КЛАСТЕРИЗАЦИЯ ДАННЫХ

Методические указания
к выполнению практических работ
для студентов 2-го курса магистратуры, обучающихся по образовательным программам
направления 09.04.03 «Прикладная информатика» всех форм обучения

Воронеж 2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«Воронежский государственный технический университет»

Кафедра инноватики и строительной физики

КЛАСТЕРИЗАЦИЯ ДАННЫХ

Методические указания
к выполнению практических работ
для студентов 2-го курса магистратуры, обучающихся по образовательным программам
направления 09.04.03 «Прикладная информатика» всех форм обучения

Воронеж 2021

УДК 519.85(07)
ББК 22.18.я73

Составители:
П. А. Головинский, А. О. Шаталова, Еникеев Э.И.

КЛАСТЕРИЗАЦИЯ ДАННЫХ: методические указания к выполнению практических работ по дисциплине «Машинное обучение с Python» для студентов 2-го курса магистратуры, обучающихся по образовательной программе «Технологии искусственного интеллекта» направления 09.04.03 «Прикладная информатика» /ФГБОУ ВО «Воронежский государственный технический университет»; сост.: П.А. Головинский, А.О. Шаталова, Э.И. Еникеев. - Воронеж, 2021. – 23 с.

Содержат комплекс требований к содержанию, порядку выполнения и оформлению практических работ студентов магистрантов в соответствии с перечнем общекультурных, общепрофессиональных и профессиональных компетенций, осваиваемых в процессе обучения. Описана структура индивидуальных заданий для самостоятельного выполнения по дисциплине «Машинное обучение».

Предназначены для студентов, обучающихся по образовательной программе «Технологии искусственного интеллекта» направления 09.04.03 «Прикладная информатика» всех форм обучения.

Ил. 6. Библиогр.: 10 назв.

УДК 519.85(07)
ББК 22.18.я73

Печатается по решению редакционно-издательского совета Воронежского государственного технического университета

***Рецензент** – С. А. Баркалов, д.т.н., профессор кафедры управления строительством Воронежского государственного технического университета*

Введение

Целью практических работ по дисциплине «Машинное обучение» является закрепление теоретических навыков и их практическое применение к анализу данных. В разделах 1-5 приводится краткое описание кластерного анализа структуры данных. Соответствующий теоретический материал содержится в лекциях и учебном пособии [1]. Дополнительный материал можно найти в [2-6]. В разделе 6 приводятся программы кластеризации данных на языке Python методом *k*-средних, более подробно описанные в [7, 8]. В разделе 7 приведена программа кластеризации на языке Python методом самоорганизующихся карт Кохонена [9, 10]. Задание на выполнение практической работы выдается преподавателем, ведущим данную дисциплину. Содержание задания описано в разделе 8.

Отчет по практической работе должен содержать титульный лист и основной текст, состоящий из двух разделов:

1. теоретического описания алгоритма и исходных данных;
2. практической части с результатами численной кластеризации данных и выводам.

Отчет объемом 6-7 стр. набирается в редакторе Microsoft Word шрифтом Times New Roman размером 14 pt, 1,5 межстрочным интервалом, отступом красной строки 1,25 см, верхнее и нижнее поля – 2 см, левое – 3 см, правое – 1,5 см.

Основной текст выравнивается по ширине, нумерация страниц дается внизу по центру. Нумерация страниц начинается с введения. Названия разделов, глав, пунктов и подпунктов выделяются полиграфическими средствами и выравниваются по центру, Точка в конце заголовка не ставится и переносы слов в нем не допускаются.

Практическая работа предоставляется в электронном виде для проверки и дальнейшего хранения.

1. Постановка задачи кластеризации

При первичном анализе больших объемов данных необходимо придать им боле упорядоченную форму. Этим целям служит разбиение данных на кластеры, т.е. всего множества на подмножества, в которых элементы обладают схожестью. В результате кластеризации мы получим значительно меньшее число элементов – кластеров, анализировать свойства, которых значительно легче. Подобную работу в биологии проделал Карл Линней, создав классификацию растительного и животного мира. Согласно этой классификации существа одного вида похожи между собой, но сильно отличаются от особей других видов. Изучение живых организмов в результате сведено, главным образом, к изучению отдельных их видов.

Для математического описания схожести элементов введем характеризующий каждый элемент числовой набор из n признаков $x_i, i = 1,2, \dots n$, объединенных в некоторый вектор $x = (x_1, x_2, \dots, x_n)^T$. Будем считать, что эти векторы являются элементами некоторого пространства X. Такое пространство называется пространством признаков. В некоторых случаях это может быть векторное пространство, в котором определены операции сложения векторов и умножения вектора на число, но эти операции не всегда могут быть определены. Например, признак может быть качественным и характеризующее его число – просто целое число, нумерующее этот признак. В таком случае, сложение этих чисел будет лишено какого-либо смысла, и его нельзя определить. Введем в пространстве признаков понятие метрики, как некоторое обобщение понятия расстояния между точками. Для элементов x, y, z метрика должна обладать свойствами расстояния

$$\rho(x, y) \geq 0, \rho(x, x) = 0, \rho(x, y) \leq \rho(x, z) + \rho(z, y). \quad (1)$$

Первое из этих соотношений означает положительность расстояния. Второе уравнение указывает, что расстояние от элемента до самого себя равно нулю. Третье соотношение представляет собой, так называемое, неравенство треугольника. Оно означает, что расстояние между двумя точками короче расстояния при обходе через третью точку (сумма длин двух сторон в треугольнике больше длины третье стороны). В частном случае евклидова пространства расстояние

$$\rho(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots (x_n - y_n)^2}. \quad (2)$$

Выбор метрики в виде (2) кажется естественным только в силу схожести с теоремой Пифагора в евклидовой геометрии. На самом деле, целесообразность того или иного выбора метрики зависит от решаемой задачи.

