Постановка задачи

Реализация алгорима кластеризации K-means с целью лучшего понимания его работы. Оценка точности полученных на выходе результатов с помощью внешних и внутренних метрик качества.

Теоретическое описание алгорима

Имеем множество X, на котором определена некоторая метрика ρ. В частности, в качестве X рассмотрим пространство $\mathbb{R}^N$ с евклидовой метрикой $\|\cdot\|$.

Имеем множество образцов $X_L \subset X$ — обучающая выборка. Задача состоит в том, чтобы по множеству X_L построить метод кластеризации, который будет разбивать все множество X по аналогии с X_L на K кластеров. Количество K задано заранее.

Пусть X_L^k, $k = 1, 2, \ldots, K$ представляет собой множество с элементами k-го кластера, тогда

$$X_L = \bigcup_{k=1}^{K} X_L^k.$$

Пусть имеется **центральный элемент** k-го кластера $\mu_k \in M_k$.

Рассмотрим произвольный $x \in X_L$. Если этот $x \in X_L^{k^*}$ $k^* \in \{1, 2, \ldots, K\}$, то

$$\rho(x, \mu_{k^*}) = \min_{k=1,\ldots,K} \rho(x, \mu_k). \tag{1}$$

То есть элемент обучающей выборки x принадлежит кластеру k^*, если x **ближе всего** к его центру тяжести μ_{k^*}.

Если окажется так, что $\rho(x, \mu_{k^*}) = \rho(x, \mu_{k^{**}})$, то нужно принять дополнительное соглашение о выборе кластера для x. Пусть, например, из равносильных претендентов будет выбран тот, который имеет меньший номер. Тогда запись (1) будет однозначно определять кластер, к которому принадлежит выбранный x.

Описание алгорима K-means:

1. Инициализируем $\{\mu_k\}$ произвольно, таким образом, что $\mu_k \neq \mu_l$ при $k \neq l$. Это могут быть случайные элементы множества X_L. Как только выбраны центры — получена разбивка на кластеры
$$X_L^k = \{\mu_k\}.$$
2. Выбираем произвольный элемент $x \in X_L$. На этом элементе происходит обучение.
3. Находим k^*, такой, что
$$\rho(x, \mu_{k^*}) = \min_k \rho(x, \mu_k),$$
то есть $x \in X_L^{k^*}$.
4. Добавляем x в найденное подмножество
$$X_L^{k^*} = X_L^{k^*} \cup \{x\}.$$
5. Проводим процедуру усреднения. Находим новый центр кластера
$$\mu_{k^*} = \frac{x_{k^*}^1 + x_{k^*}^2 + \cdots + x_{k^*}^{|X_L^{k^*}|}}{|X_L^{k^*}|}.$$
6. Переход к шагу 2.

На выходе получаем усредненные $\{\mu_k\}$, которые занимают точные положения центров кластеров.

Использование алгорима на синтетических данных

Сначала создадим искусственную выборку исходных данных (точек на плоскости), из смеси двух гауссиан (получаем два кластера). Для синей смеси параметры заданы как $\mathcal{M}[X] = -3, \mathcal{M}[Y] = -1, \sigma_X = 2, \sigma_Y = 3$, для зеленой — $\mathcal{M}[X] = 2, \mathcal{M}[Y] = 7, \sigma_X = 1, \sigma_Y = 3$.

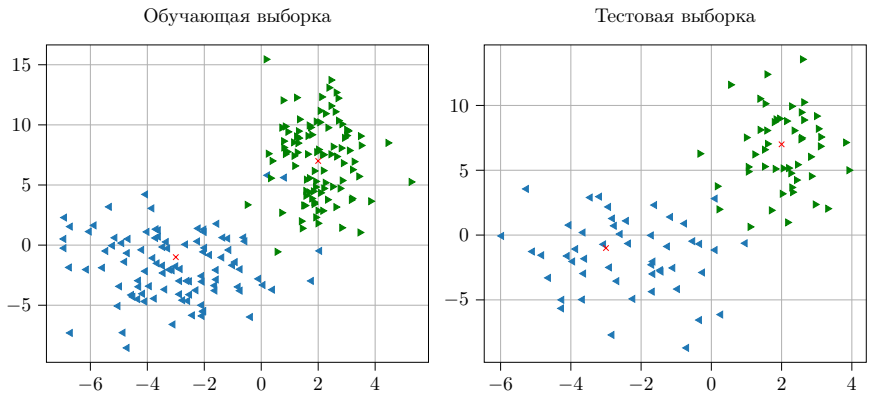

Рис. 1: Сгенерированные данные в виде точек на плоскости

В обучающей выборке 200 объектов (по 100 на кластер), в тестовой — 100 объектов (по 50 на кластер). Крестиками обозначены центры кластеров — заданные мат. ожидания. Вычисленный коэффициент силуэта для обучающей выборки — 0.589, для тестовой — 0.57.

