МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ И СЕТЕЙ

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

доцент, к.т.н., доцент		Н.В. Соловьев
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Кластерный анализ в распознавании образов

по дисциплине: Интеллектуальные системы

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № Z0440M 11.02.2021 Митрофанов Д.А.

Санкт-Петербург 2021

Цель работы

Ознакомиться с методами кластеризации образов и расстояниями в пространстве признаков, разработать программу, выполняющую кластеризацию заданного множества образов.

1 Исходные данные

Вариант: 11

Для расчета расстояний между образами используется метод **Евклида** и **доминирования**

1) Метод Евклида

$$d_{lp} = \sqrt{\sum_{i=1}^{n} \left(x_{il} - x_{ip}\right)^2}$$

2) Метод доминирования

$$d_{lp} = \max_{i=1,n} \left(\left|x_{il} - x_{ip}\right|\right)$$

Для расчета расстояний между кластерами используется метод центров (расстояние **между центрами** кластеров) и метод **ближнего соседа**:

1) Между центрами
   Сначала находим центральную точку каждого кластера

$$x^* = \frac{1}{m_1} \sum_{i=1}^{m_1} \vec{x}_i, \;\; y^* = \frac{1}{m_2} \sum_{i=1}^{m_2} \vec{y}_i$$

   После чего находим уже само расстояние между ними любым способом.

2) Метод ближнего соседа

   Расстояние между ближайшими точками, принадлежащими разным классам w1, w2

$$d(w_1, w_2) = \min\left(d_{lp}\right), (l = 1, \dots, m_1; p = 1, \dots, m_2);$$

Кластеризация производится посредством метода **слияния**:

Сначала каждый образ считается отдельным кластером, далее вычисляется расстояние между всеми кластерами, т.е. формируется квадратная, диагонально-симметричная таблица расстояний, строки и столбцы которой – имеющиеся кластеры. На каждом шаге сливаются два самых близких кластера, после чего размер таблицы уменьшается на единицу и вычисляются новые расстояния между изменившимися кластерами. Процесс прекращается при достижении заданного числа кластеров или когда расстояние между ближайшими кластерами больше заданной пороговой величины.

В отличие от порогового и цепного алгоритмов кластеризации, алгоритм слияния может работать и без порога. Таким образом, сливать кластеры можно при помощи минимального расстояния между ними.

Так как требовалось провести анализ с введением весов, то они расчитывались как было указано в дополнительных материалах, т.е. через обратную дисперсию каждого признака (всего 3). Эти веса учитываются при расчете расстояния между образами как, например, в расстоянии Евклида.

2 Интерфейс разработанной программы

Так как программа разрабатывалась при помощи языка Python в Jupyter Notebook, то в качестве интерфейса была реализована страница с виджетами:

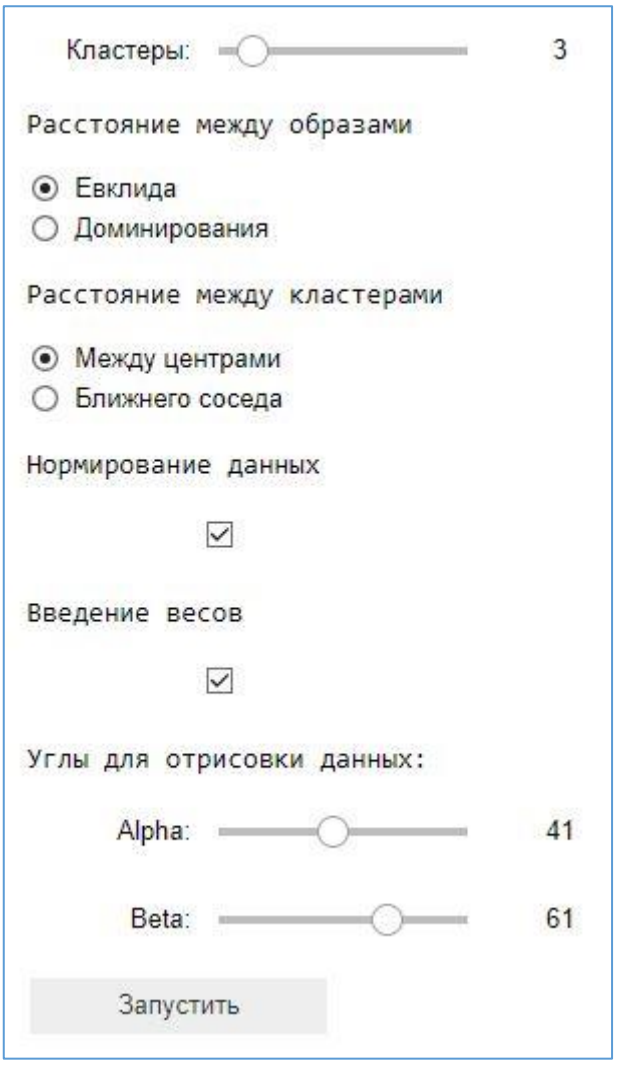

Рисунок 1. Интерфейс программы

С помощью разработанного интерфейса возможно изменить кол-во кластеров, методы для расчета расстояний как между образами, так и между кластерами, а также включить или выключить нормировку данных с учетом весов.

Помимо прочего для лучшего понимания процесса кластеризации были введены углы, позволяющие посмотреть данные под различными углами.

После нажатия кнопки “Запустить” под ней произойдет отрисовка 3D графика с данными различного цвета (кластерами).

3 Результаты экспериментов

Настройки выбирались случайным образом по возможности покрыть все варианты кластеризации.