2. Стандартизация данных

Отметим, что признаки могут иметь различную природу и соответственно этому различную размерность, например, данные о человеке могут содержать его рост в сантиметрах и вес в килограммах и уровень образования, который вообще не является числом, а относится к так называемым символьным данным, которые можно кодировать числами. Операции сравнения или сложения величин, измеряемых в разных единицах невозможны, поскольку изменение масштаба изменит результат. Изменяя единицы измерения одного из признаков, мы сразу изменим и расстояние между элементами в пространстве X. Чтобы исключить влияние выбора масштаба по отдельному признаку на расстояние между элементами, нужно привести все признаки к единому масштабу с помощью процедуры стандартизации. В качестве естественной единицы измерения признака x_i удобно выбрать, например с помощью его выборочной дисперсии σ_i^2 по всему набору N данных $\{x_i^{(k)}\}$:

$$\sigma_i = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(x_i^{(k)} - m_i)^2}. \quad (3)$$

Здесь m_i является средним значением i-го признака

$$m_i = \frac{1}{N}\sum_{k=1}^{N} x_i^{(k)}. \quad (4)$$

В результате, после нормировки получим величины $\tilde{x}_i^{(k)} = x_i^{(k)}/\sigma_i$. В качестве единицы измерения можно использовать и максимальное по модулю значение параметра в наборе данных.

3. Метрики

Кроме широко распространенной метрики Евклида, нередко используют и другие метрики:

1) Манхеттэнскую метрику (расстояние между городскими кварталами, (т.е. при разбиении на прямоугольные блоки)

$$\rho(x, y) = |x_1 - y_1| + |x_2 - y_2| + \cdots |x_n - y_n|. \quad (5)$$

В случае биполярных векторов $x_i \in \{-1,1\}$ получается так называемая метрика Хэмминга, которая равна удвоенному числу несовпадающих признаков.

2) Равномерную метрику

$$\rho(x, y) = \max_{1 \le i \le N} |x_i - y_i|. \quad (6)$$

3) Метрику Минковского

$$\rho(x,y) = \sqrt{(x_1 - y_1)^p + (x_2 - y_2)^p + \cdots (x_n - y_n)^p},\ p \geq 1. \qquad (7)$$

Метрика Минковского позволяет регулировать степень удаленность элементов друг от друга по сравнению с евклидовой метрикой.

4) Метрику Махаланобиса

$$\rho(x,y) = \sqrt{(x-y)S^{-1}(x-y)} = \qquad (8)$$
$$= \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - y_i)S_{ij}^{-1}(x_j - y_j)}$$

где (S_{ij}) – ковариационная матрица данных. Эта метрика позволяет учитывать распределение элементов и их взаимозависимость.

5) Метрику Канберра

$$\rho(x,y) = \sum_{i=1}^{n}\frac{|x_i - y_i|}{|x_i| + |y_i|}. \qquad (9)$$

Данная метрика определена, если хотя бы один из двух элементов отличен от нуля.

Можно убедиться, что все перечисленные способы задания метрики удовлетворяет необходимым свойствам метрики, т.е. метрики положительны, выполняется правило треугольника и расстояние между тождественными объектами равно нулю. Очевидно, что выбор метрики может существенно отразиться на процедуре и результатах кластеризации. Например, находящиеся на одном расстоянии по евклидовой метрике точки могут быть на разном расстоянии по манхэттенской метрике, поскольку при одинаковой длине гипотенузы у прямоугольных треугольников могут быть совсем разные катеты и разная сумма их длин. Ясно, что при решении задачи кластеризации желательно выбирать метрику, наиболее точно соответствующую характеру анализируемых данных. В случае если такой выбор заранее неизвестен или неясен, обычно используют евклидову метрику.

Дальнейшее распределение элементов по классам предполагает разделение на кластеры и задание способа определения принадлежности элемента к тому или иному кластеру.

4. Определение расстояния между вектором-образом элемента и классом

Если мы сможем определить центр кластера $c^{(k)}$, соответствующего данному классу k, то в качестве естественной меры расстояния от данного

образа x до класса A_k можно выбрать расстояние до центра кластера $\rho(x, c^{(k)})$ в соответствии с выбранной для классификации метрикой. Центр $c^{(k)}$ класса A_k можно определить как среднее арифметическое по N_k точкам кластера:

$$c^{(k)} = \frac{1}{N_k} \sum_{x \in A_k} x. \tag{10}$$

Принадлежность к кластеру A_i определяется условием ближайшего центра кластера

$$\rho(x, A_i) = \min_k \rho(x, c^{(k)}) \tag{11}$$

Другим простым способом определения расстояния от пробного элемента (образа) до класса является метод ближайшего соседа из кластера, т.е. находят ближайший элемент из всех, и пробный элемент относят к его кластеру

$$\rho(x, A_i) = \min_k \rho(x, x_k). \tag{12}$$

В ряде случаев класс плохо описывается одним эталонным образом в виде центра кластера, например, при сильной вытянутости или разветвленности (изрезанности) кластера. В этом случае его можно характеризовать несколькими эталонными образами и вычислять расстояние между предъявляемым образом x и классом, как расстояние до ближайшего эталонного образа данного класса.

