СОДЕРЖАНИЕ

ВВЕДЕНИЕ

1.1 Актуальность темы

В современном мире, где объемы данных стремительно растут, анализ и интерпретация больших данных становятся все более значимыми задачами. Кластеризация, как один из методов машинного обучения, играет важную роль в выявлении скрытых структур и паттернов в данных. Классические методы кластеризации, такие как k-средние (k-means) и иерархическая кластеризации, являются основополагающими инструментами, которые применяются в различных областях, например, маркетинг, экономика, обработка изображений и анализ текстов.

Актуальность исследования классических методов кластеризации обусловлена их применимостью в решении множества прикладных задач. Несмотря на развитие новых методов, таких как, например, методы глубокого обучения, классические алгоритмы остаются востребованными благодаря своей простоте, эффективности и интерпретируемости. Понимание и правильное применение этих методов позволяет не только эффективно обрабатывать данные, но и закладывает фундамент для освоения более сложных алгоритмов.

1.2 Цель и задачи исследования

Целью данного исследования является ознакомление с классическими методами кластеризации, их принципами работы, а также сравнительный анализ эффективности и применимости этих методов в различных сценариях.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить основные понятия и особенности классических алгоритмов кластеризации;
2. Провести обзор основных классических методов кластеризации, таких как k-средние (k-means) и иерархическая кластеризация;
3. Провести сравнительный анализ методов на основе различных критериев, таких как скорость работы, устойчивость к шуму, точность кластеризации и интерпретируемость результатов;
4. Реализовать два классических алгоритма: иерархический и k-means;
5. Применить реализованные алгоритмы к одному набору данных и сравнить полученные результаты;
6. Сделать выводы о преимуществах и недостатках каждого метода.

Литературный обзор

2.1 Основная информация о классических методах кластеризации

Задачи машинного обучения делятся на обучение с учителем и без учителя. Обучение без учителя (unsupervised learning) подразумевает создание модели машинного обучения без необходимости использования размеченных тренировочных данных.

В отличие от задач с помеченными данными, где алгоритмы обучаются классифицировать данные на основе предоставленных меток, в реальных задачах часто отсутствуют размеченные данные. В таких случаях может быть множество данных, которые необходимо распределить по категориям, количество и характеристики которых заранее неизвестны. Здесь и используется обучение без учителя. Алгоритмы этого типа создают модели, способные находить подгруппы в наборе данных, используя различные метрики сходства. Данные в подгруппах, найденных алгоритмами обучения без учителя, обладают схожими характеристиками или свойствами, но отличаются от других данных групп в наборе данных.

Когда имеется набор данных без меток, предполагается, что эти данные формируются под влиянием скрытых переменных, управляющих их распределением. Обучение в таких условиях может следовать иерархической схеме, начиная с индивидуальных точек данных и постепенно переходя к более сложным уровням их представления.

Кластеризация представляет собой один из методов обучения без учителя. Основная задача заключается в выявлении скрытых характеристик данных, определяющих их принадлежность к одной и той же подгруппе. Универсальных метрик сходства, подходящих для всех случаев, не существует – всё зависит от конкретной задачи. Например, может интересовать определение представительной точки данных для каждой подгруппы или обнаружение выбросов. В зависимости от ситуации выбирается наиболее подходящая метрика, учитывающая специфику задач.

Классические методы кластеризации

3.1 Иерархический метод

Иерархическая кластеризация представляет собой метод группировки данных, при котором создается иерархия (дерево) вложенных кластеров. Множество объектов характеризуется определенной степенью связности, т.е.

в иерархической кластеризации каждый объект из множества связан с другими объектами. На основе степеней связности и строится древо кластеров.

Новые кластеры могут создаваться двумя методами: агломеративными и дивизионными.

- В агломеративных методах каждый объект рассматривается как отдельный кластер, и на каждом шаге объединяются два наиболее близких кластера до тех пор, пока все объекты не будут объединены в один кластер. Этот процесс визуализируется с помощью древовидной структуры (дендрограммы).
- Дивизионные методы, в отличие от агломеративных, начинают с одного большого кластера, включающего все объекты, и постепенно делят его на более мелкие кластеры. Хотя дивизионные методы используются реже, они могут быть полезны в некоторых специфических задачах, например сегментация клиентов в маркетинге, данные разделяются на кластеры до тех пор, пока не примут необходимый вид.

