Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский государственный национальный исследовательский университет»

Институт компьютерных наук и технологий

# Отчет
# по лабораторной работе № 8

по дисциплине
«Введение в анализ данных»

Студент Власова Елизавета Александровна

Группа ИТ–14–2023

Пермь 2025

# Оглавление

## Основная часть (5 баллов)

В этом задании дан датасет с синтетическими (специально сгенерированными) данными.

Ученый решил провести кластеризацию некоторого множества звезд по их расположению на карте звездного неба. Кластер звезд – это набор звезд (точек) на карте, лежащий внутри круга радиусом R. Каждая звезда принадлежит ровно одному кластеру. Центр кластера, или центроид, – это одна из звезд на карте, сумма расстояний от которой до всех остальных звезд кластера минимальна. Под расстоянием понимается расстояние Евклида.

В файле хранятся данные о звездах трех кластеров, R=3 для каждого кластера. В каждой строке записана информация о расположении на карте одной звезды: сначала координата x, затем координата y. Значения даны в условных единицах.

Определите координаты центра (центроида) каждого кластера.

**Решение:**

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist


# функция поиска точки в файле
def find_star(file, x, y):
    data = pd.read_csv(file, sep=';', decimal=',') # читаем файл с указанием, что десятичный разделитель – запятая
    match = data[(data['X'] == x) & (data['Y'] == y)] # фильтрация: выбираем строки где X и Y точно равны x и y
    # проверка и вывод результата
    if not match.empty:
        print("Точка найдена:")
        print(match)
        return match
    else:
        print("Точка не найдена")
        return None
```

Функция поиска точки в файле: открывает CSV-файл file (разделитель ';', десятичный разделитель ',') и пытается найти строки, где значения колонок 'X' и 'Y' равны переданным x и y. Возвращает:

- pandas.DataFrame с найденными строками, если найдены;

- None, если совпадений нет.

```
# чтение данных
df = pd.read_csv("27_B_17834.csv", encoding='utf-8', delimiter=';')
```

Загружаем CSV-файл. delimiter=';' — разделитель столбцов; encoding='utf-8' — стандартная кодировка.

```
# исправляем запятые
df['X'] = df['X'].str.replace(',', '.').astype(float)
df['Y'] = df['Y'].str.replace(',', '.').astype(float)
```

В исходном файле числа записаны с запятой как десятичным разделителем, поэтому сначала заменяем запятую на точку (строковая операция), затем приводим к float.

```
# извлекаем массив координат
data = df[['X', 'Y']].values
```

Извлекаем массив координат (n x 2) для использования в sklearn и вычислениях. Это двумерный массив, в котором 2 столбца (координаты x и y) и строк столько, сколько точек в файле.

```
# 1
# кластеризация KMeans
# создаем экземпляр KMeans на 3 кластера
# random_state задает зерно генератора случайных чисел, чтобы результаты были воспроизводимы
kmeans = KMeans(n_clusters=3, random_state=0)

# fit_predict одновременно обучает модель на данных и возвращает метки кластеров для каждой точки
# результат сохраняем в колонку 'cluster' датафрейма df, чтобы привязывать точки к кластерам
df['cluster'] = kmeans.fit_predict(data)
```

KMeans — это алгоритм, который автоматически группирует точки на плоскости в кластеры. Он делает 2 вещи:

- Находит, какие точки похожи/близки друг к другу, и помещает их в один кластер;
- Вычисляет центр каждого кластера. Центр – это средняя точка группы.

Здесь KMeans делает:

1. Случайно выбирает 3 центра
2. Назначает каждой точке ближайший центр
3. Пересчитывает центры
4. Повторяет процесс
5. Выдает:
    - номер кластера каждой точки
    - расчетные центры (но они – не всегда настоящие центроиды)

```python
# 2
# поиск центроидов

# список, в котором для каждого кластера будет храниться координата центроида (точка из набора, минимизирующая сумму расстояний)
real_centroids = []

# sorted(df['cluster'].unique()) — перебираем метки кластеров в упорядоченном виде
for cluster_id in sorted(df['cluster'].unique()):

    # выбираем все точки, принадлежащие данному кластеру, и берем только столбцы X и Y
    # .values дает массив точек в этом кластере
    cluster_points = df[df['cluster'] == cluster_id][['X', 'Y']].values

    # построение матрицы попарных евклидовых расстояний между точками кластера
    # distances[i, j] = расстояние между i-й и j-й точками
    distances = cdist(cluster_points, cluster_points, 'euclidean')

