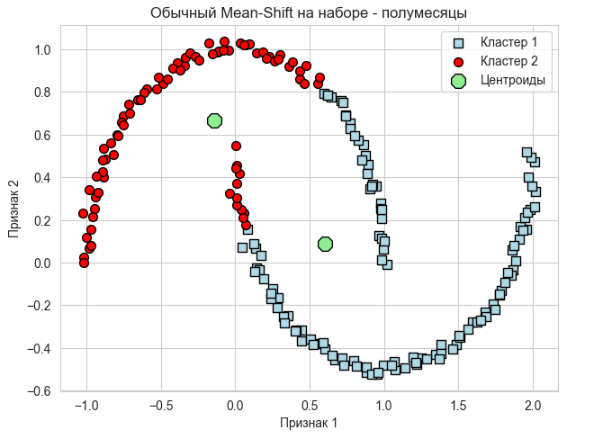
plt.xlabel('Признак 1')

plt.ylabel('Признак 2')

plt.legend()

plt.tight\_layout()

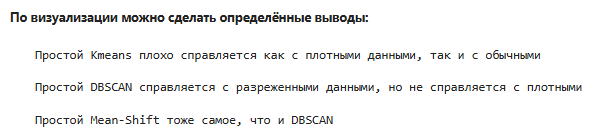
plt.title('Обычный Mean-Shift на наборе - полумесяцы');



Видим, что хотя алгоритм и определил 2 кластера, но это не совсем то, что ожидалось. Каждый полумесяц должен быть выделен отдельным цветом. Да и центроиды явно находятся не на своих местах.

Задание 5: оценить результаты моделирования, произвести настройку параметров модели.

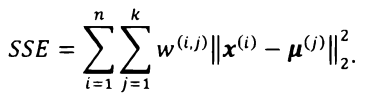
Решение:



Теперь перейдём к самому интересному этапу, настройке гиперпараметров, чтобы помочь моделям правильно определять кластеры.

**Настройка Kmeans - капли:**

Для настройки этого алгоритма я решил применить “Метод локтя”, так как мы заранее можем задать количество кластеров, а встроенный метод \_inertia позволяет вывести значение искажения, которое считается по формуле SSE.



distortions\_blobs, distortions\_moons = [], []

for i in range(1, 11):

km = KMeans(n\_clusters=i,

init='k-means++',

n\_init=10,

max\_iter=300,

random\_state=42)

km.fit(X)

distortions\_blobs.append(km.inertia\_)

km = KMeans(n\_clusters=i,

init='k-means++',

n\_init=10,

max\_iter=300,

random\_state=42)

km.fit(X1)

distortions\_moons.append(km.inertia\_)

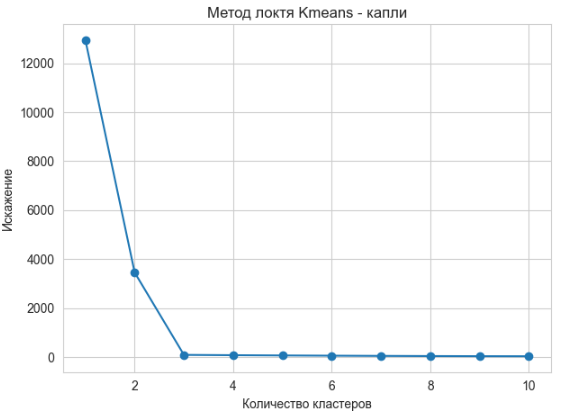
plt.plot(range(1, 11), distortions\_blobs, marker='o')

plt.xlabel('Количество кластеров')

plt.ylabel('Искажение')

plt.title('Метод локтя Kmeans - капли')

plt.tight\_layout()



После 3 кластера график становится более сбалансированным, и по логике мы должны выбрать 3 кластера, следовательно и возьмём это количество.