На каждом шаге необходимо уметь быстро подсчитывать расстояние от образовавшегося кластера W=U ∪ V до любого другого кластера S, используя известные расстояния с предыдущих шагов. Это легко выполняется при использовании формулы, предложенной Лансом и Уильямсом в 1967 году:

$$\mathbf{R(W,S)} = \boldsymbol{\alpha}_U \times \mathbf{R(U,S)} + \boldsymbol{\alpha}_V \times \mathbf{R(V,S)} + \boldsymbol{\beta} \times \mathbf{R(U,V)} +$$

$$+ \boldsymbol{\gamma} \times |\mathbf{R(U,S)} - \mathbf{R(V,S)}|$$

Где: $\boldsymbol{\alpha}_U, \boldsymbol{\alpha}_V, \boldsymbol{\beta}, \boldsymbol{\gamma}$ – параметры, зависящие от выбранной функции подсчета.

Методы для вычисления расстояния между кластерами:

- **метод одиночной связи:** $\alpha_U = {}^1\!/_2,\ \alpha_V = {}^1\!/_2,\ \beta = 0,\ \gamma = -{}^1\!/_2$
- **метод полной связи:** $\alpha_U = {}^1\!/_2,\ \alpha_V = {}^1\!/_2,\ \beta = 0,\ \gamma = -{}^1\!/_2$
- **метод средней связи:** $\alpha_U = {}^{|U|}\!/_{|W|},\ \alpha_V = {}^{|V|}\!/_{|W|},\ \beta = -\alpha_U \times \alpha_V,\ \gamma = 0$
- **метод Уорда:** $\alpha_U = {}^{(|U| + |S|)}\!/_{(|W| + |S|)},\ \alpha_V = {}^{(|U| + |S|)}\!/_{(|W| + |S|)},$

  $\beta = -{}^{|S|}\!/_{(|S| + |W|)},\quad \gamma = 0$

3.2 Алгоритм k-means

K-means или метод k-средних – алгоритм кластеризации, работающий с уже известным количеством кластеров. Каждому объекту назначается кластер, центроид (центр тяжести кластера, другое название – главная точка) которого находится ближе всего по выбранной метрике расстояния. Далее центроид кластера пересчитывается. Итеративный процесс продолжается до тех пор, пока центроиды не примут оптимальные положения. Очевидно, что выбор положения кластеров важен для меньшего числа итераций и более верного разбиения данных на кластеры.

Алгоритм k-средних имеет ряд улучшенных алгоритмов:

- K-means++

Улучшенная инициализация центроидов, снижающая вероятность худшей кластеризации и увеличивающая скорость сходимости.

- Mini-batch k-means

Используется для работы с большими данными, использующая мини-батчи для обновления центроидов и уменьшающая вычислительные затраты

Реализация алгоритмов

Реализация методов k-means и иерархической кластеризации в данной работе проводится на языке Python в среде разработки Jupiter Notebook. Для работы импортируются следующие библиотеки:

- Библиотека *numpy* предоставляет реализации вычислительных алгоритмов, оптимизированные для работы с многомерными массивами.
- Библиотека *matplotlib* необходима для визуализации данных.
- *random* импортируется за неимением готового датасета, потому в примере с разбиением точек на плоскости на различные кластеры, расположения этих точек задаются случайным образом.
- *pandas* — программная библиотека на языке Python для обработки и анализа данных. Предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временны́ми рядами.

4.1 Реализация метода k-means

На этапе инициализации функция присваивает случайным точкам заданного количества наименование центроида. Другие точки проверяются на принадлежность к кластерам по наименьшему евклидову расстоянию до центроидов. При каждой итерации пересчитываются центроиды как среднее

значение координат точек в кластерах. При совпадении значений старого и нового множеств центроидов функция прекращает работу, возвращая массивы кластеров и центроидов.

Итоговые значения центроидов отмечены в визуализированных данных черным цветом. Код представлен в дополнении.

4.2 Реализация иерархической кластеризации

Для иерархической кластеризации в данной работе использован метод полной связи при подсчете расстояний между кластерами по формуле Ланса и Уильямса.

Функция иерархической кластеризации на вход получает двумерный массив точек из набора данных с координатами по осям X и Y, а также ожидаемое количество кластеров (Эта переменная используется для своевременной остановки работы алгоритма, т.к. в противном случае он данные «объединит» в один кластер). На нулевом шаге каждая точка является кластером. Подсчитываются расстояния между всеми кластерами методом полной связи путём вызова вышеописанной функции. Пока количество кластеров больше входного параметра алгоритм объединяет кластеры: создается новый массив кластеров, в который добавляются новые (объединенные) кластеры в порядке увеличения расстояний.