5. Алгоритмы кластеризации

Кластеризация проводится с целью разделения всего множества элементов на группы похожих между собой. Задача кластеризации элементов обладает значительной неопределенностью в своей постановке. Во-первых, необходимо определить параметры, характеризующие элементы и задать метрику. Во-вторых, результат кластеризации зависит от числа задаваемых кластеров, которое чаще всего нужно определить заранее.

Рассмотрим некоторые из основных алгоритмов классификации.

А. Алгоритм *k*-групповых средних.

Этот алгоритм является итерационным. В методе *k*-средних сначала случайным образом выбираются центры $c^{(k)}$ для k кластеров. Далее согласно близости к одному из заданных таким образом центров все данные группируются в кластеры. Далее за следующий центр кластера принимается среднее значение координат элементов из данного кластера. Затем процедура повторяется, и состав кластеров может измениться. Итерации производятся до сходимости алгоритма, т.е. до тех пор, пока состав кластеров не перестает изменяться. Итерации быстро сходятся. Но результат зависит от инициации.

Вариантом решения этой проблемы является многократный прогон процедуры с тем, чтобы выбрать из полученных результатов наилучший, например, с минимальной суммой дисперсий σ каждом из кластеров

$$\sigma = \sum \sigma_i. \tag{13}$$

Рассмотренный нами алгоритм относится к семейству так называемых методов жесткой кластеризации, когда каждый образец из набора предназначен строго одному кластеру.

B. Алгоритмы мягкой (нечеткой) кластеризации.

В отличие от алгоритмов жесткой кластеризации, алгоритмы мягкой кластеризации (fuzzy clustering) назначают образец в той или иной степени нескольким кластерам. Степенью принадлежности можно управлять с помощью коэффициента нечеткости. Такой подход позволяет сформулировать алгоритм нечетких *k*-средних, или, иначе алгоритм *c*-средних.

C. Иерархическая кластеризация.

Среди алгоритмов иерархической кластеризации выделяют восходящие и нисходящие алгоритмы. В нисходящих алгоритмах вначале все объекты помещаются в один кластер. Затем он разбивается на все более мелкие кластеры. Восходящие алгоритмы в начале работы помещают каждый объект в отдельный кластер, а затем объединяют кластеры во все более крупные, пока все объекты выборки не будут содержаться в одном кластере. Таким образом, строится система вложенных разбиений. Результаты работы таких алгоритмов принято представлять в виде дерева – так называемой дендрограммы. Она показывает детализацию разбиения данных на все большее количество кластеров.

D. Самоорганизующиеся карты Кохонена.

Карты Кохонена дают простой и высокоэффективный метод кластеризации при заданном количестве кластеров. Сначала производится инициализация сети путем выбора случайных векторов в пространстве признаков, или выбора случайным образом *k*- векторов из набора данных. Далее случайным образом предъявляется один из векторов массива данных, и ближайший вектор сети (центр кластера) смещается в его сторону. Это смещение тем меньше, чем боле расстояние между вектором данных и центром кластера. Убывание смещения с расстоянием чаще всего выбирают в форме функции Гаусса. Далее процесс повторяется, а шаги постепенно уменьшаются. Полученную карту удобнее всего отображать на двумерной картине узлов.

E. Гравитационная кластеризация.

Алгоритм карт Кохонена можно рассматривать как физическое притяжение пробных частиц (центров кластеров) к ближайшим частицам среды. Другим вариантом физической аналогии является алгоритм гравитационной кластеризации. Согласно этому алгоритму точки данных притягиваются друг другу с «силой», убывающей с расстоянием, подобно

закону всемирного тяготения Ньютона. Движение под действием такой «силы» будет приводить к слипанию частиц и автоматической кластеризации данных.

Перечисленные нами методы кластеризации не учитывают более глубокие свойства данных, например их симметрию, позволяющую установить сходство между внешне разными объектами. Кроме того, кластеризация данных возможна не всегда. Выбранные признаки могут оказаться таковы, что они не позволят различить элементы, относящиеся к разным группам. Возможно также, что различия между группами элементов могут быть слишком малы для уверенного разделения их на классы.

6. Программа кластеризации методом k-средних [7,8]

Программа кластеризации методам *k*-средних состоит из трех шагов:

1. Инициализация
2. Присвоение
3. Обновление

При работе потребуется использовать пакет pandas

https://benalexkeen.com/k-means-clustering-in-python/

П.1. Инициализация

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#matplotlib in line

df = pd.DataFrame({
'x': [12, 20, 28, 18, 29, 33, 24, 45, 45, 52, 51, 52, 55, 53, 55, 61, 64, 69, 72],
'y': [39, 36, 30, 52, 54, 46, 55, 59, 63, 70, 66, 63, 58, 23, 14, 8, 19, 7, 24]
})
np.random.seed(200)
k = 3
# centroids[i] = [x, y]
centroids = {
i+1: [np.random.randint(0, 80), np.random.randint(0, 80)]
for i in range(k)
}

fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color='k')
colmap = {1: 'r', 2: 'g', 3: 'b'}
for i in centroids.keys():
plt.scatter(*centroids[i], color=colmap[i])
```