Произведено 3 запуска алгоритма на 100 итераций. Решением алгорима являются центры кластеров, которые при каждом запуске стремятся занять положения заданных на этапе генерации мат. ож. распределений.

Истинные центроиды		Полученные центроиды		
Кластер 1	Кластер 2	Запуск	Кластер 1	Кластер 2
(-3, -1)	(2, 7)	1	(-3.06, -1.66)	(2.09, 7.4)
		2	(-2.34, -1.9)	(1.96, 7.16)
		3	(-3.13, -1.98)	(1.73, 7.02)

Таблица 1: Результаты работы алгорима после 3 запусков

Рассмотрим подробнее 3 запуск. На следующих графиках видно, как изменялись центроиды и разбиения на каждой 10 итерации.

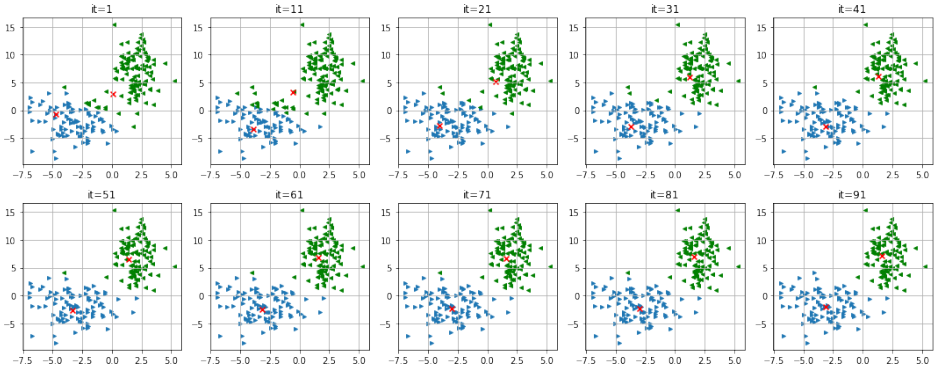

Рис. 2: Изменения положений центроидов и разбиений

Рассмотрим графики точности кластеризации: изменение внутренней метрики (Silhouette) и внешней метрики (Adjusted Rand Index, ARI) на каждой итерации (3 запуск).

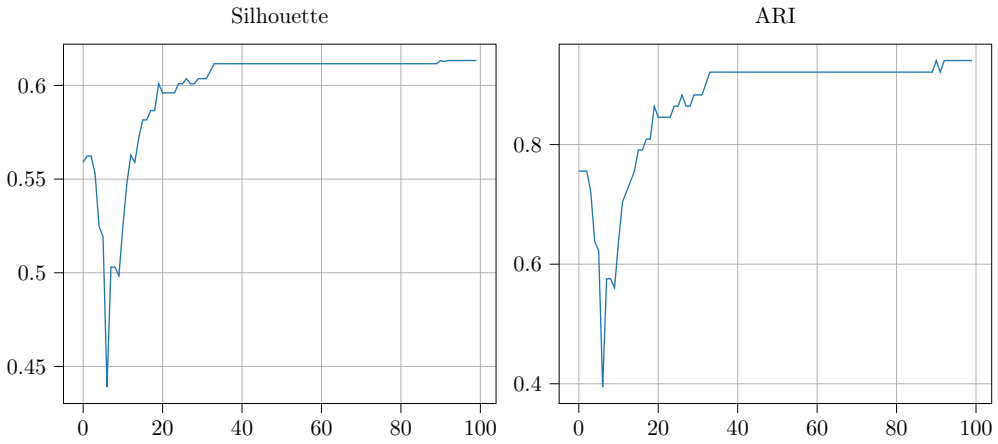

Рис. 3: Графики точности кластеризации

Приведем также сводную таблицу результатов (по третьему запуску).