Функция возвращает эту матрицу номеров кластеров, к которым относится каждая точка. Функции подсчета расстояний и иерархической кластеризации представлены в дополнении.

Наборы данных для тестирования

Рассматриваемый набор данных для тестирования алгоритмов – множество точек, сгенерированных случайным образом в пределах заданных функций. Рассматривается процесс кластеризации этих точек, который реализуется через вычисление расстояний между ними

5.1 Случайные точки внутри трёх окружностей

Точки генерируются случайным образом внутри трех окружностей в декартовой системе координат с центрами в заданных точках с целью уменьшить расстояния между данными. Кроме того генерируется «шум» – точки со случайными координатами в промежутке (-15, 15). Создаётся двумерный массив данных с определенным количеством ячеек. Функция генерации данных представлена в дополнении..

Так как функции кластеризации добавляют к набору данных новый столбец с указанием на номер кластера для каждой точки, создаются копии исходного датасета.

Вводится переменная k – ожидаемое число кластеров, а также исполняются функции подсчета. В решении задач k – гиперпараметр, который подбирается для улучшения качеств кластеризации. Однако в данной работе тестовые данные предполагают определенные значения параметра k. Описание кода представлено в приложении.

Далее представлена визуализация кластеризации, представленная в приложении (1.1 и 1.2).

Далее копирование набора данных и вызов функций не будет указан, т.к. они станутся без изменений. Меняться будет только переменная k.

5.2 Спираль

Точки генерируются случайным образом в полярных координатах, образуя две спирали (по часовой стрелке и против часовой стрелки) с малым значением «шума» для лучшей кучности точек. Функция генерации точек представлена в дополнении.

Ожидаемое количество кластеров (переменная k) в этой задаче будет равно 2.

Визуализация иерархической кластеризации и k-means представлены в приложении (2.1 и 2.2).

5.3 Круги внутри двух полуокружностей

Точки генерируются случайным образом, образуя 4 фигуры: две полуокружности (верхняя и нижняя) и два круга внутри каждой окружности. Все фигуры создаются с малым шумом для большей кучности точек в них. Функция генерации точек представлена в дополнении.

Ожидаемое количество кластеров (переменная k) в этой задаче будет равно 4.

Визуализация иерархической кластеризации и k-means с этим набором данных представлены в приложении (3.1 и 3.2).

ЗАКЛЮЧЕНИЕ

В данной работе были изучены классические методы кластеризации в машинном обучении, а именно метод k-means и иерархическая кластеризация. Изучены алгоритмы и их работа. Функции с алгоритмами были использованы на практике с наборами данных: точки, сгенерированные случайным образом внутри трех окружностей, в форме двух спиралей и двух кругов между двумя полуокружностями. В результате исследования выявлены особенности каждого метода. Например, на рассматриваемых наборах данных была отмечена высокая скорость работы алгоритма k-means, в то время как алгоритм иерархической кластеризации проявил себя менее эффективно. Однако стоит отметить, что данные выводы основаны на проведенных экспериментах и могут быть зависимы от конкретного набора данных и условий эксперимента.

ПРИЛОЖЕНИЕ

Дополнение к п. 4.1:

```
def k_means(X, k=3, iteration=1000):
    centroids = X[np.random.choice(range(len(X)), k, replace=False)]
    for _ in range(iteration):
        # Расчет расстояний между точками и центроидами
        distances = np.sqrt(((X - centroids[:, np.newaxis])**2).sum(axis=2))

        # Назначение кластеров на основе ближайшего центроида
        clusters = np.argmin(distances, axis=0)

        # Обновление центроидов как среднее значение точек в каждом кластере
        new_centroids = np.array([X[clusters == i].mean(axis=0) for i in
range(k)])

        # Проверка на сходимость
        if np.all(centroids == new_centroids):
            break
        centroids = new_centroids

    return clusters, centroids
```

Дополнение к п. 4.2 (1):

```
def complete_linkage_distance(U, V, S):
    # Calculate distances
    R_U_S = np.max(pdist(np.vstack([U, S]), 'euclidean'))
    R_V_S = np.max(pdist(np.vstack([V, S]), 'euclidean'))
    R_U_V = np.max(pdist(np.vstack([U, V]), 'euclidean'))

    # Вычисление αU, αV, β, γ
    alpha_U = 0.5
    alpha_V = 0.5
    beta = 0
    gamma = 0.5

    # Вычисление расстояний методом полной связи
    R_W_S = alpha_U * R_U_S + alpha_V * R_V_S + beta * R_U_V + gamma *
abs(R_U_S - R_V_S)

    return R_W_S
```