kmeans\_blobs = KMeans(n\_clusters=3, random\_state=42)

preds = kmeans\_blobs.fit\_predict(X)

plt.scatter(X[preds == 0, 0],

X[preds == 0, 1],

s=50, c='lightgreen',

marker='s', edgecolor='black',

label='Кластер 1')

plt.scatter(X[preds == 1, 0],

X[preds == 1, 1],

s=50, c='orange',

marker='o', edgecolor='black',

label='Кластер 2')

plt.scatter(X[preds == 2, 0],

X[preds == 2, 1],

s=50, c='lightblue',

marker='v', edgecolor='black',

label='Кластер 3')

plt.scatter(kmeans\_blobs.cluster\_centers\_[:, 0],

kmeans\_blobs.cluster\_centers\_[:, 1],

s=150, marker='8',

c='red', edgecolor='black',

label='Центроиды')

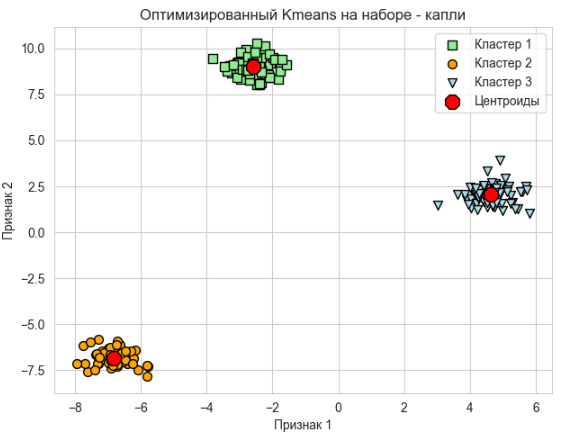
plt.xlabel('Признак 1')

plt.ylabel('Признак 2')

plt.legend()

plt.tight\_layout()

plt.title('Оптимизированный Kmeans на наборе - капли');



Установив значение n\_clusters = 3, мы наконец получили правильную кластеризацию с хорошими центроидами.

**Настройка Kmeans - полумесяцы:**

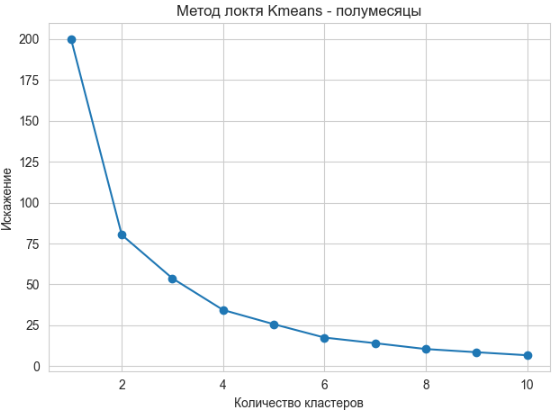
plt.plot(range(1, 11), distortions\_moons, marker='o')

plt.xlabel('Количество кластеров')

plt.ylabel('Искажение')

plt.title('Метод локтя Kmeans - полумесяцы')

plt.tight\_layout()



На этом графике уже не всё так очевидно, потому что задача кластеризации сложно даётся Kmeans, ведь полумесяцы близко расположены друг к другу, что может сбивать с толку. Выберем 2 кластера, так как это является решением этой задачи и посмотрим, что покажет модель.

kmeans\_moons = KMeans(n\_clusters=2, random\_state=42)

preds = kmeans\_moons.fit\_predict(X1)

plt.scatter(X1[preds == 0, 0],

X1[preds == 0, 1],

s=50, c='lightgreen',

marker='s', edgecolor='black',

label='Кластер 1')

plt.scatter(X1[preds == 1, 0],

X1[preds == 1, 1],

s=50, c='orange',

marker='o', edgecolor='black',

label='Кластер 2')

plt.scatter(kmeans\_moons.cluster\_centers\_[:, 0],

kmeans\_moons.cluster\_centers\_[:, 1],

s=150, marker='8',

c='red', edgecolor='black',

label='Центроиды')