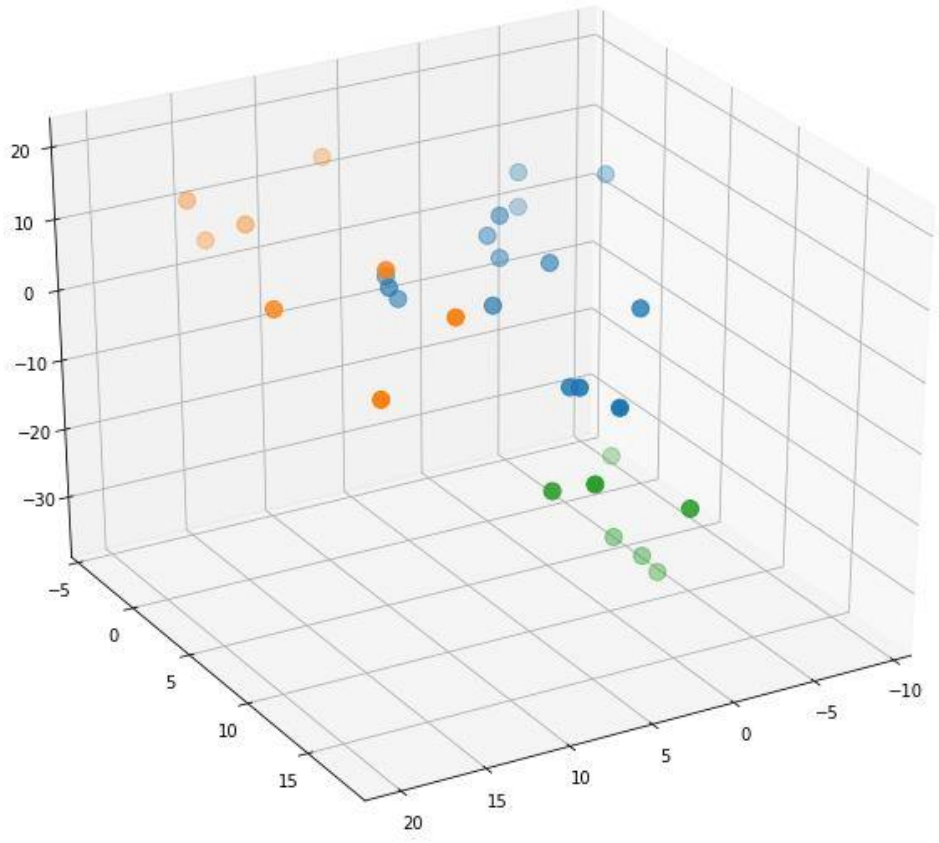

Рисунок 2. Метод Евклида, между центрами, 3–кластера

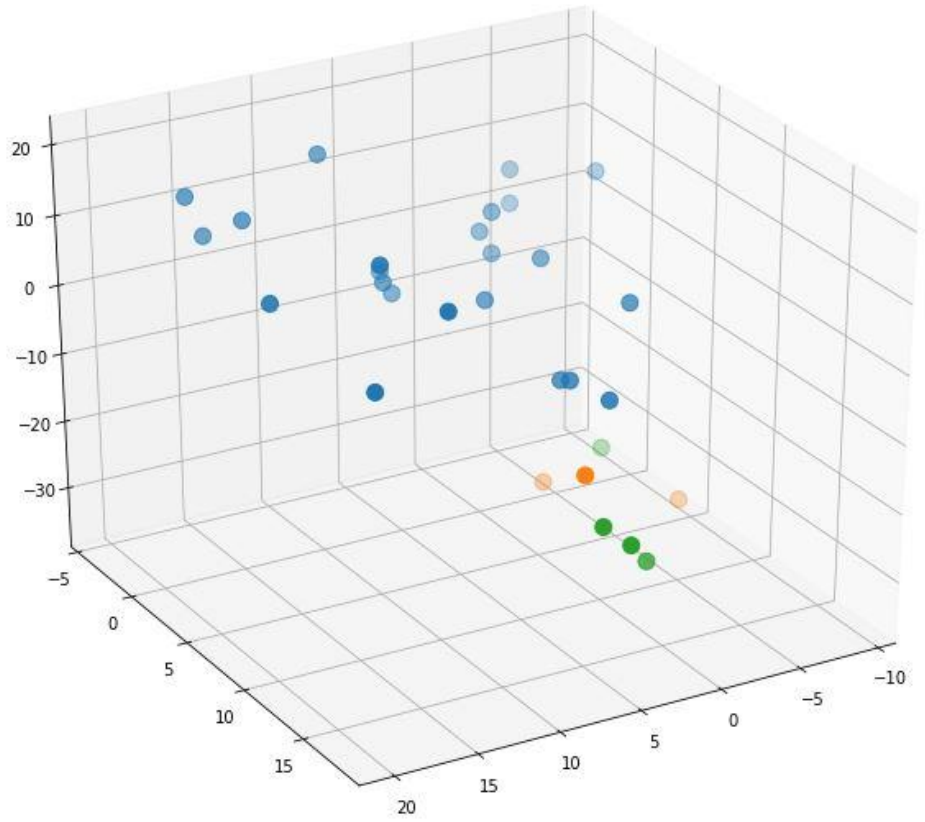

Рисунок 3. Метод доминирования, между центрами, 3-кластера

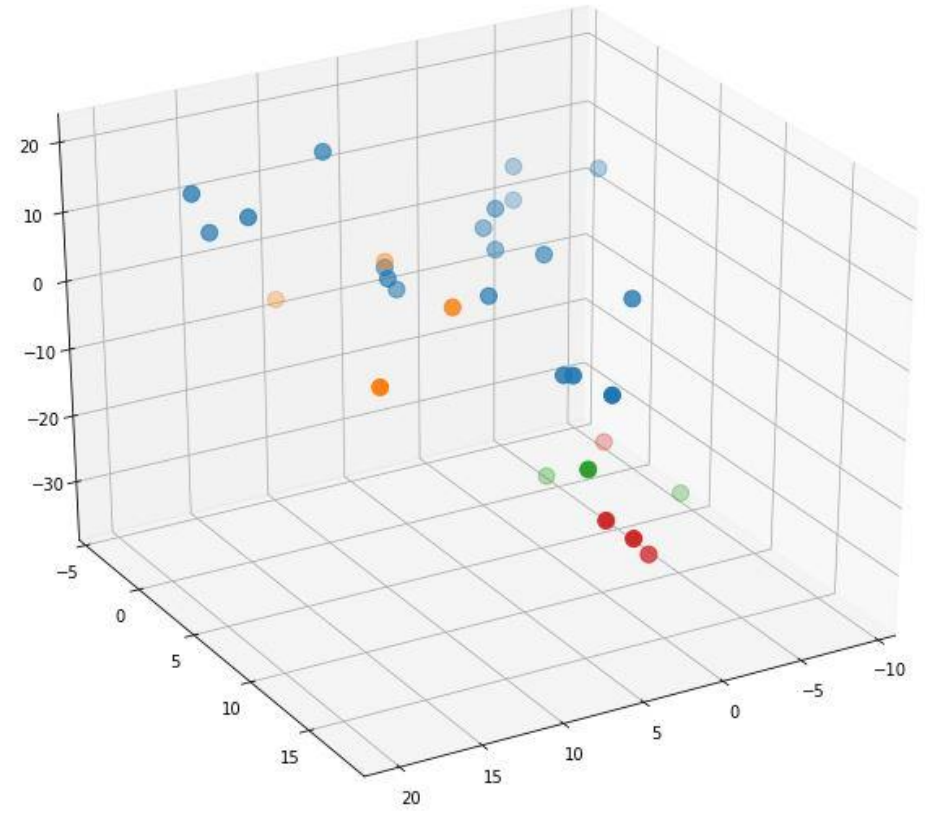