Дополнение к п. 4.2 (2):

```
def hierarchical_clustering(points, num_clusters):
    points = points.values

    # Инициализируем кластеры
    clusters = [{i} for i in range(len(points))]
    distances = {}

    def calc_distance_matrix():
        for i in range(len(clusters)):
```

```
            for j in range(i + 1, len(clusters)):
                U = points[list(clusters[i])]
                V = points[list(clusters[j])]
                S = points[np.concatenate([list(clusters[i]),
list(clusters[j])])]
                dist = complete_linkage_distance(U, V, S)
                distances[(i, j)] = dist

    calc_distance_matrix()

    while len(clusters) > num_clusters:
        # Найти два ближайших кластера
        i, j = min(distances, key=distances.get)

        # Объединить их в новый кластер
        new_cluster = clusters[i].union(clusters[j])

        # Обновить список кластеров
        clusters = [clusters[k] for k in range(len(clusters)) if k != i and k
!= j]
        clusters.append(new_cluster)

        # Пересчитать расстояния
        distances = {}
        calc_distance_matrix()

    cluster_labels = np.zeros(len(points), dtype=int)
    for cluster_id, cluster in enumerate(clusters):
        for index in cluster:
            cluster_labels[index] = cluster_id

    return cluster_labels
```

Дополнение к п. 5.1(1):

```
def generate_points_inside_circle(center_x, center_y, radius, num_points):
    r = radius* np.sqrt(np.random.rand(num_points))
    theta = np.random.rand(num_points) * 2 * np.pi
    x = center_x + r* np.cos (theta)
    y = center_y + r * np.sin(theta)
    return x, y

N = 100
X = []
Y = []
for i in range(N):
    X.append(random. uniform (-10, 10))
    Y.append(random. uniform (-10, 10))
X = np.array(X)
Y = np.array(Y)
centersX = [-5, 7, 0]
centersY= [5, 2, -5]
for i in range(3): if i == 0:
    x1, y1 = generate_points_inside_circle(-5, 5, 7, 50) if i == 1:
    x2, y2 = generate_points_inside_circle(7, 2, 7, if i == 2:
    50)
    x3, y3 = generate_points_inside_circle(0, -5, 7, 50)

noiseX = []
noiseY = []
```

```
for i in range(30):
    noiseX.append(random. uniform (-15, 15))
    noisey.append(random. uniform (-15, 15))

X_tmp = np.concatenate((x1, x2, x3, noiseX), axis=0)
Y_tmp = np.concatenate((y1, y2, y3, noisey), axis=0)
data = pd.DataFrame({"x":X_tmp,"y":Y_tmp})
```

Дополнение к п. 5.1(2):

```
pointsForKmeans = data.copy()
pointsForHier = data.copy()

k = 3

luster_num, center = k_means(X = pointsForKmeans.values, k = k, iteration =
1000)
pointsForKmeans["clusterKmeans"] = cluster_num

cluster_labels = hierarchical_clustering(pointsForHier, k)
pointsForHier["clusterHier"] = cluster_labels
```