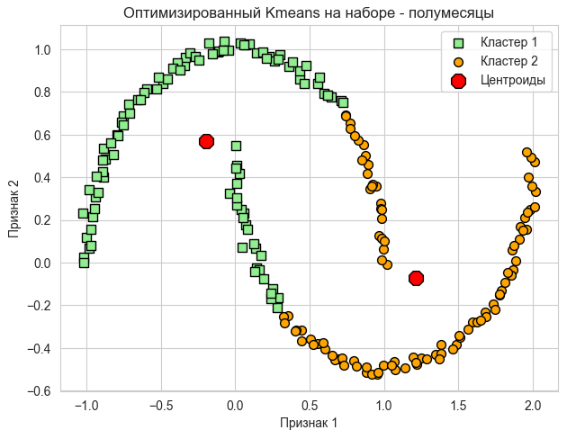
plt.xlabel('Признак 1')

plt.ylabel('Признак 2')

plt.legend()

plt.tight\_layout()

plt.title('Оптимизированный Kmeans на наборе - полумесяцы');



Картина примерно такая же, как и с Mean-Shift, он не справился с кластеризацией.

**Настройка DBSCAN - полумесяцы:**

Я не стал рассматривать капли для этого алгоритма, потому что он и так показал себя хорошо в этой задаче. Намного интереснее разобраться с полумесяцами.

Здесь не вычисляются расстояния, поэтому рассмотрим ещё один способ выбора количества кластеров, *силуэтный график* – он считает коэффициенты силуэтов для каждой точки. Считается хорошим разделением на кластеры, когда на графике силуэты имеют примерно одинаковую длину и ширину.

Функция для отрисовки графика:

def print\_siluette(model, name, param, name\_param):

cluster\_labels = np.unique(preds)

n\_clusters = cluster\_labels.shape[0]

silhouette\_vals = silhouette\_samples(X, preds, metric='euclidean')

y\_ax\_lower, y\_ax\_upper = 0, 0

yticks = []

for i, c in enumerate(cluster\_labels):

c\_silhouette\_vals = silhouette\_vals[preds == c]

c\_silhouette\_vals.sort()

y\_ax\_upper += len(c\_silhouette\_vals)

color = plt.cm.jet(float(i) / n\_clusters)

plt.barh(range(y\_ax\_lower, y\_ax\_upper), c\_silhouette\_vals, height=1.0,

edgecolor='none', color=color)

yticks.append((y\_ax\_lower + y\_ax\_upper) / 2.)

y\_ax\_lower += len(c\_silhouette\_vals)

silhouette\_avg = np.mean(silhouette\_vals)

plt.axvline(silhouette\_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster\_labels + 1)

plt.ylabel('Кластер')

plt.xlabel('Коэффициент Силуэта')

plt.title(f'{name} при {name\_param}={param}')

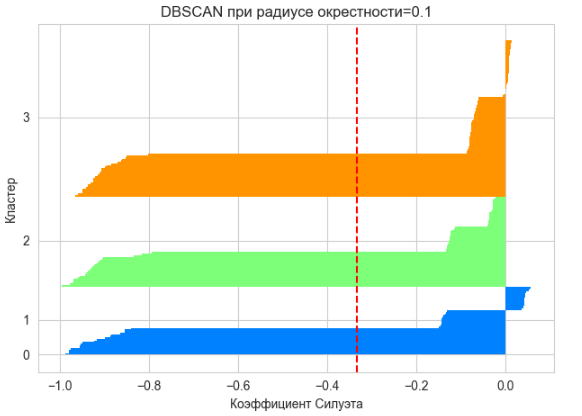
plt.tight\_layout()

Для начала возьмём маленький показатель радиуса, eps = 0.1:

db = DBSCAN(eps=0.1)

preds = db.fit\_predict(X1)

print\_siluette(db, 'DBSCAN', 0.1, 'радиусе окрестности')



Не совсем хорошие формы силуэтов свидетельствуют о плохой кластеризации, к тому же, нам необходимо 2 кластера для этих данных, а не 3, проверим на графике:

plt.scatter(X1[preds == 0, 0], X1[preds == 0, 1],

c='lightblue', marker='o', s=40,

edgecolor='black',

label='Кластер 1')

plt.scatter(X1[preds == 1, 0], X1[preds == 1, 1],

c='red', marker='s', s=40,

edgecolor='black',

label='Кластер 2')