№ итерации	Количество элементов в выборке	Количество правильно кластеризованных	Количество неправильно кластеризованных	Accuracy	Silhouette	ARI
Обучающая выборка						
1	200	187	13	0.935	0.56	0.76
11		180	20	0.9	0.52	0.64
21		192	8	0.96	0.6	0.85
31		194	6	0.97	0.6	0.88
41		196	4	0.98	0.61	0.92
51		196	4	0.98	0.61	0.92
61		196	4	0.98	0.61	0.92
71		196	4	0.98	0.61	0.92
81		196	4	0.98	0.61	0.92
91		197	3	0.985	0.61	0.94
Тестовая выборка						
-	100	97	3	0.97	0.58	0.88

Таблица 2: Сводная таблица результатов

Использование алгорима на реальных данных

Выбранный датасет представляет собой реальные данные о классификации зерен трех разных сортов пшеницы на основе их геометрических свойств [1]. У каждого случайно выбранного для эксперимента представителя выборки выделены следующие 7 признаков:

- **Площадь** A
- **Периметр** P
- $C = \frac{4\pi A}{P^2}$
- **Длина зерна**
- **Ширина зерна**
- **Коэффициэнт ассиметрии**
- **Длина желобка**

Целевым признаком являются **сорта** пшеницы : “Kama”, “Rosa”, “Canadian”.

В обучающей выборке данные о 150 зернах (по 50 на каждый сорт). В тестовом датасете данные о 60 объектах (в каждом классе по 20). Вычисленный коэффициент силуэта для обучающей выборки — 0.45, для тестовой — 0.35.

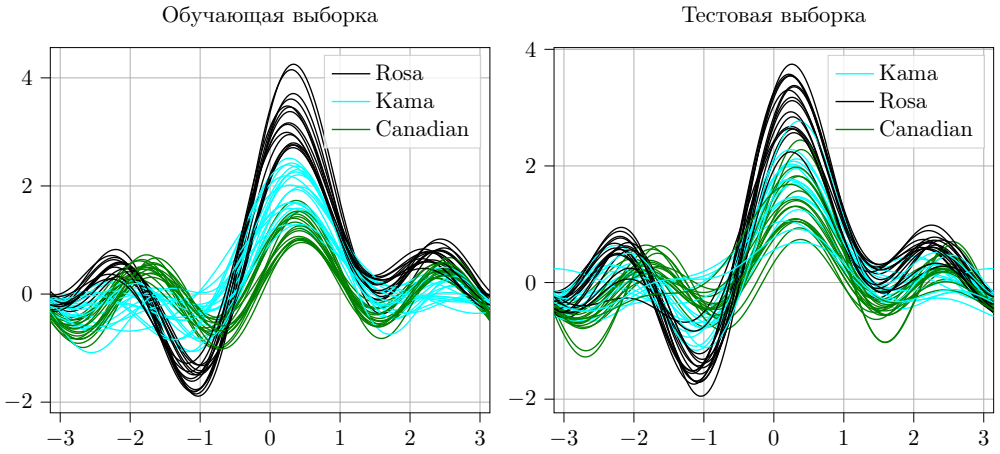

Рис. 4: Многомерные данные в виде кривых Эндрюса

Данные нормализованы к отрезку $[0, 1]$, произведено 3 запуска алгоритма на 100 итераций.

Истинные центроиды			**Полученные центроиды**			
Kama	Rosa	Canadian	Запуск	Kama	Rosa	Canadian
(0.37, 0.41, 0.68, 0.36, 0.46, 0.28, 0.24)	(0.77, 0.81, 0.68, 0.75, 0.77, 0.42, 0.77)	(0.11, 0.17, 0.31, 0.2, 0.12, 0.60, 0.27)	1	(0.41, 0.45, 0.69, 0.4, 0.5, 0.26, 0.28)	(0.74, 0.79, 0.67, 0.73, 0.74, 0.41, 0.74)	(0.13, 0.18, 0.35 , 0.2, 0.15,0.57, 0.25)
			2	(0.28, 0.32, 0.57, 0.29, 0.34, 0.35, 0.25)	(0.73, 0.77, 0.67, 0.71, 0.75, 0.41, 0.1)	(0.08, 0.15, 0.26, 0.2, 0.09, 0.73, 0.27)
			3	(0.41, 0.44, 0.71, 0.38, 0.5, 0.35, 0.29)	(0.76, 0.8, 0.68, 0.74, 0.75, 0.44, 0.76)	(0.11, 0.18, 0.29 , 0.2, 0.12, 0.58, 0.26)

Таблица 3: Результаты работы алгорима после 3 запусков

Рассмотрим графики точности кластеризации: изменение внутренней метрики (Silhouette) и внешней метрики (Adjusted Rand Index, ARI) на каждой итерации (3 запуск).