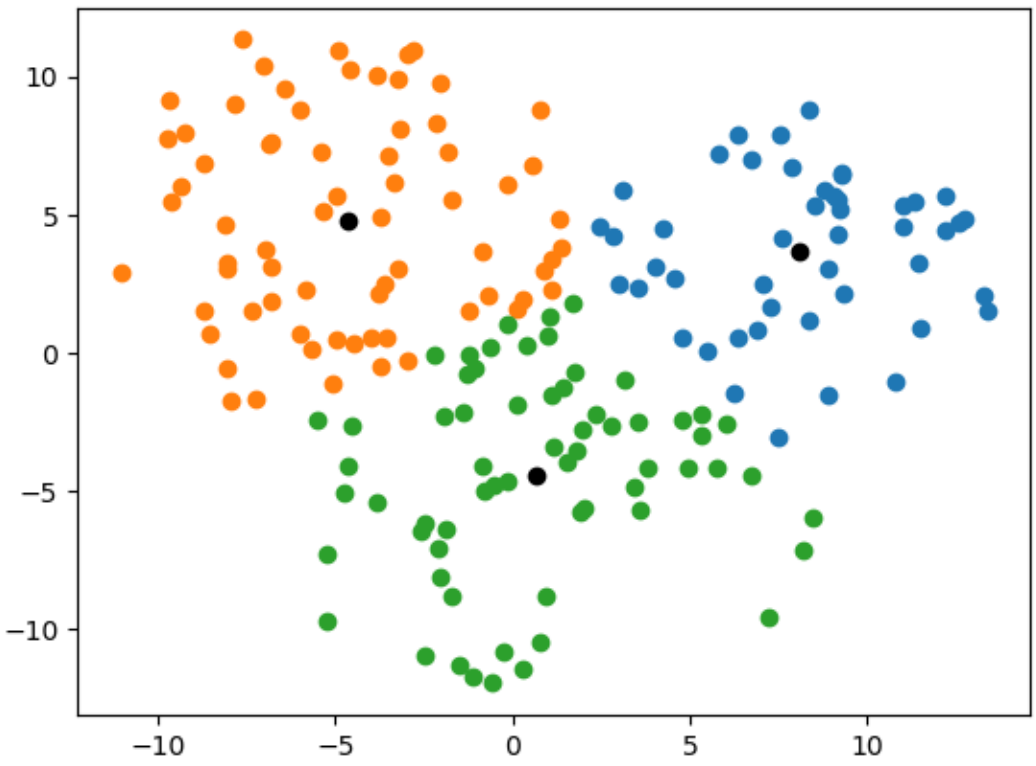
Дополнение к п. 5.2:

```
def generate_spiral1(num_points, noise=0.5):
    n = np.sqrt(np.random.rand(num_points)) * 780 * (2 * np.pi) / 360
    x = -np.cos(n) * n + np.random.rand(num_points) * noise
    y = np.sin(n) * n + np.random.rand(num_points) * noise
    return x, y

def generate_spiral2(num_points, noise=0.5):
    n = np.sqrt(np.random.rand(num_points)) * 780 * (2 * np.pi) / 360
    x = np.cos(n) * n + np.random.rand(num_points) * noise
    y = -np.sin(n) * n + np.random.rand(num_points) * noise
    return x, y

# Генерация данных для двух спиралей
num_points = 750
x1, y1 = generate_spiral1(num_points)
x2, y2 = generate_spiral2(num_points, noise=0.5)

X_tmp = np.concatenate((x1, x2), axis=0)
Y_tmp = np.concatenate((y1, y2), axis=0)
data = pd.DataFrame({"x":X_tmp,
                     "y":Y_tmp})
```

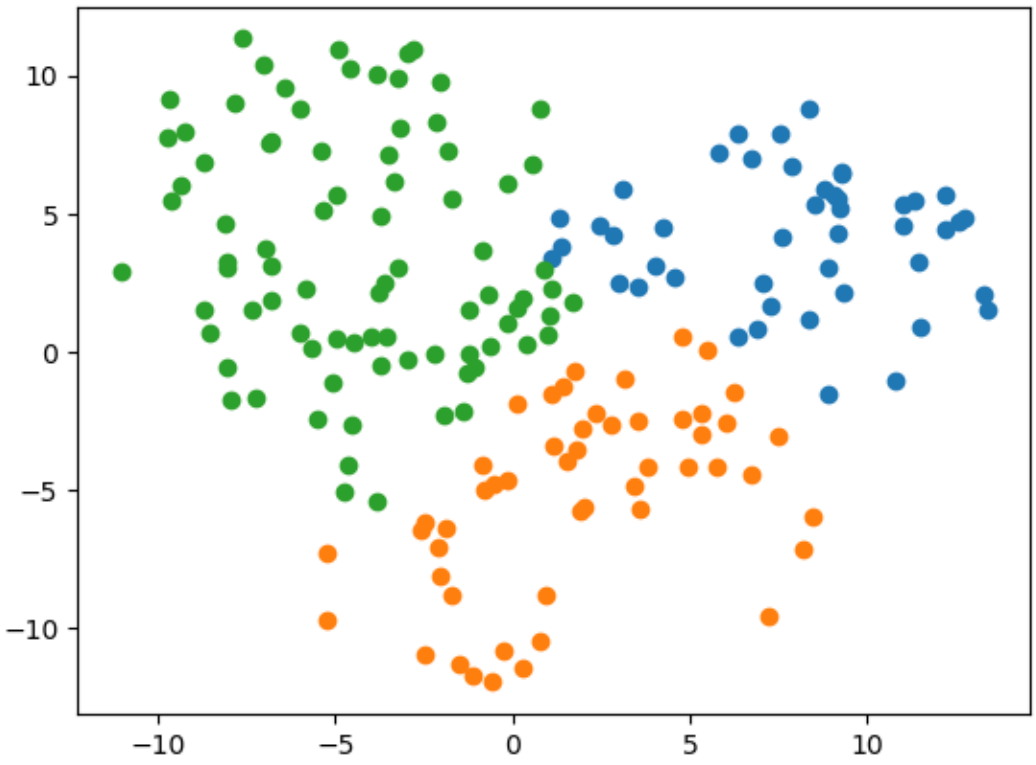
Дополнение к п. 5.3:

```
def generate_half_circle(center_x, center_y, radius, num_points, start_angle,
end_angle, noise=0.05):
    angles = np.linspace(start_angle, end_angle, num_points)
    x = center_x + radius * np.cos(angles) + np.random.normal(0, noise,
num_points)
    y = center_y + radius * np.sin(angles) + np.random.normal(0, noise,
num_points)
    return x, y

def generate_circle(center_x, center_y, radius, num_points, noise=0.05):
    angles = np.linspace(0, 2 * np.pi, num_points)
    x = center_x + radius * np.cos(angles) + np.random.normal(0, noise,
```