    # для каждой точки i вычисляем сумму расстояний до всех остальных: sums[i]
    # в матрице расстояний ищем сумму по каждой строке, чтобы найти минимум
    sums = distances.sum(axis=1)

    # np.argmin(sums) возвращает индекс точки с минимальной суммой расстояний
    # centroid — координаты этой точки (внутри кластера)
    centroid = cluster_points[np.argmin(sums)]

    # добавляем найденную точку в список центроидов
    real_centroids.append(centroid)

# преобразуем список в массив формы (k, 2), где k — число кластеров, и координаты центров
real_centroids = np.array(real_centroids)
```

```python
# 3
# фильтрация точек в заданном радиусе R=3

def filter_points_within_radius(data, labels, cluster_id, radius=3):

    # берем все точки, у которых метка == cluster_id
    cluster_points = data[labels == cluster_id]

    # в этом методе центр окружности берется как среднее арифметическое точек кластера (не центроид для кластера)
    # .mean(axis=0) - вычисляет среднее значение по каждому столбцу в массиве
    centroid = cluster_points.mean(axis=0)

    # distances — евклидовы расстояния от каждой точки к вычисленному среднему центроиду
    distances = np.linalg.norm(cluster_points - centroid, axis=1)

    # возвращаем только те точки, у которых расстояние <= radius
    return cluster_points[distances <= radius]

# применяем фильтрацию ко всем кластерам и собираем результаты
filtered_data = []
labels = df['cluster'].values

# перебираем кластеры у порядоченном виде
for cluster_id in sorted(df['cluster'].unique()):
    # добавляем массив отфильтрованных точек для текущего кластера в список
    filtered_data.append(filter_points_within_radius(data, labels, cluster_id))

# np.vstack объединяет список массивов в один массив (сложение по вертикали)
filtered_data = np.vstack(filtered_data)

# превращаем результат фильтрации в DataFrame для удобства отображения
filtered_df = pd.DataFrame(filtered_data, columns=["X", "Y"])
```

Фильтрует точки одного кластера, оставляя только те, которые находятся на расстоянии <= radius от (среднего) центроида кластера.

Параметры:

- data: numpy array shape (n,2) с координатами всех точек;

- labels: массив меток кластеров длины n (например, df['cluster'].values);

- cluster_id: метка кластера, для которого выполняется фильтрация;

- radius: порог расстояния (в тех же единицах, что и X,Y).

Возвращает:

numpy array shape (m,2) — отфильтрованные точки данного кластера.

```
# 4
# графики (центроиды + данные)
plt.figure(figsize=(7, 7))

# рисуем все точки
plt.scatter(df['X'], df['Y'], c=df['cluster'], cmap='viridis', s=10, alpha=0.6)
```

Рисуем все точки: цвет точки соответствует метке кластера (колонка df['cluster']). Параметры:

- s=10: размер маркера;

- alpha=0.6: прозрачность (чтобы при наложении точек не терять вид);

- cmap='viridis': колормар для визуального разделения кластеров.

```
# рисуем центроиды
plt.scatter(real_centroids[:, 0], real_centroids[:, 1],
            c='black', s=80, marker='o', label='Центроиды')
```

Рисуем реальные центроиды (точки из данных) черными кругами. Параметры:

- c='black': цвет чёрный;

- s=80: размер маркера (умеренно большой для видимости);

- marker='o': круг;

- label — подпись для легенды.

```
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Кластеризация с центроидами")
plt.legend()
plt.grid(True)
plt.show()
```

Подписи осей и заголовок графика.

Показываем легенду (обозначение центроидов).

Включаем сетку для удобства визуального считывания координат.

Отображаем окно с графиком.

```
# 5
# вывод координат центроидов и поиск в файле

# для найденных центроидов печатаем их координаты
# и вызываем find_star для проверки, есть ли такая запись в датасете
for i, c in enumerate(real_centroids):
    print(f"\nКластер {i+1}: центроид ({c[0]:.3f}, {c[1]:.3f})")
    find_star("27_B_17834.csv", c[0], c[1])
```

Форматированный вывод с тремя знаками после запятой для удобства читаемости. Ищем в исходном файле точную строку с такими же координатами.

**Результаты работы программы:**

```
Кластер 1: центроид (7.563, 5.851)
Точка найдена:
          X      Y
9435  7.563  5.851

Кластер 2: центроид (0.549, 5.016)
Точка найдена:
          X      Y
2585  0.549  5.016

Кластер 3: центроид (4.146, -0.294)
Точка найдена:
          X      Y
3374  4.146 -0.294
```