plt.scatter(X1[preds == 2, 0], X1[preds == 2, 1],

c='orange', marker='\*', s=40,

edgecolor='black',

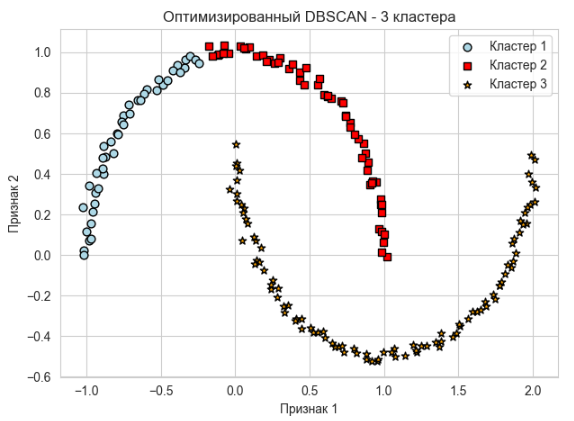
label='Кластер 3')

plt.xlabel('Признак 1')

plt.ylabel('Признак 2')

plt.legend()

plt.tight\_layout()



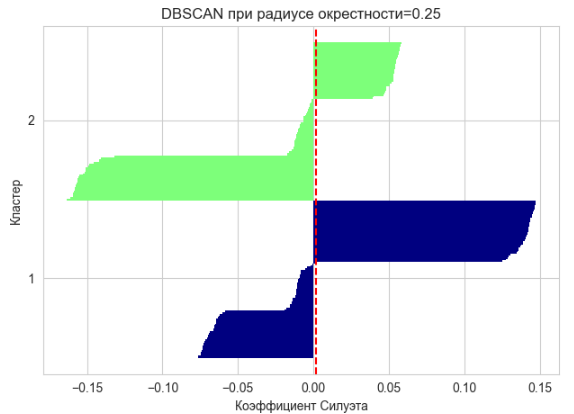
Так и есть – плохая кластеризация.

Теперь увеличим eps, чтобы количество кластеров уменьшилось:

db = DBSCAN(eps=0.25)

preds = db.fit\_predict(X1)

print\_siluette(db, 'DBSCAN', 0.25, 'радиусе окрестности')



Силуэты хоть и отличаются на первый взгляд, но если присмотреться, то они симметричны, как и наши специфичные полумесяцы.

plt.scatter(X1[preds == 0, 0], X1[preds == 0, 1],

c='lightblue', marker='o', s=40,

edgecolor='black',

label='Кластер 1')

plt.scatter(X1[preds == 1, 0], X1[preds == 1, 1],

c='red', marker='s', s=40,

edgecolor='black',

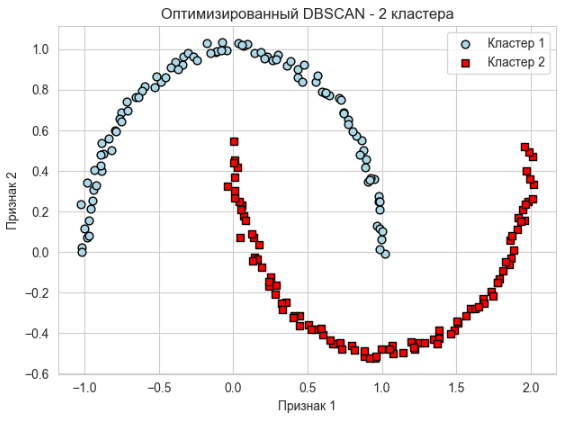
label='Кластер 2')

plt.xlabel('Признак 1')

plt.ylabel('Признак 2')

plt.title('Оптимизированный DBSCAN - 2 кластера')

plt.legend()

plt.tight\_layout()  


А теперь алгоритм безупречно отделил кластеры при оптимальном радиусе!

**Настройка Mean-Shift - полумесяцы:**

В нём я тоже не вижу смысла отдельно смотреть на капли, так как он там справился.

Лучше посмотрим на полумесяцы, возможно, он не справится с этой задачей, так как отталкивается от центроидов.