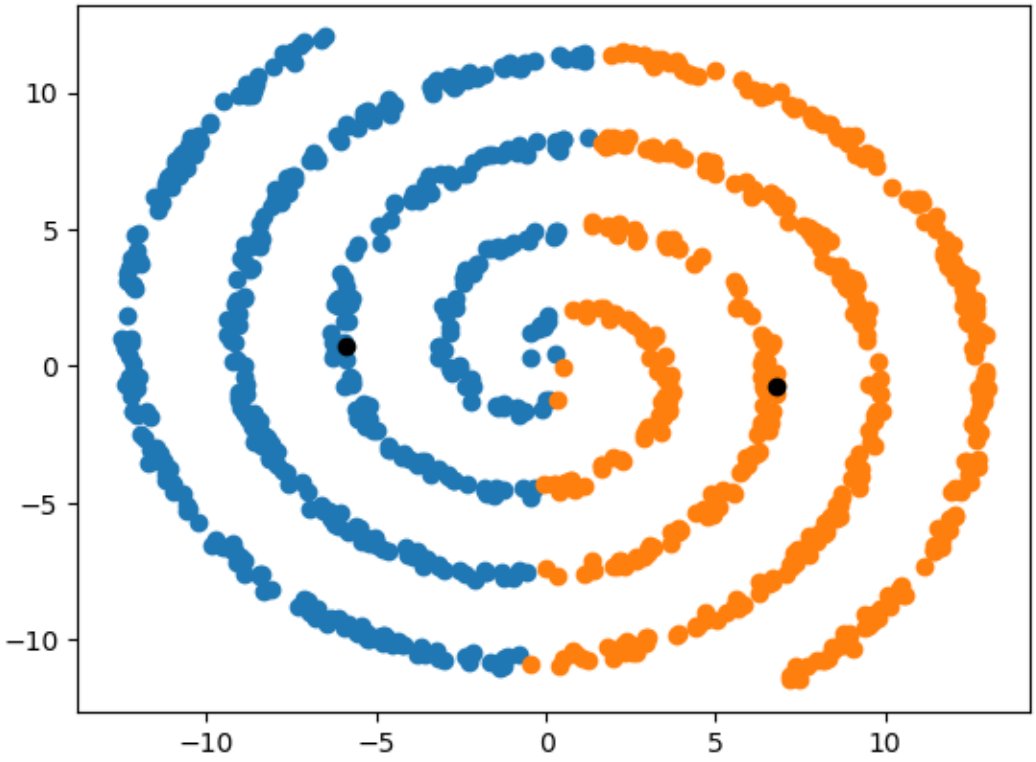
```
num_points)
    y = center_y + radius * np.sin(angles) + np.random.normal(0, noise,
num_points)
    return x, y

# Количесвто данных
num_points = 300

# Верхний полукруг
x1, y1 = generate_half_circle(0, 0.5, 1, num_points, 0, np.pi)

# Нижний полукруг
x2, y2 = generate_half_circle(0, -0.5, 1, num_points, np.pi, 2 * np.pi)

# Маленький круг внутри верхнего полукруга
x3, y3 = generate_circle(0, 0.5, 0.2, num_points // 3)

# Маленький круг внутри нижнего полукруга
x4, y4 = generate_circle(0, -0.5, 0.2, num_points // 3)

# Объединяем данные в один массив
X_tmp = np.concatenate((x1, x2, x3, x4))
Y_tmp = np.concatenate((y1, y2, y3, y4))
data = pd.DataFrame({"x":X_tmp,
                     "y":Y_tmp})
```