```
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()
```

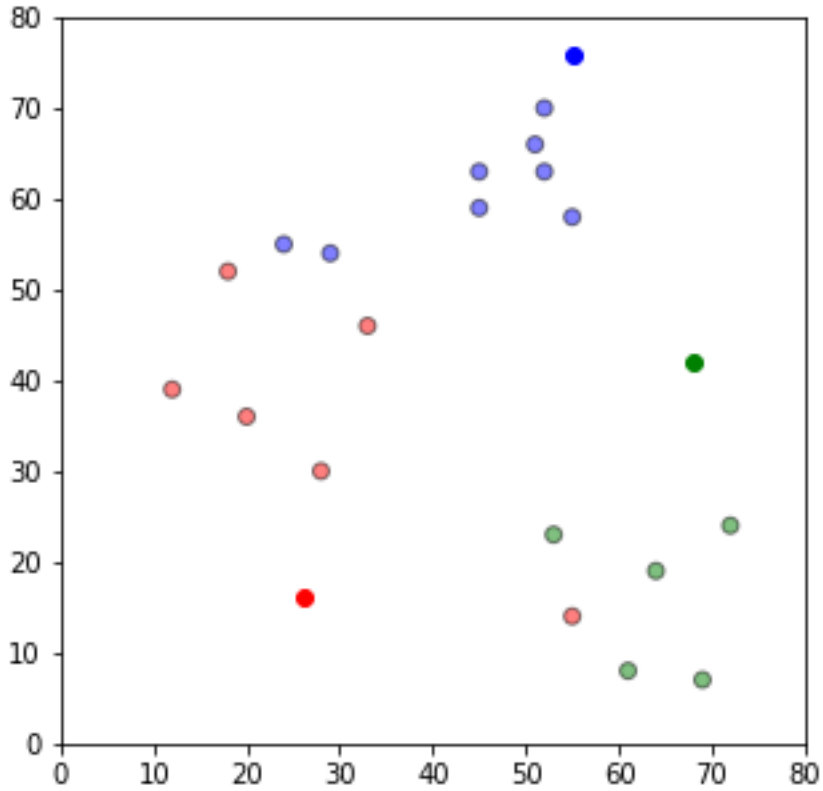

Рис.1. Исходные данные

П.2. Стадия присвоения центров кластеров и нахождения их элементов

```
df = pd.DataFrame({
    'x': [12, 20, 28, 18, 29, 33, 24, 45, 45, 52, 51, 52, 55, 53, 55, 61, 64, 69, 72],
    'y': [39, 36, 30, 52, 54, 46, 55, 59, 63, 70, 66, 63, 58, 23, 14, 8, 19, 7, 24]
})
np.random.seed(200)
k = 3
# centroids[i] = [x, y]
centroids = {
    i+1: [np.random.randint(0, 80), np.random.randint(0, 80)]
    for i in range(k)
}

fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color='k')
colmap = {1: 'r', 2: 'g', 3: 'b'}
for i in centroids.keys():
```

```
        plt.scatter(*centroids[i], color=colmap[i])
    plt.xlim(0, 80)
    plt.ylim(0, 80)
    plt.show()

    def assignment(df, centroids):
        for i in centroids.keys():

            df['distance_from_{}'.format(i)] = (
                np.sqrt(
                    (df['x'] - centroids[i][0]) ** 2
                    + (df['y'] - centroids[i][1]) ** 2
                )
            )
        centroid_distance_cols = ['distance_from_{}'.format(i) for i in
centroids.keys()]
        df['closest'] = df.loc[:, centroid_distance_cols].idxmin(axis=1)
        df['closest'] = df['closest'].map(lambda x: int(x.lstrip('distance_from_')))
        df['color'] = df['closest'].map(lambda x: colmap[x])
        return df

    df = assignment(df, centroids)
    print(df.head())

    fig = plt.figure(figsize=(5, 5))
    plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
    for i in centroids.keys():
        plt.scatter(*centroids[i], color=colmap[i])
    plt.xlim(0, 80)
    plt.ylim(0, 80)
    plt.show()
```

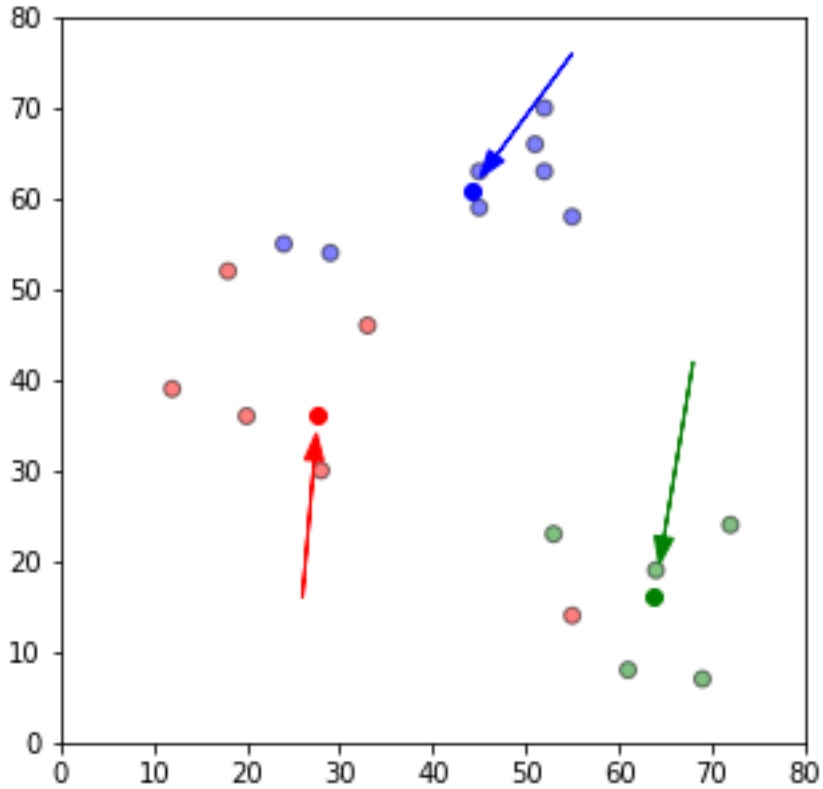

Рис. 2. Первичные центры кластеров

П.3. Стадия обновления кластеров