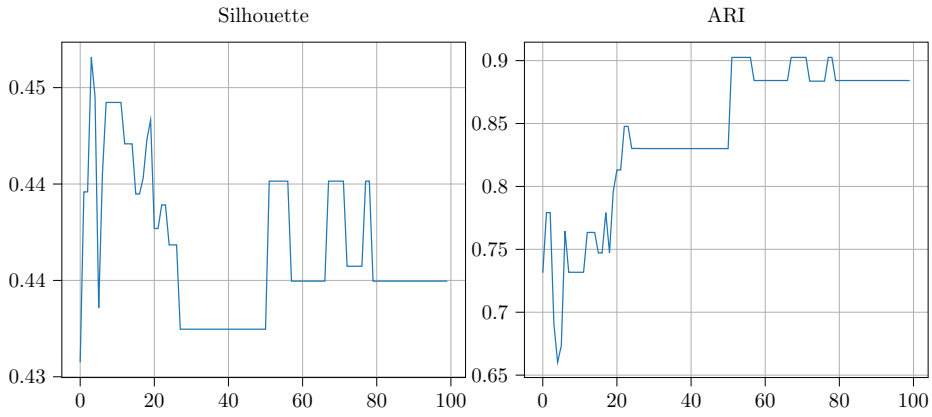

Рис. 5: Графики точности кластеризации

Приведем также сводную таблицу результатов (по третьему запуску).

№ итерации	Количество элементов в выборке	Количество правильно кластеризованных	Количество неправильно кластеризованных	Accuracy	Silhouette	ARI
Обучающая выборка						
11	150	135	15	0.9	0.44	0.73
11		135	15	0.9	0.44	0.73
21		140	10	0.93	0.44	0.81
51		141	9	0.94	0.43	0.83
51		141	9	0.94	0.43	0.83
51		141	9	0.94	0.43	0.83
61		144	6	0.96	0.43	0.88
71		145	5	0.97	0.44	0.90
81		144	6	0.96	0.43	0.88
91		144	6	0.96	0.43	0.88
Тестовая выборка						
-	60	49	11	0.82	0.38	0.54

Таблица 4: Итерационная сводная таблица результатов

Также рассмотрим сводную таблицу по классам.

Класс	Количество элементов в выборке	Количество правильно кластеризованных	Количество неправильно кластеризованных	Accuracy
Обучающая выборка				
Kama	50	45	5	0.90
Rosa	50	49	1	0.98
Canadian	50	50	0	1.00
Тестовая выборка				
Kama	20	16	4	0.80
Rosa	20	18	2	0.90
Canadian	20	15	5	0.75

Таблица 5: Сводная таблица результатов по классам

Заключение

1. Использование любого алгоритма машинного обучения на синтетических данных позволяет познакомиться с его спецификой: в данном случае интересно было наблюдать процесс обучения — поиск методом k-means центроидов для простых двумерных данных. Исходя из таблицы 1 и рисунка 2 видно, что центроиды из начального их положения стремятся занять положения мат. ож. — центров тяжести компонетов смеси (кластеров). Отмечу также, что внутренняя мера силуэт может быть вычислена априори для сгенерированной выборки — вычисленная на каждом шаге алгоритма апостериорная метрика не может сильно превышать априорную и может использоваться в качестве критерия окончания работы алгоритма. Также интересно изменение внешней метрики ARI: ее значение зависит от точности (accuracy) и может как увеличиваться во время обучения, так и уменьшаться. Также метрика силуэт и метрика ARI на начальных итерациях увеличиваются и уменьшаются взаимосвязанно. Итоговая точность как на обучающей, так и на тестовой выборке сильно зависит от начального выбора центроидов.