Рисунок 4. Метод доминирования, ближнего соседа, 4-кластера

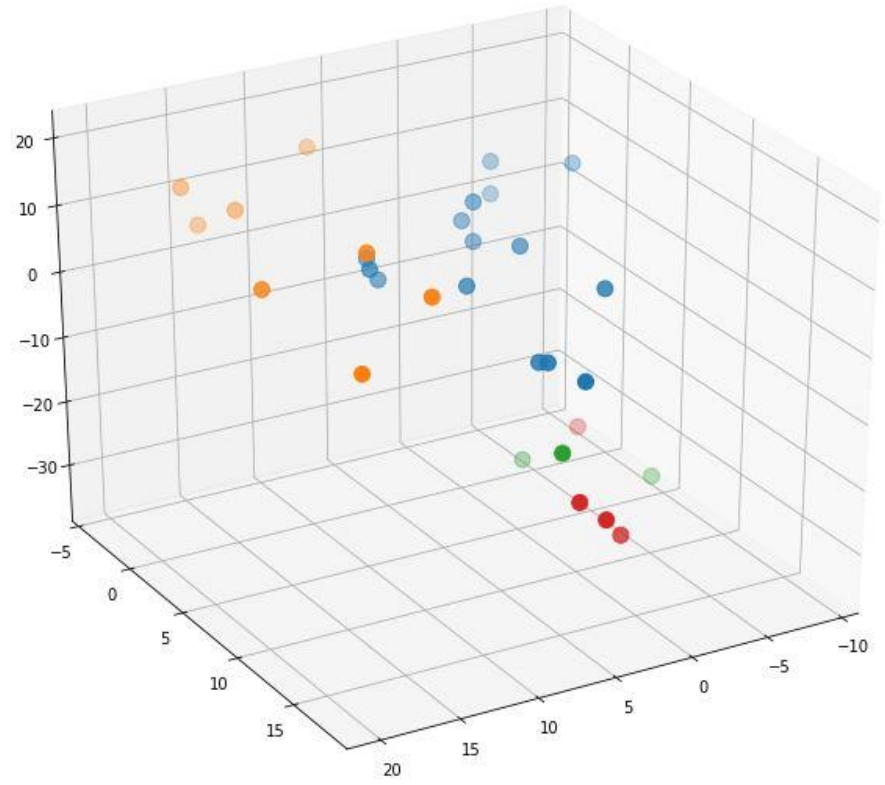

Рисунок 5. Метод Евклида, ближнего соседа, 4-кластера

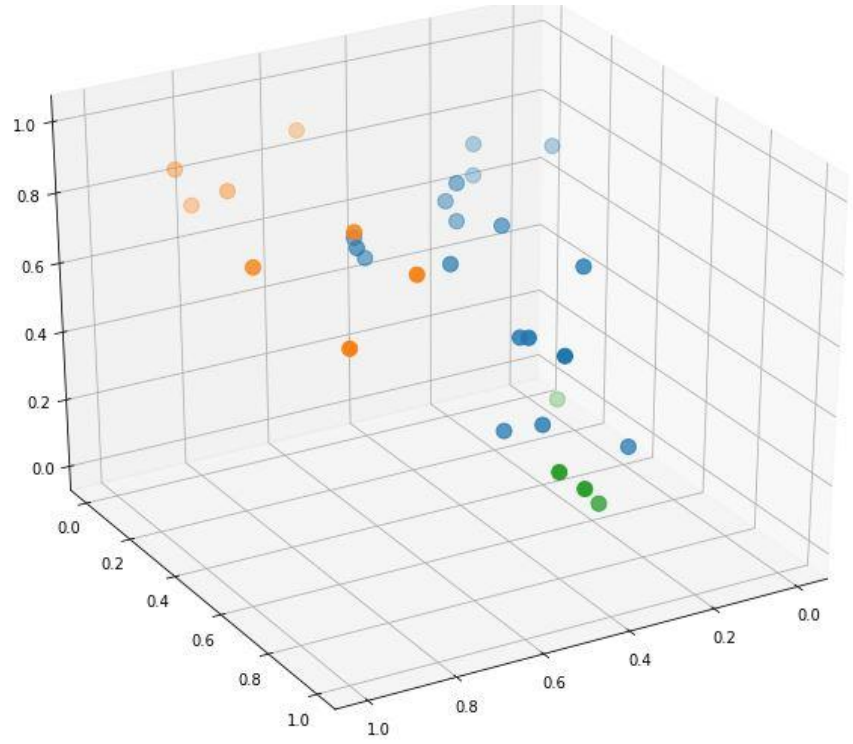