```
import copy
old_centroids = copy.deepcopy(centroids)

def update(k):
    for i in centroids.keys():
        centroids[i][0] = np.mean(df[df['closest'] == i]['x'])
        centroids[i][1] = np.mean(df[df['closest'] == i]['y'])
    return k

centroids = update(centroids)

fig = plt.figure(figsize=(5, 5))
ax = plt.axes()
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
for i in old_centroids.keys():
    old_x = old_centroids[i][0]
```

```
        old_y = old_centroids[i][1]
        dx = (centroids[i][0] - old_centroids[i][0]) * 0.75
        dy = (centroids[i][1] - old_centroids[i][1]) * 0.75
        ax.arrow(old_x, old_y, dx, dy, head_width=2, head_length=3, fc=colmap[i],
ec=colmap[i])
    plt.show()

    import copy

    old_centroids = copy.deepcopy(centroids)

    def update(k):
        for i in centroids.keys():
            centroids[i][0] = np.mean(df[df['closest'] == i]['x'])
            centroids[i][1] = np.mean(df[df['closest'] == i]['y'])
        return k

    centroids = update(centroids)

    fig = plt.figure(figsize=(5, 5))
    ax = plt.axes()
    plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
    for i in centroids.keys():
        plt.scatter(*centroids[i], color=colmap[i])
    plt.xlim(0, 80)
    plt.ylim(0, 80)
    for i in old_centroids.keys():
        old_x = old_centroids[i][0]
        old_y = old_centroids[i][1]
        dx = (centroids[i][0] - old_centroids[i][0]) * 0.75
        dy = (centroids[i][1] - old_centroids[i][1]) * 0.75
        ax.arrow(old_x, old_y, dx, dy, head_width=2, head_length=3, fc=colmap[i],
ec=colmap[i])
    plt.show()
```

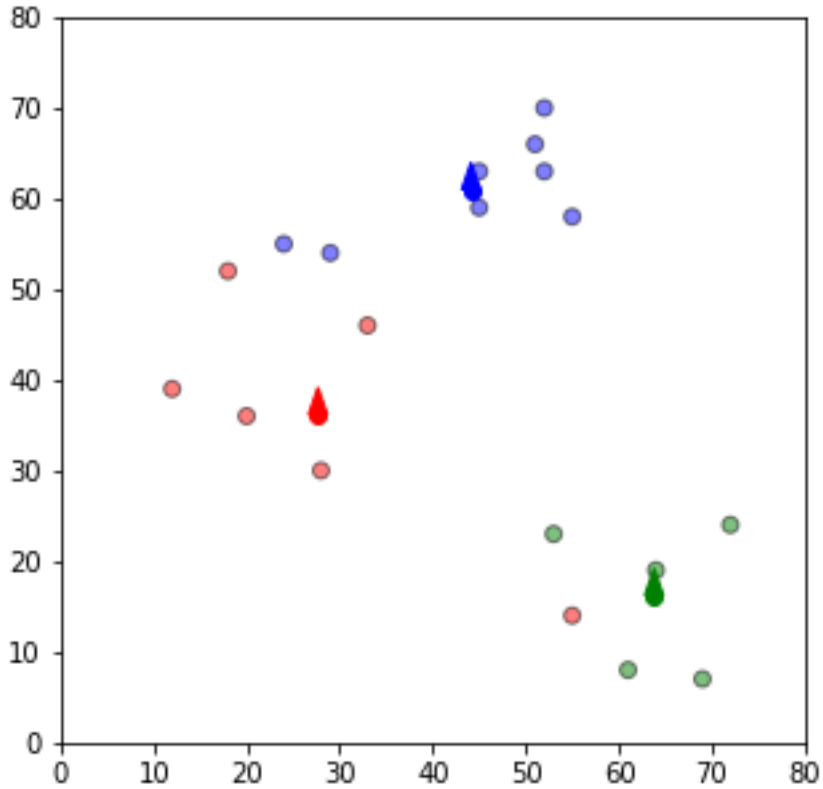

Рис. 3. Обновленные центры кластеров

П.4. Повторение присвоения

```
df = assignment(df, centroids)

# Печать результатов
fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()
while True:
    closest_centroids = df['closest'].copy(deep=True)
    centroids = update(centroids)
    df = assignment(df, centroids)
    if closest_centroids.equals(df['closest']):
        break

fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
for i in centroids.keys():
```