2. Использование алгоритма МО на реальных многомерных данных позволяет проверить его работоспособность в “боевых” условиях. Полезным открытием является возможность визуализации многомерных данных с помощью кривых Эндрюса: уже на них четко наблюдается искомое разбиение. Также была проведена предварительная нормализация исходных данных: так как внутри вектора значения никак не связаны друг с другом, имеют разные единицы измерения, нормализацию проводим по признаку. Поскольку выбранный датасет не является огромным, то целесообразно использовать все имеющиеся данные в качестве обучающей выборки, но мы решили разбить исходные данные с целью посмотреть, хороши ли найденные на обучающей выборке центры. Убедились, что данные на тестовой выборке классифицируются хуже, чем на обучающей. Также интересно рассмотреть, какова точность на каждом конкретном классе: какие-то классы обнаруживаются лучше, какие-то хуже, исходя из специфики работы алгорима. Зависимость точности от увеличения количества итераций не выявлена — всегда есть предельная точность, ниже 100%.

Список литературы

[1] Charytanowicz,Magorzata, Niewczas,Jerzy, Kulczycki,Piotr, Kowalski,Piotr, and Lukasik,Szymon. (2012). seeds. UCI Machine Learning Repository. DOI

[2] Курс "Математические методы анализа технологической информации". А.А.Ефремов. Томский политехнический университет. 2021. Лекция 4. Кластеризация

[3] Курс "Машинное обучение и искусственный интеллект в математике и приложениях". Р.В.Шамин. НОЦ МИАН им. В.А.Стеклова. 2017. Лекция 9. EM-алгоритмы в задачах автоматической классификации