Рисунок 6. Метод Евклида, между центрами, 3-кластера, нормирование данных и учет весов

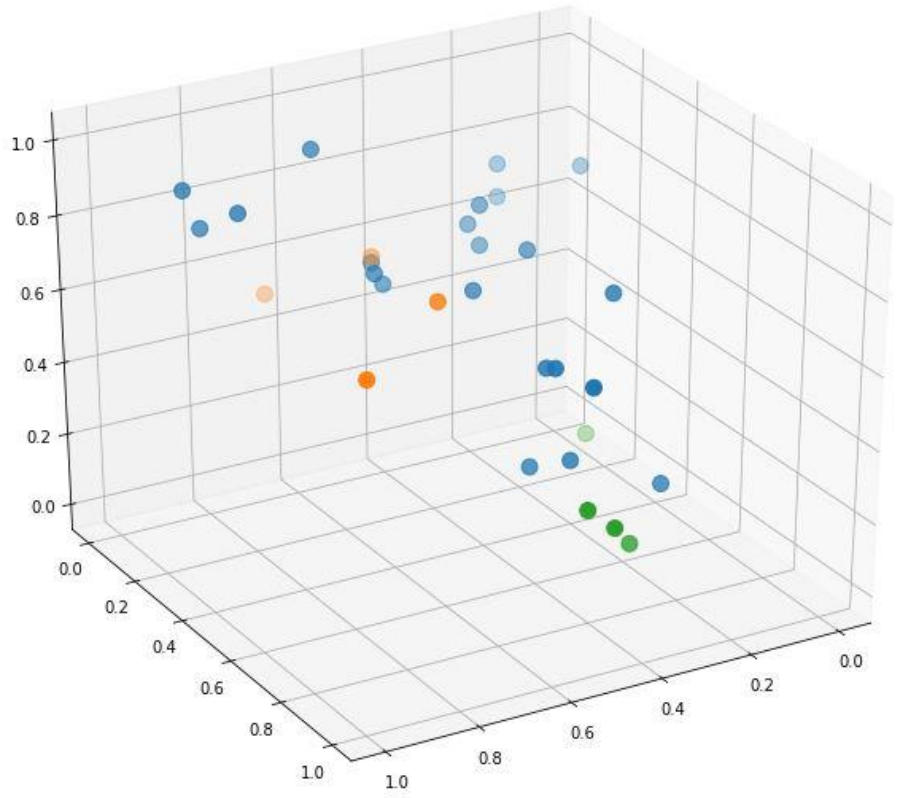

Рисунок 7. Метод доминирования, ближнего соседа, 3-кластера, нормирование данных с учетом весов