```
    plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()

 df = pd.DataFrame({
    'x': [12, 20, 28, 18, 29, 33, 24, 45, 45, 52, 51, 52, 55, 53, 55, 61, 64, 69, 72],
    'y': [39, 36, 30, 52, 54, 46, 55, 59, 63, 70, 66, 63, 58, 23, 14, 8, 19, 7, 24]
})
```

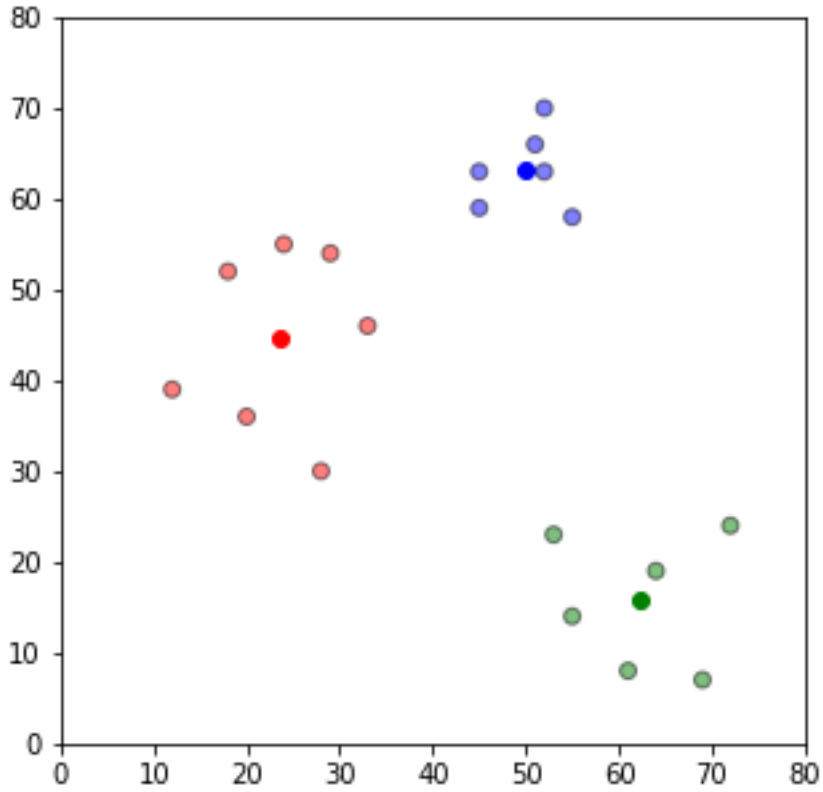

Рис. 4. Результаты кластеризации

Отчетливо видны три выделенных кластера и их центры.

Более компактный код можно получить, используя библиотеку scikit-learn. Далее приведен пример такого кода из [8].

```
# Author: Phil Roth <mr.phil.roth@gmail.com>
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
```

```
        from sklearn.datasets import make_blobs

        plt.figure(figsize=(12, 12))

        n_samples = 1500
        random_state = 170
        X, y = make_blobs(n_samples=n_samples, random_state=random_state)

        # Incorrect number of clusters
        y_pred = KMeans(n_clusters=2, random_state=random_state).fit_predict(X)

        plt.subplot(221)
        plt.scatter(X[:, 0], X[:, 1], c=y_pred)
        plt.title("Incorrect Number of Blobs")

        # Anisotropically distributed data
        transformation = [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]]
        X_aniso = np.dot(X, transformation)
        y_pred = KMeans(n_clusters=3,
random_state=random_state).fit_predict(X_aniso)

        plt.subplot(222)
        plt.scatter(X_aniso[:, 0], X_aniso[:, 1], c=y_pred)
        plt.title("Anisotropicly Distributed Blobs")

        # Different variance
        X_varied, y_varied = make_blobs(n_samples=n_samples,
                                cluster_std=[1.0, 2.5, 0.5],
                                random_state=random_state)
        y_pred = KMeans(n_clusters=3,
random_state=random_state).fit_predict(X_varied)

        plt.subplot(223)
        plt.scatter(X_varied[:, 0], X_varied[:, 1], c=y_pred)
        plt.title("Unequal Variance")

        # Unevenly sized blobs
        X_filtered = np.vstack((X[y == 0][:500], X[y == 1][:100], X[y == 2][:10]))
        y_pred = KMeans(n_clusters=3,
                    random_state=random_state).fit_predict(X_filtered)

        plt.subplot(224)
        plt.scatter(X_filtered[:, 0], X_filtered[:, 1], c=y_pred)
```

```
plt.title("Unevenly Sized Blobs")