Листинг программы

```
import numpy as np
import math
from matplotlib import pyplot as plt
import tikzplotlib
import pandas as pd

#
# ОБЩИЕ ФУНКЦИИ
#

def silhouette_score(data, partitioning):
    """
    [2]
    Вычисление внутренней метрики качества силуэт для заданного набора данных и его разбиения.

    Аргументы:
    data          -- заданный набор данных.
    partitioning  -- полученное тем или иным алгоритмом разбиение.

    Выходное значение:
    значение метрики.
    """
    s = [] # список вычисленных силуэтов для каждого элемента
    for elem in data:
        b = []
        for cluster in partitioning:
            if elem in cluster:
                ai = 1 / (len(cluster) - 1) if len(cluster) > 1 else 0
                ai_sum = 0
                for clelem in cluster:
                    if not np.array_equal(clelem, elem):
                        ai_sum += np.linalg.norm(clelem - elem)
                ai *= ai_sum
            else:
                bi = 1 / len(cluster)
                bi_sum = 0
                for clelem in cluster:
                    bi_sum += np.linalg.norm(clelem - elem)
                bi *= bi_sum
                b.append(bi)
        min_b = min(b) if b else 0
        s.append( (min_b - ai) / max(ai, min_b) )
    return sum(s) / len(s)

def adjusted_rand_index(true, predict):
    """
    [2]
    Вычисление внешней метрики качества ARI между истинным разбиением и разбиением алгорима.

    Аргументы:
    true     -- истинное разбиение.
    predict  -- предсказанное разбиение.

    Выходное значение:
    значение метрики.
    """
    contingency_matrix = [[0 for _ in range(len(predict))] for _ in range(len(true))]
    for i,true_cluster in enumerate(true):
        for j,predict_cluster in enumerate(predict):
            contingency_matrix[i][j] = len(set(tuple(elem) for elem in true_cluster).intersection(set(tuple(elem) for elem in predict_cluster)))

    a = sum([math.comb(contingency_matrix[i][j], 2) for i in range(len(true)) for j in range(len(predict))])
    b = sum([math.comb(len(cluster), 2) for cluster in true ])
    c = sum([math.comb(len(cluster), 2) for cluster in predict ])
    N = sum([len(cluster) for cluster in true])
    d = 2 * b * c / (N * (N - 1))

    return (a - d) / (0.5 * (b + c) - d)

def findBMU(mu_k, x): # best matching unit
    """
    Поиск наиболее близкого центра для элемента.

    Аргументы:
    mu_k -- массив центров.
    x    -- рассматриваемый элемент.

    Выходное значение:
    индекс кластера ближайшего центра.
```

```
    """
    return min([ (np.linalg.norm(x - mu), i)  for i, mu in enumerate(mu_k) ])[1]

def getPartitioning(data, mu_k, K):
    """
    Получить разбиение с текущими центрами

    Аргументы:
    data -- датасет.
    mu_k -- массив центров.
    K    -- количество кластеров.

    Выходное значение:
    разбиение.
    """
    clusters = [ [] for _ in range(K)]

    for elem in data: # определяем кластер каждого элемента выборки
        k = findBMU(mu_k, elem)
        clusters[k].append(elem)

    return [ np.array(clusters[i]) for i in range(K)]

def getRightWrongObjects(true, predict):
    """
    Получить количество верно и неверно классифицируемых объектов.

    Аргументы:
    true     -- истинное разбиение.
    predict  -- предсказанное разбиение.

    Выходное значение:
    количество верных и неверных объектов.
    """
    contingency_matrix = [[0 for _ in range(len(predict))] for _ in range(len(true))]
    for i,true_cluster in enumerate(true):
        for j,predict_cluster in enumerate(predict):
            contingency_matrix[i][j] = len(set(tuple(elem) for elem in true_cluster).intersection(set(tuple(elem) for elem in predict_cluster)))

    return sum([max(row) for row in contingency_matrix]), sum([sum(row) - max(row) for row in contingency_matrix])

def k_means_fit(K, data, true_partitioning, tmax = 1000, scores = True, make_plots = True, make_table = True):
    """
    Обучение алгоритма - его реализация

    Аргументы:
    K                -- желаемое количество кластеров.
    data             -- обучающий датасет.
    true_partitioning -- истинное разбиение.
    tmax             -- количество итераций (default 1000).
    scores           -- подсчет метрик (default True).
    make_plots       -- вывод графиков (K=2, default True).
    make_table       -- вывод сводной таблицы (default True).

    Выходное значение:
    вычисленные центры кластеров.
    """
    clusters = [ [] for _ in range(K)] # кластерные множества
    mu_k = [0] * K # центры
    len_data = len(data) # длина обучающей выборки
    partitioning = None
    if scores:
        silhouette = []
        ari = []
    if make_plots and K == 2:
        fig, axes = plt.subplots(2, 5, figsize=(15, 6))
    if make_table:
        col_width = 0

    # рандомно инициализируем 'центры' случайными элементами из обучающей выборки
    for i in range(K):
        elem = data[np.random.choice(len_data)]
        while any([np.array_equal(e, elem) for e in mu_k[:i]]):
            elem = data[np.random.choice(len_data)]
        clusters[i].append(elem)
        mu_k[i] = clusters[i][-1]

    for i in range(tmax):
        # рассматриваем случайный элемент выборки
        ind = np.random.choice(len_data)
        x = np.copy(data[ind])