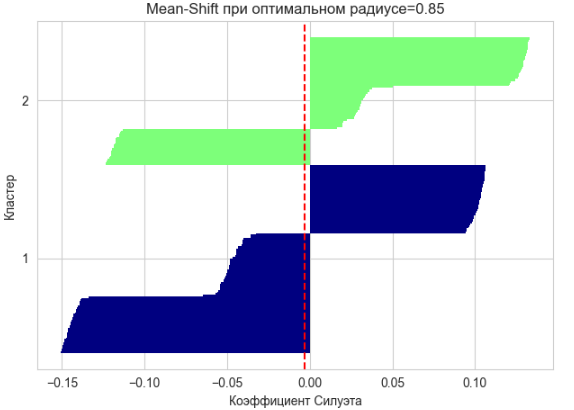
В sklearn есть специальный метод, который позволяет как раз определить оптимальный радиус окрестности, исходя из наших данных, применим его.

bandwidth = estimate\_bandwidth(X1, quantile=0.3, n\_samples=500)

ms = MeanShift(bandwidth=bandwidth)

preds = ms.fit\_predict(X1)

print\_siluette(ms, 'Mean-Shift', round(bandwidth, 2), 'оптимальном радиусе')



Хорошим радиусом было определено 0.85, но не стоит радоваться, ведь в данном случае симметричные части силуэтов отличаются по форме, что может наталкивать на плохую кластеризацию, визуализируем:

plt.scatter(X1[preds == 0, 0], X1[preds == 0, 1],

c='lightblue', marker='o', s=40,

edgecolor='black',

label='Кластер 1')

plt.scatter(X1[preds == 1, 0], X1[preds == 1, 1],

c='red', marker='s', s=40,

edgecolor='black',

label='Кластер 2')

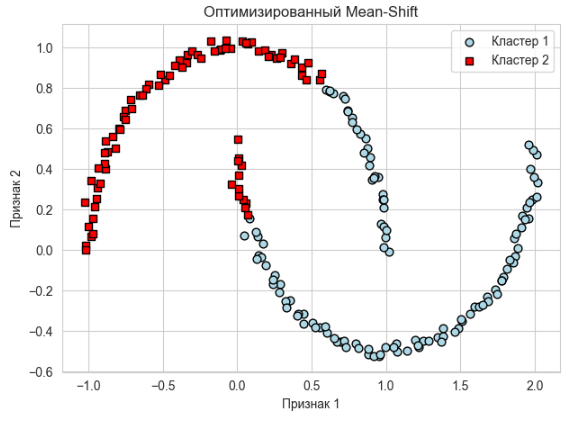
plt.xlabel('Признак 1')

plt.ylabel('Признак 2')

plt.title('Оптимизированный Mean-Shift')

plt.legend()

plt.tight\_layout()



Таким образом, Mean-Shift также не справился с этими данными. Можно сделать вывод, что только методы на основе плотностного анализа могут эффективно работать на таких образцах.

Задание 6: визуализировать результаты.

Решение:

Все необходимые графики были построены, единственное, приведём итоговую таблицу гиперпараметров.

print('ОПТИМАЛЬНЫЕ ПАРАМЕТРЫ ДЛЯ АЛГОРИТМОВ')

columns = pd.MultiIndex.from\_tuples([

('Количество кластеров', 'капли'),

('Количество кластеров', 'полумесяцы'),

('Радиус окрестности', 'капли'),

('Радиус окрестности', 'полумесяцы'),

])

data = pd.DataFrame(columns=columns,

data=[[3, 2, "-", "-"], ['-', '-', 0.5, 0.25], ['-', '-', 1, 0.85]])

data.index = ['Kmeans', 'DBSCAN', 'Mean-Shift']

data



**Список используемой литературы**

1. McKinney, W. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. — 2nd Edition. — O'Reilly Media, 2017. — 544 p.
2. Pedregosa, F., et al. Scikit-learn: Machine Learning in Python // Journal of Machine Learning Research. — 2011. — Vol. 12. — P. 2825–2830.
3. Waskom, M. L. seaborn: statistical data visualization // Journal of Open Source Software. — 2021. — Vol. 6, № 60. — P. 3021.
4. Hunter, J. D. Matplotlib: A 2D Graphics Environment // Computing in Science & Engineering. — 2007. — Vol. 9, № 3. — P. 90–95.
5. VanderPlas, J. Python Data Science Handbook: Essential Tools for Working with Data. — O'Reilly Media, 2016. — 548 p.