plt.show()
```

На рис. 5 представлены результаты работы программы кластеризации для разных наборов данных.

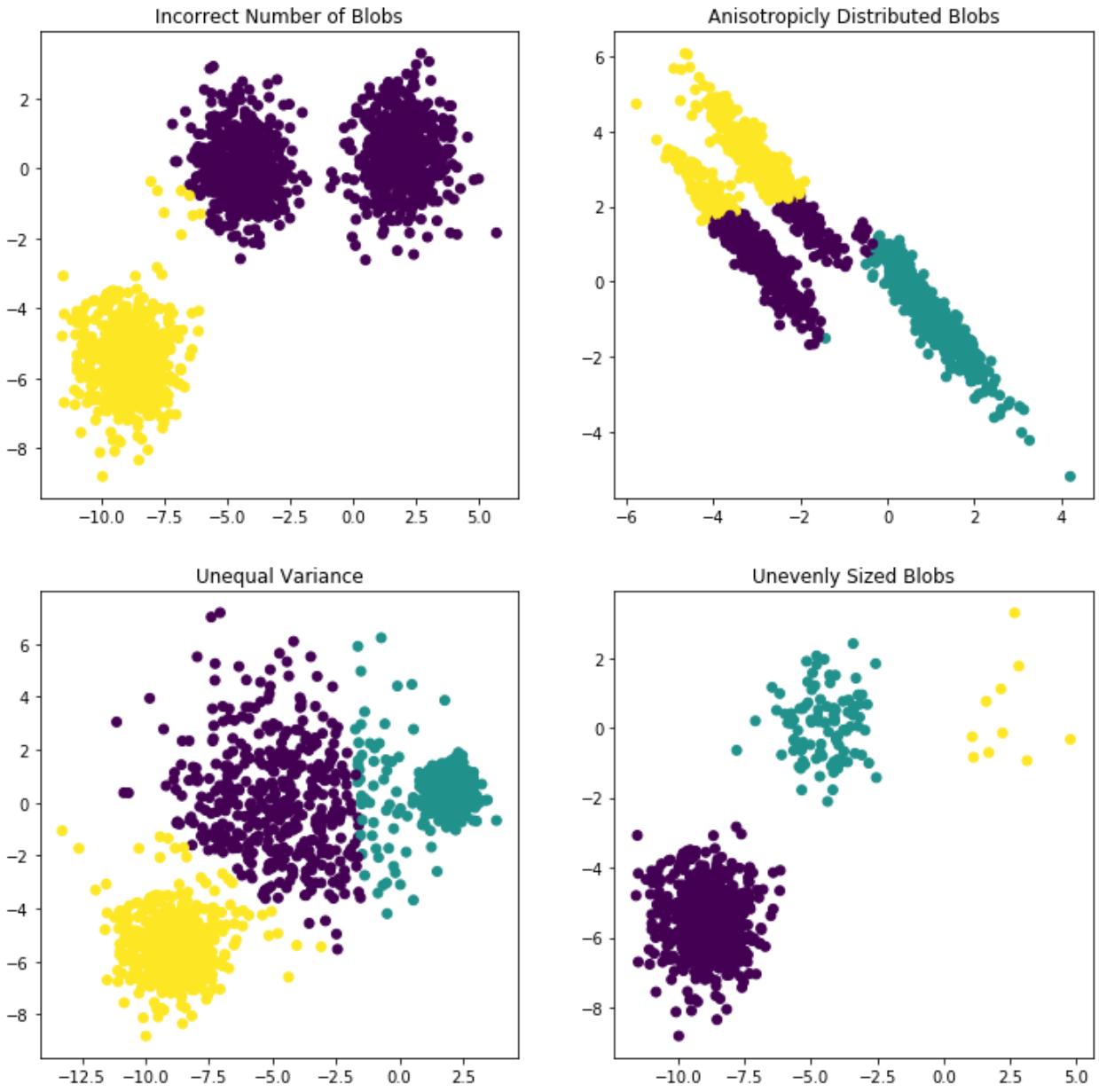

Рис. 5. Результаты кластеризации для разных типов данных

7. Программа кластеризации с помощью карт Кохонена [8,9]

```
import time
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
class SOMNetwork():
    def __init__(self, input_dim, dim=10, sigma=None, learning_rate=0.1, tay2=1000, dtype=tf.float32):
        if not sigma:
            sigma = dim / 2
        self.dtype = dtype
        #constants
        self.dim = tf.constant(dim, dtype=tf.int64)
        self.learning_rate = tf.constant(learning_rate, dtype=dtype, name='learning_rate')
        self.sigma = tf.constant(sigma, dtype=dtype, name='sigma')
        self.tay1 = tf.constant(1000/np.log(sigma), dtype=dtype, name='tay1')
        self.minsigma = tf.constant(sigma * np.exp(-1000/(1000/np.log(sigma))), dtype=dtype, name='min_sigma')
        self.tay2 = tf.constant(tay2, dtype=dtype, name='tay2')
        #input vector
        self.x = tf.placeholder(shape=[input_dim], dtype=dtype, name='input')
        #iteration number
        self.n = tf.placeholder(dtype=dtype, name='iteration')
        #variables
        self.w = tf.Variable(tf.random_uniform([dim*dim, input_dim], minval=-1, maxval=1, dtype=dtype),
            dtype=dtype, name='weights')
        #helper
        self.positions = tf.where(tf.fill([dim, dim], True))

    def feed(self, input):
        init = tf.global_variables_initializer()
        with tf.Session() as sess:
            init.run()
            win_index = sess.run(self.__competition(), feed_dict={self.x: input})
            win_index_2d = np.array([win_index//self.dim.eval(), win_index-win_index//self.dim.eval()*self.dim.eval()])
        return win_index_2d
```

```
    def training_op(self):
        win_index = self.__competition('train_')
        with tf.name_scope('cooperation') as scope:
            coop_dist = tf.sqrt(tf.reduce_sum(tf.square(tf.cast(self.positions -
                [win_index//self.dim, win_index-
win_index//self.dim*self.dim],
                dtype=self.dtype)), axis=1))
            sigma = tf.cond(self.n > 1000, lambda: self.minsigma, lambda:
self.sigma * tf.exp(-self.n/self.tay1))
            sigma_summary = tf.summary.scalar('Sigma', sigma)
            tnh = tf.exp(-tf.square(coop_dist) / (2 * tf.square(sigma))) #
topological neighbourhood
        with tf.name_scope('adaptation') as scope:
            lr = self.learning_rate * tf.exp(-self.n/self.tay2)
            minlr = tf.constant(0.01, dtype=self.dtype,
name='min_learning_rate')
            lr = tf.cond(lr <= minlr, lambda: minlr, lambda: lr)
            lr_summary = tf.summary.scalar('Learning rate', lr)
            delta = tf.transpose(lr * tnh * tf.transpose(self.x - self.w))
            training_op = tf.assign(self.w, self.w + delta)
        return training_op, lr_summary, sigma_summary