```

```
        # ищем ближайший 'центр'
        k = findBMU(mu_k, x)

        # если этого элемента нет в найденном кластере
        if not any([np.array_equal(elem, x) for elem in clusters[k]]):
            clusters[k].append(x) # добавляем его в кластерное множество

            mu_k[k] = np.sum(clusters[k], axis = 0) / len(clusters[k]) # уточняем центр после добавления
                                                                      # усредняем

        if scores:
            partitioning = getPartitioning(data, mu_k, K)
            silhouette.append(silhouette_score(data, partitioning))
            ari.append(adjusted_rand_index(true_partitioning, partitioning))
        if not i % (tmax / 10):
            if partitioning is None:
                partitioning = getPartitioning(data, mu_k, K)
            if make_plots and K == 2:
                ax = axes[i // (tmax // 2), (i - (tmax // 2)) // (tmax // 10) if i // (tmax // 2) else i // (tmax // 10) ]
                ax.scatter(partitioning[0][:, 0], partitioning[0][:, 1], s = 15, marker = '>')
                ax.scatter(partitioning[1][:, 0], partitioning[1][:, 1], s = 15, c = 'green', marker = '<')
                ax.scatter(mu_k[0][0], mu_k[0][1], s = 35, marker = 'x', c = 'red')
                ax.scatter(mu_k[1][0], mu_k[1][1], s = 35, marker = 'x', c = 'red')
                ax.set_title(f'it={i + 1}')
                ax.grid()
            if make_table:
                if not i:
                    headers = ["Итерация", "Количество элементов в обучающей выборке", \
                               "Количестве правильно кластеризованных", "Количестве неправильно кластеризованных"]
                    if scores:
                        headers.append("Силуэт")
                        headers.append("ARI")
                    col_width = len(max(headers, key=len)) + 2
                    print(" | ".join(header.ljust(col_width) for header in headers))
                    print("-" * (len(headers) * (col_width + 2) - 1))

                row = [i + 1, len(data)]

                right, wrong = getRightWrongObjects(true_partitioning, partitioning)

                row += [right, wrong]

                if scores:
                    row.append(silhouette[-1])
                    row.append(ari[-1])

                print(" | ".join(str(item).ljust(col_width) for item in row))

    if make_plots and K == 2:
        plt.tight_layout()
        tikzplotlib.save("iter.tex", flavor="context")
        plt.show()

    if scores:
        fig, axes = plt.subplots(1, 2, figsize=(15, 8))
        axes[0].plot(silhouette)
        axes[0].set_title('Silhouette')
        axes[0].grid()
        axes[1].plot(ari)
        axes[1].set_title('ARI')
        axes[1].grid()
        plt.tight_layout()
        tikzplotlib.save("metrics.tex", flavor="context")
        plt.show()

    return mu_k

def k_means_val(K, data, mu_k, true_partitioning):
    """
    Валидация на тестовых данных

    Аргументы:
    data              -- тестовый датасет.
    mu_k              -- массив центров.
    K                 -- количество кластеров.
    true_partitioning -- истинное разбиение.

    Выходное значение:
    строка с результатами на тесте.
    """
```

```
    partitioning = getPartitioning(data, mu_k, K)
    silhouette = silhouette_score(data, partitioning)
    ari = adjusted_rand_index(true_partitioning, partitioning)

    right, wrong = getRightWrongObjects(true_partitioning, partitioning)

    headers = ["Количество элементов в тестовой выборке", "Количестве правильно кластеризованных", \
               "Количестве неправильно кластеризованных", "Силуэт", "ARI"]
    row = [len(data), right, wrong, silhouette, ari]

    col_width = len(max(headers, key=len)) + 2
    return " | ".join(header.ljust(col_width) for header in headers) + "\n"\
           "" + " | ".join(str(item).ljust(col_width) for item in row)

#
# РАБОТА С СИНТЕТИЧЕСКИМ ДАТАСЕТОМ
#

def make_blobs_two(N: int, mu1: tuple, mu2: tuple, sigma1: tuple, sigma2: tuple, \
                   make_plot=True, tikz=True, silhouette=True, cent=True):
    """
    Генерация смеси из двух гауссовых распределений

    Аргументы:
    N              -- количество элементов в распределении.
    mu1, mu2       -- кортежи мат. ож. распределений.
    sigma1, sigma2 -- кортежи ср. кв. отклонений.
    make_plot      -- вывод графика (default True).
    tikz           -- сохранение выведенного графика в формате tikz (make_plot=True, default True).
    silhouette     -- вывод силуэта сгенерированного распределения (default True).
    cent           -- вывод на графике положений центров (мат. ож.) (default True).

    Выходное значение:
    data_set       -- сгенерированный датасет.
    clusters       -- истинное разбиение.
    """
    x1, y1 = np.random.normal(mu1[0], sigma1[0], N), np.random.normal(mu1[1], sigma1[1], N)
    x2, y2 = np.random.normal(mu2[0], sigma2[0], N), np.random.normal(mu2[1], sigma2[1], N)

    if make_plot:
        plt.scatter(x1, y1, s = 20, marker = '>')
        plt.scatter(x2, y2, s = 20, c = 'green', marker = '<')
        if cent:
            plt.scatter([mu1[0], mu2[0]], [mu1[1], mu2[1]], s = 40, marker = 'x', c = 'red')
        plt.grid()
        if tikz:
            filename = input("Input .tex filename: ")
            tikzplotlib.save(f"{filename}.tex", flavor="context")
        plt.show()

    clusters = [np.c_[x1, y1], np.c_[x2, y2]]
    data_set = np.concatenate(clusters)
    np.random.shuffle(data_set)
    if silhouette:
        print(f"Silhouette of generated dataset: {silhouette_score(data_set, clusters)}")

    return data_set, clusters


train_set, train_partitioning = make_blobs_two(100, (-3, -1), (2, 7), (2, 3), (1, 3))
test_set, test_partitioning = make_blobs_two(50, (-3, -1), (2, 7), (2, 3), (1, 3))
mu_k = k_means_fit(2, train_set, train_partitioning, tmax=100)
# print("Запуск 1: ", mu_k)
# print("Запуск 2: ", mu_k)
print("Запуск 3: ", mu_k)
print(k_means_val(2, test_set, mu_k, test_partitioning))

#
# РЕАЛЬНЫЕ ДАННЫЕ
#

def normalize(data):
    """
    Нормализация данных так, чтобы каждый столбец содержал 0, 1 и значения из интервала (0, 1).

    Аргументы:
    data -- набор ненормализованных данных.

    Выходное значение:
    нормализованный набор данных.
    """
```

```
    for n in range(len(data[0])):
        max_n, min_n = max(data[:, n]), min(data[:, n])
        k, b = 1 / (max_n - min_n), -min_n / (max_n - min_n)
        for m in range(len(data)):
            data[m, n] *= k
            data[m, n] += b
    return data

def load_set(names: list, set_type='train', make_plot=True, norm=True, balanced=False,\
             silhouette=True, statistic=True, cent=False):
    """
    Загрузка данных из .csv

    Аргументы:
    names      -- наименования классов.
    set_type   -- тип загружаемого датасета (train, test) (default train).
    make_plot  -- вывод графика (default True).
    silhouette -- вывод силуэта сгенерированного распределения (default True).
    cent       -- вывод на графике положений центров (мат. ож.) (default True).
    norm       -- нормализация данных (default True).
    balanced   -- сбалансировать обучающую выборку (set_type = train, default False)
    statistic  -- вывести количество объектов по классам.

    Выходное значение:
    data_set   -- сгенерированный датасет.
    clusters   -- истинное разбиение.
    """
    if set_type != 'train':
        set_type = 'test'

    set_df = pd.read_csv(f'{set_type}.csv')
    data_set = np.array(set_df.iloc[:, :-1])
    labels = np.array(set_df.iloc[:, -1])

    if norm:
        data_set = normalize(data_set)
        set_df.iloc[:, :-1] = pd.DataFrame(data_set)

    if make_plot and norm:
        x = pd.plotting.andrews_curves(set_df.sample(frac=1, random_state=42).iloc[:50],\
                                       list(set_df.columns)[-1], color=['black', 'cyan', 'green'])
        tikzplotlib.save(f"andrews_{set_type}.tex", flavor="context")
        plt.show()

    clusters = [[] for _ in range(len(names))]
    for i, label in enumerate(labels):
        clusters[names.index(label)].append(data_set[i].copy())

    if set_type == 'train' and balanced:
        min_cl_len = len(min(clusters, key=len))
        for i in range(len(clusters)):
            clusters[i] = clusters[i][:min_cl_len]
        data_set = np.array([ elem for cluster in clusters for elem in cluster ])

    clusters = [np.array(cluster) for cluster in clusters]
    np.random.shuffle(data_set)

    if silhouette:
        print(f"Silhouette of load {set_type} dataset: {silhouette_score(data_set, clusters)}")

    if statistic:
        print(f"In {set_type} dataset ...")
        for i, name in enumerate(names):
            print(f"{name} - {len(clusters[i])} instances;")
        print(f"ALL - {len(data_set)} instances")

    if set_type == 'train' and cent:
        print(f"Expected centroids ...")
        for i, name in enumerate(names):
            print(f"{name} - {np.sum(clusters[i], axis = 0) / len(clusters[i])}")

    return data_set, clusters

def getRightWrongObjectsPerClass(true, predict, names):
    """
    Получить количество верно и неверно классифицируемых объектов в каждом классе

    Аргументы:
    true     -- истинное разбиение.
    predict  -- предсказанное разбиение.
    names    -- наименования классов.
```

```
    Выходное значение:
    результирующая строка.
    """
    contingency_matrix = [[0 for _ in range(len(predict))] for _ in range(len(true))]
    for i,true_cluster in enumerate(true):
        for j,predict_cluster in enumerate(predict):
            contingency_matrix[i][j] = len(set(tuple(elem) for elem in true_cluster).intersection(set(tuple(elem) for elem in predict_cluster)))

    result = []
    [ result.append(f"{names[i]} : {max(row)}, {sum(row) - max(row)}") for i,row in enumerate(contingency_matrix)]
    return '\n'.join(result)


class_names = ['Kama', 'Rosa', 'Canadian']
train_set, train_partitioning = load_set(class_names, cent=True)
test_set, test_partitioning = load_set(class_names, set_type = 'test')
mu_k = k_means_fit(3, train_set, train_partitioning, tmax=100, make_plots = False)
# print("Запуск 1: ", mu_k)
# print("Запуск 2: ", mu_k)
print("Запуск 3: ", mu_k)
print(k_means_val(3, test_set, mu_k, test_partitioning))
print(getRightWrongObjectsPerClass(train_partitioning, getPartitioning(train_set, mu_k, 3), class_names))
print(getRightWrongObjectsPerClass(test_partitioning, getPartitioning(test_set, mu_k, 3), class_names))
```