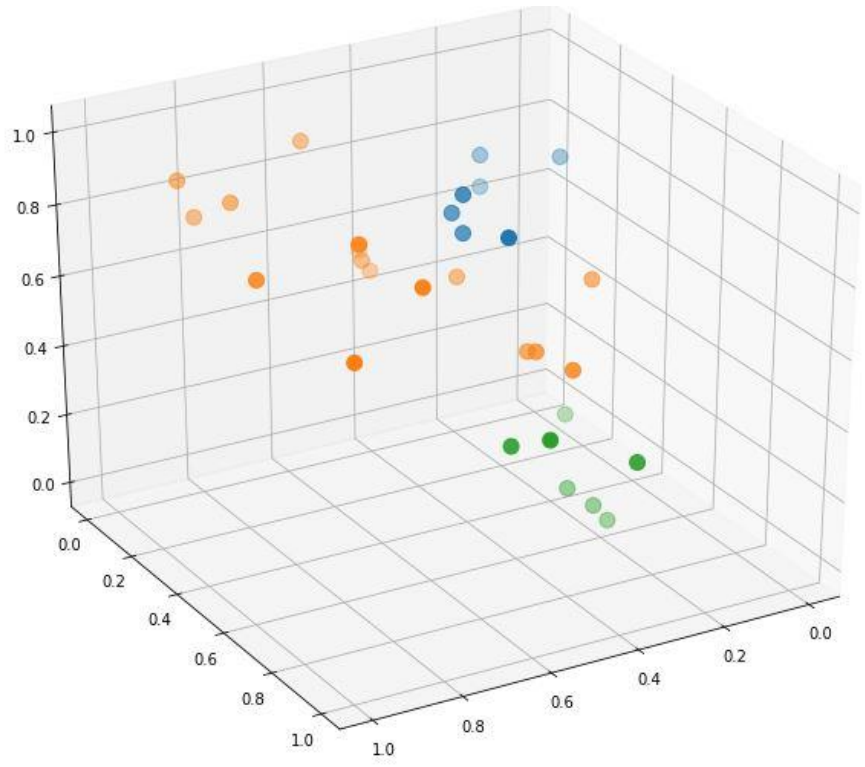

Рисунок 8. Метод доминирования, между центрами, 3-кластера, нормирование данных с учетом весов

4 Анализ результатов и выводы

В результате проведенного тестирования разработанной программы можно сказать, что метод Евклида справляется с задачей разделения на классы лучше, чем метод доминирования, а также стоит заметить, что при введении нормализации и до, получились различные результаты кластеризации при использовании метода доминирования.