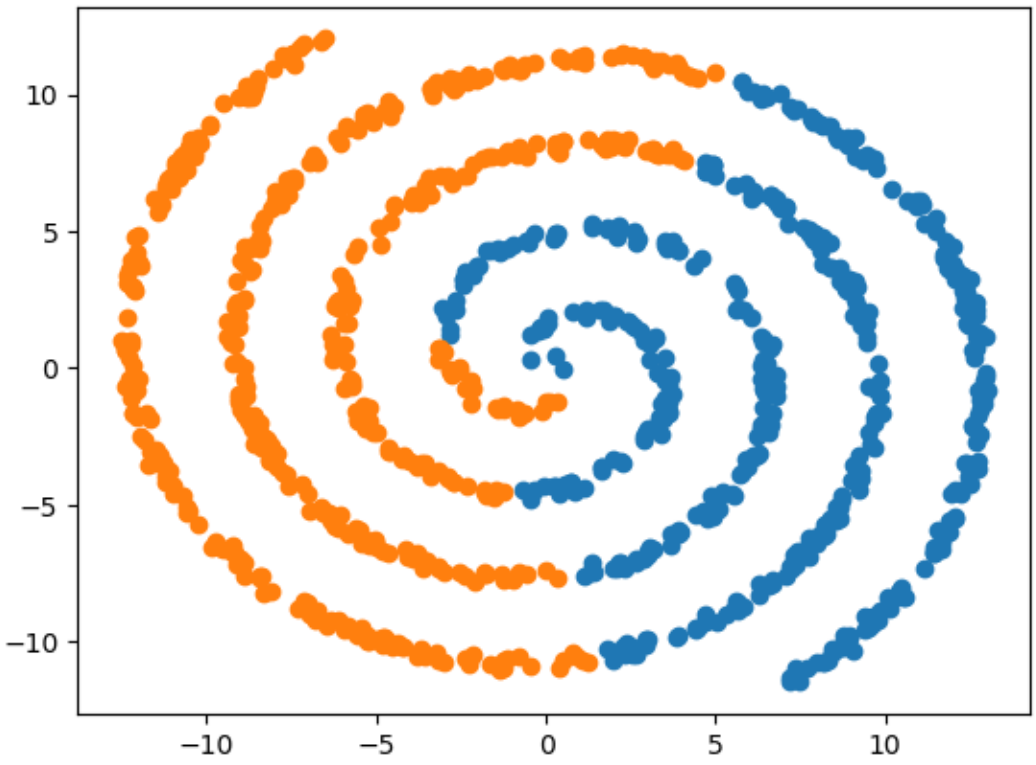
Приложение 1.1



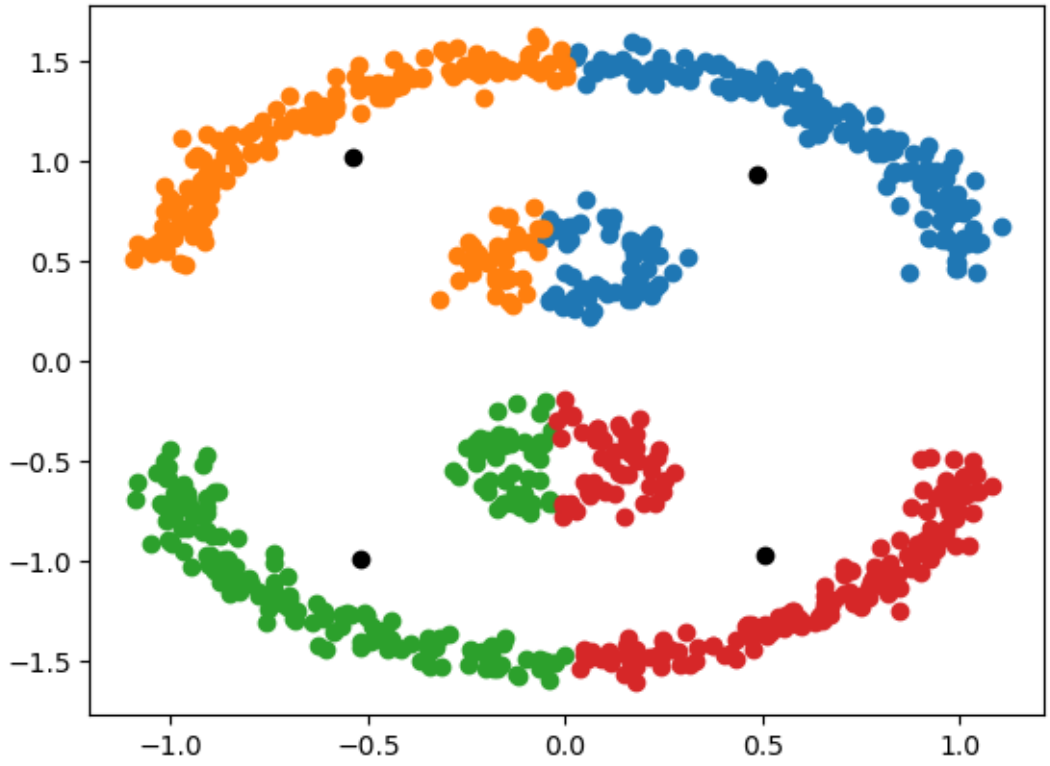
Приложение 1.2



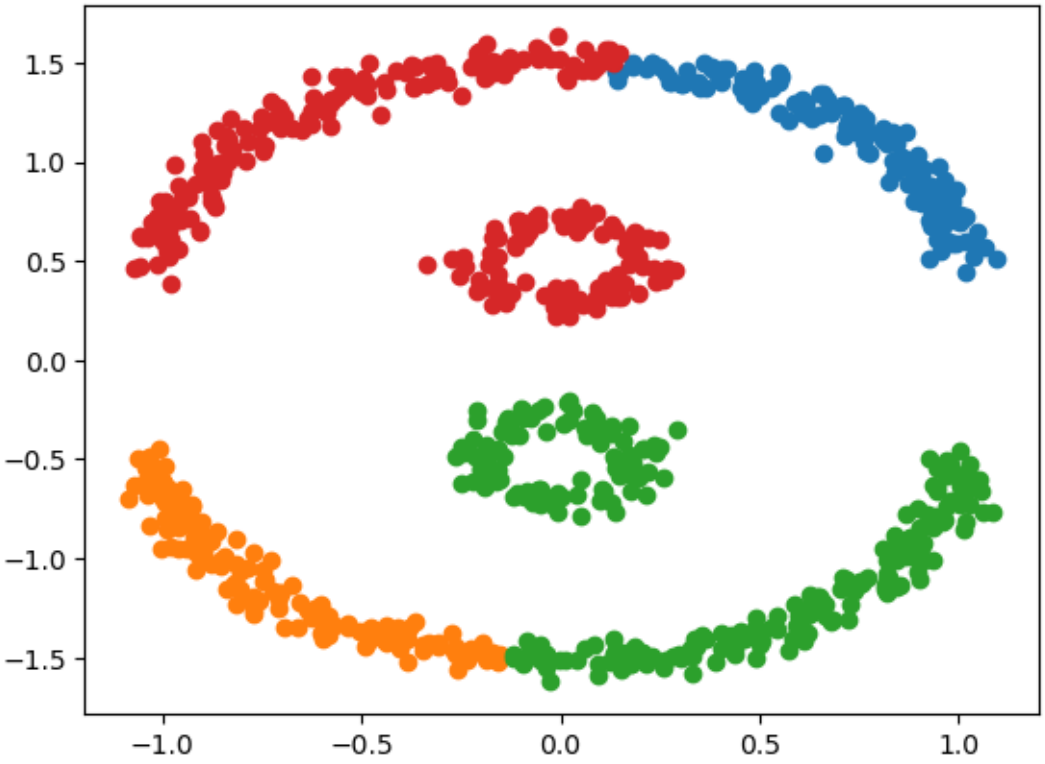
Приложение 2.1



Приложение 2.2



Приложение 3.1



Приложение 3.2

ЛИТЕРАТУРА

- Джоши, Пратик. «Искусственный интеллект с примерами на Python.» : Пер. с англ. – СПб. : ООО «Диалектика», 2019. – 448 с. – Парал. тит. англ.
- Машинное обучение. Кластеризация и частичное обучение. К.В. Воронцов, Школа анализа данных, Яндекс. https://www.youtube.com/watch?v=UTm2-G3VeQQ&list=PLJOzdkh8T5krxc4HsHbB8g8f0hu7973fK&index=13
- Преимущества применения кластерного подхода в целях развития экономики региона – тема научной статьи по экономике и бизнесу читайте бесплатно текст научно-исследовательской работы в электронной библиотеке КиберЛенинка (cyberleninka.ru)
- Применение метода Уорда для кластеризации пикселей цифрового изображения – тема научной статьи по компьютерным и информационным наукам читайте бесплатно текст научно-исследовательской работы в электронной библиотеке КиберЛенинка (cyberleninka.ru)