    def __competition(self, info=''):
        with tf.name_scope(info+'competition') as scope:
            distance = tf.sqrt(tf.reduce_sum(tf.square(self.x - self.w), axis=1))
        return tf.argmin(distance, axis=0)
#== Test SOM Network ==
def test_som_with_color_data():
    som_dim = 100
    som = SOMNetwork(input_dim=3, dim=som_dim, dtype=tf.float64, sigma=3)
    test_data = np.random.uniform(0, 1, (250000, 3))
    training_op, lr_summary, sigma_summary = som.training_op()
    init = tf.global_variables_initializer()
    writer = tf.summary.FileWriter('./logs/', tf.get_default_graph())
    with tf.Session() as sess:
        init.run()
        img1 = tf.reshape(som.w, [som_dim,som_dim,-1]).eval()
        plt.figure(1)
        plt.subplot(121)
        plt.imshow(img1)
        start = time.time()
        for i, color_data in enumerate(test_data):
            if i % 1000 == 0:
```

```
                print('iter:', i)
            sess.run(training_op, feed_dict={som.x: color_data, som.n:i})
        end = time.time()
        print(end - start)
        img2 = tf.reshape(som.w, [som_dim,som_dim,-1]).eval()
        plt.subplot(122)
        plt.imshow(img2)
    writer.close()
    plt.show()
test_som_with_color_data()
```

На рис. 6 приведены исходные данные (слева) и результат кластеризации по цветам (справа).

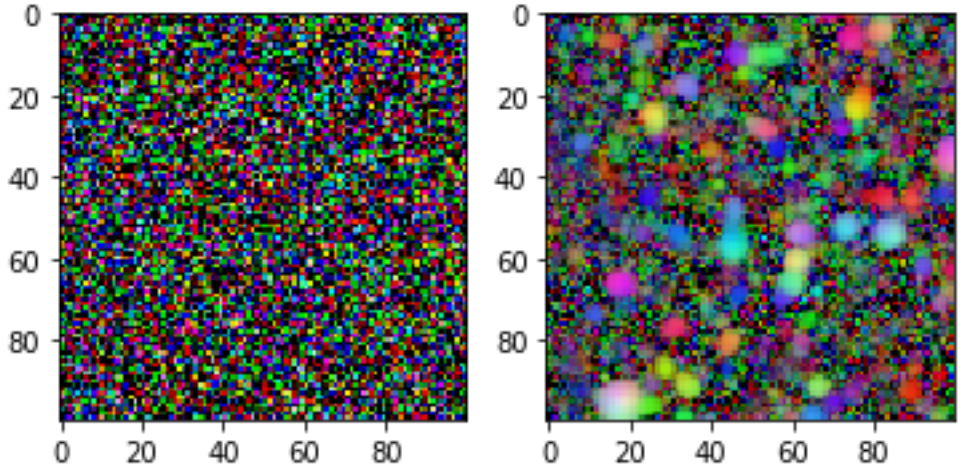

Рис. 6. Кластеризация по цветам

8. Практические задания

1. Взять лист бумаги в клетку.
2. Получить от преподавателя набор данных на плоскости для кластеризации.
3. Выбрать одну из программных реализаций кластеризации [7] или [8] методом k-средних.
4. Выполнить кластеризацию и сделать выводы.
5. Установить в ANACONDA дополнительный пакет tesorflow.
6. Выполнить программу кластеризации изображения по цветам с помощью карт Кохоннена [9, 10].
7. Создать отчет в Word о выполненной практической работе с описанием исходных данных, промежуточных результатов работы программ, полученным результатам классификации и выводами.

Библиографический список

1. К.М. Бишоп. Распознавание образов и машинное обучение. СПб: ООО Диалектика», 2020. – 960с.
2. Дж. Ту, Р. Гонсалес. Принципы распознавания образов. – М.: Мир, 1978. – 411с.
3. Т. Кохонен. Самоорганизующиеся карты. – М.: БИНОМ. Лаборатория знаний, 2008. – 655с.
4. Д. Грасс. Data Science. Наука о данных с нуля. – СПб: БХВ-Перербург, 2019. – 336с.
5. С. Рашка. Python и машинное обучение. – М.: ДМК Пресс, 2017. – 418с.
6. А.Е. Ленская, А.Г. Броневич. Математические методы распознавания образов. Таганрог, Изд.-во ТТИ, ЮФУ, 2009. –155с.
7. https://benalexkeen.com/k-means-clustering-in-python/
8. https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_assumptions.html#sphx-glr-auto-examples-cluster-plot-kmeans-assumptions-py
9. https://habr.com/ru/post/334810/
10. https://github.com/moonl1ght/Self-orginizing-Map/blob/master/SOM.py

ОГЛАВЛЕНИЕ

КЛАСТЕРИЗАЦИЯ ДАННЫХ

Методические указания
к выполнению практических работ
для студентов 2-го курса магистратуры, обучающихся по образовательным программам
направления 09.04.03 «Прикладная информатика» всех форм обучения

Составители: Головинский Павел Абрамович,
Шаталова Ангелина Олеговна,
Еникеев Эльдар Ильясович

Подписано в печать 00.00.2021 Формат 60x84 1/16. Уч.-изд. л. 0,0.
Усл. печ. л. 0.0. Бумага для множительных аппаратов. Тираж 30 экз. Заказ №___.

ФГБОУ ВО «Воронежский государственный технический университет»
394026 г. Воронеж, Московский проспект, 14

Участок оперативной полиграфии издательство ВГТУ
394026 г. Воронеж, Московский проспект, 14