5 Программный код

```
from math import sqrt
import ipywidgets as widgets
from IPython.display import display
from IPython.display import clear_output
import matplotlib.pyplot as plt
import copy as cp
from mpl_toolkits.mplot3d import Axes3D

# Множество образов
mn = [(0,1,1),(0,1,7),(5,7,4),
      (0,5,5),(9,4,5),(7,1,2),
      (10,0,19),(0,12,7),(-5,-4,5),
      (20,10,15),(0,16,-16),(-1,9,-30),
      (18,0,17),(6,18,4),(17,0,11),(14,16,18),
      (5,13,0),(-5,-4,0),(5,15,-11),(-3,10,-35),
      (16,2,15),(6,15,3),(-9,-2,5),(0,0,3),
      (19,17,11),(6,13,-14),(-4,5,-25),(15,12,20),
      (-2,10,-32),(7,2,0)]

# Формируем данные для отображения
data = []
x,y,z = [],[],[]
for i in mn:
    data.append([i[0],i[1],i[2]])
    x.append(i[0])
    y.append(i[1])
    z.append(i[2])

fig = plt.figure(figsize=(12.,10.))
ax = fig.add_subplot(111, projection='3d')
ax.view_init(30, 60)
ax.scatter(x,y,z,s=100)

# Метод Евклида
def euclid(a,b,h=False):
    d = 0
    if(h != False):
        for i in range(len(a)):
            d += h[i]*(a[i]-b[i])**2 # Учет веса
```

```
    else:
        for i in range(len(a)):
            d += (a[i]-b[i])**2
    return sqrt(d)

# Метод доминирования
def domin(a,b,h=False):
    d = 0
    if(h != False):
        d = [h[i]*abs(a[i]-b[i]) for i in range(len(a))]
    else:
        d = [abs(a[i]-b[i]) for i in range(len(a))]
    return max(d) # Берем максимум среди признаков

# Расстояние между центрами разных кластеров
def center(a,b, method,h=False):
    p1 = [sum([i[0] for i in a])/len(a),sum([i[1] for i in a])/len(a),sum([i[2] for i in a])/len(
a)]
    p2 = [sum([i[0] for i in b])/len(b),sum([i[1] for i in b])/len(b),sum([i[2] for i in b])/len(
b)]
    return method(p1,p2,h)

# Расстояние между ближайшими точками разных кластеров
def dclose(a,b,method,h=False):
    ds = []
    for i in a:
        for j in b:
            ds.append(method(i,j,h))
    return min(ds)

# Кластеризация методом слияния
def cluster(data,m1,m2,k,h=False):
    n = len(data) # Начальное кол-во кластеров

    # Все элементы принадлежат разным кластерам
    cls = {}
    for i in range(n):
        cls[i] = [data[i]]

    while(n != k):
        # Формируем расстояния между кластерами, чтобы далее слить два самых близких из них
        ds = {}
        for i,pts1 in cls.items():
            ds[i] = {}
            for j,pts2 in cls.items():
                if(i != j):
                    ds[i][j] = m2(pts1,pts2,m1,h)

        # Находим минимум расстояния между кластерами
        idx = next(iter(ds))
        jdx = next(iter(ds[idx]))
```

```
        for i,pts1 in ds.items():
            for j,pts2 in pts1.items():
                if(ds[idx][jdx] > ds[i][j]):
                    idx = i
                    jdx = j

        cls[idx] += cls[jdx] # Добавляем данные в первый найденный кластер
        del cls[jdx] # Убираем данные из второго найденного кластера
        n -= 1 # Уменьшаем кол-во кластеров на 1

    return cls # Возвращаем сформированные кластеры

# Нормализация данных
def normalize(data):
    # Нормировка данных методом минимакса
    mx = [max(i[0] for i in data),max(i[1] for i in data),max(i[2] for i in data)]
    ms = [min(i[0] for i in data),min(i[1] for i in data),min(i[2] for i in data)]
    norm = []
    for d in data:
        norm.append(d)
        for i in range(len(d)):
            norm[len(norm)-1][i] = (d[i] - ms[i])/(mx[i]-ms[i])
    return norm

# Расчет дисперсии
def disp(ls):
    avg = sum(ls)/len(ls)
    d = 0
    for i in ls:
        d += (i-avg)**2
    return d/len(ls)

# Получение весов
def weights(norm):
    h = []
    for i in range(3):
        h.append(1/disp([b[i] for b in norm]))
    return h

# Отрисовка меню выбора параметров и 3д-графика
def paint3d(cls,a=30,b=60):
    fig = plt.figure(figsize=(12.,10.))
    ax = fig.add_subplot(111, projection='3d')
    ax.view_init(a, b)
    for k,v in cls.items():
        x,y,z = [p[0] for p in v],[p[1] for p in v],[p[2] for p in v]
        ax.scatter(x,y,z,s=100,depthshade=True)

# Кол-во кластеров
slider = widgets.IntSlider(
    min=2,
```

```
    max=10,
    step=1,
    description='Кластеры:',
    value=3
)

# Первый метод - расчет расстояния между образами
btn1 = widgets.RadioButtons(
    options=['Евклида', 'Доминирования'],
    disabled=False
)

# Второй метод - расчет расстояния между кластерами
btn2 = widgets.RadioButtons(
    options=['Между центрами', 'Ближнего соседа'],
    disabled=False
)

# Ввод нормализации данных
box1 = widgets.Checkbox(False)

# Ввод весов
box2 = widgets.Checkbox(False)

# Углы для поворота рисунка
alpha = widgets.IntSlider(
    min=1,
    max=90,
    step=1,
    description='Alpha:',
    value=30
)

beta = widgets.IntSlider(
    min=1,
    max=90,
    step=1,
    description='Beta:',
    value=60
)

# Кнопка для применения изменений
start = widgets.Button(description='Запустить')

# Данные для тестирования
dataset = cp.deepcopy(data)

# Для упрощения
def menu():
    display(slider)
    print('Расстояние между образами')
```

```
    display(btn1)
    print('Расстояние между кластерами')
    display(btn2)
    print('Нормирование данных')
    display(box1)
    print('Введение весов')
    display(box2)
    print('Углы для отрисовки данных:')
    display(alpha)
    display(beta)
    display(start)

# Метод для применения изменений
def on_change(btn):
    dataset = cp.deepcopy(data)
    clear_output()
    menu()
    m1 = btn1.value # Первый метод
    m2 = btn2.value # Второй
    size = slider.value # Кол-во кластеров
    nr = box1.value # С нормализацией
    w = box2.value # С весами
    h = False
    if(m1 == 'Евклида'):
        m1 = euclid
    else:
        m1 = domin
    if(m2 == 'Между центрами'):
        m2 = center
    else:
        m2 = dclose
    if(nr):
        dataset = normalize(dataset)
    if(w):
        h = weights(dataset)
    cls = cluster(dataset,m1,m2,size,h)
    print('Результат кластеризации:')
    paint3d(cls,alpha.value,beta.value)

menu()
start.on_click(on_change)
# https://github.com/mammuthus/SUAI-Stuff/blob/main/clustering.py
```