МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе № 5
по дисциплине «Машинное обучение»
Тема: Кластеризация (k-средних, иерархическая)

Студенты гр. 6304 _____ Григорьев И.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы

Ознакомиться с методами кластеризации модуля *Sklearn*.

Ход работы

Загрузка данных

Датасет загружен в датафрейм. Вид данных представлен на рис. 1.

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

Рисунок 1 – Исходные данные

K-means

1. Выполнена кластеризация исходных данных с помощью метода k-средних.
2. Получены центры кластеров и для каждого наблюдения определена принадлежность к кластеру.
3. Результаты кластеризации для попарных признаков (1 и 2, 2 и 3, 3 и 4) приведены на рис. 2.

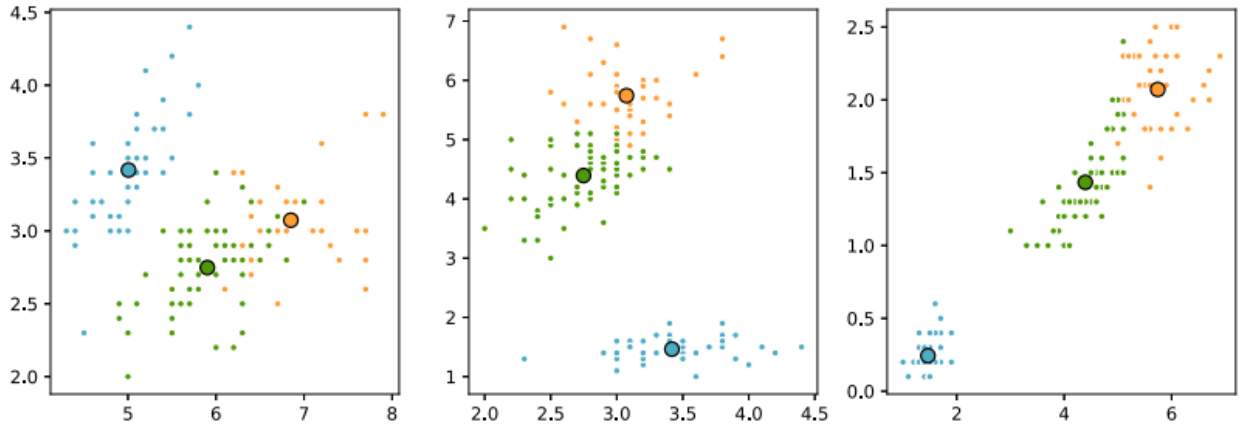

Рисунок 2 – Результат кластеризации для попарных признаков

Наилучшее разделение произошло по 3 и 4 признаку (3 график). Параметр *n_init* определяет количество раз, когда алгоритм k-средних будет выполняться с разными начальными значениями центроидов. По итогу выбирается лучший результат (с наименьшим значением ошибки).

4. С помощью метода главных компонент размерность данных уменьшена до 2. Карта для всей области значений, на которой каждый кластер занимает определенную область своим цветом, представлена на рис. 3.

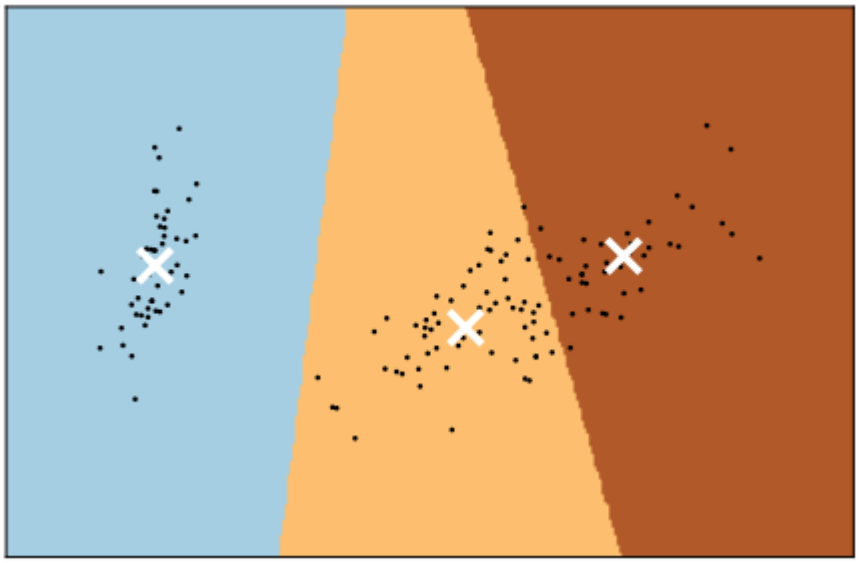

Рисунок 3 – Карта области значений для уменьшенной размерности

5. Исследована работа алгоритма k-средних при различных параметрах *init*. Результаты запуска с параметром random представлены на рис. 4. Результаты запуска для вручную выбранных начальных центроидов представлены на рис. 5.

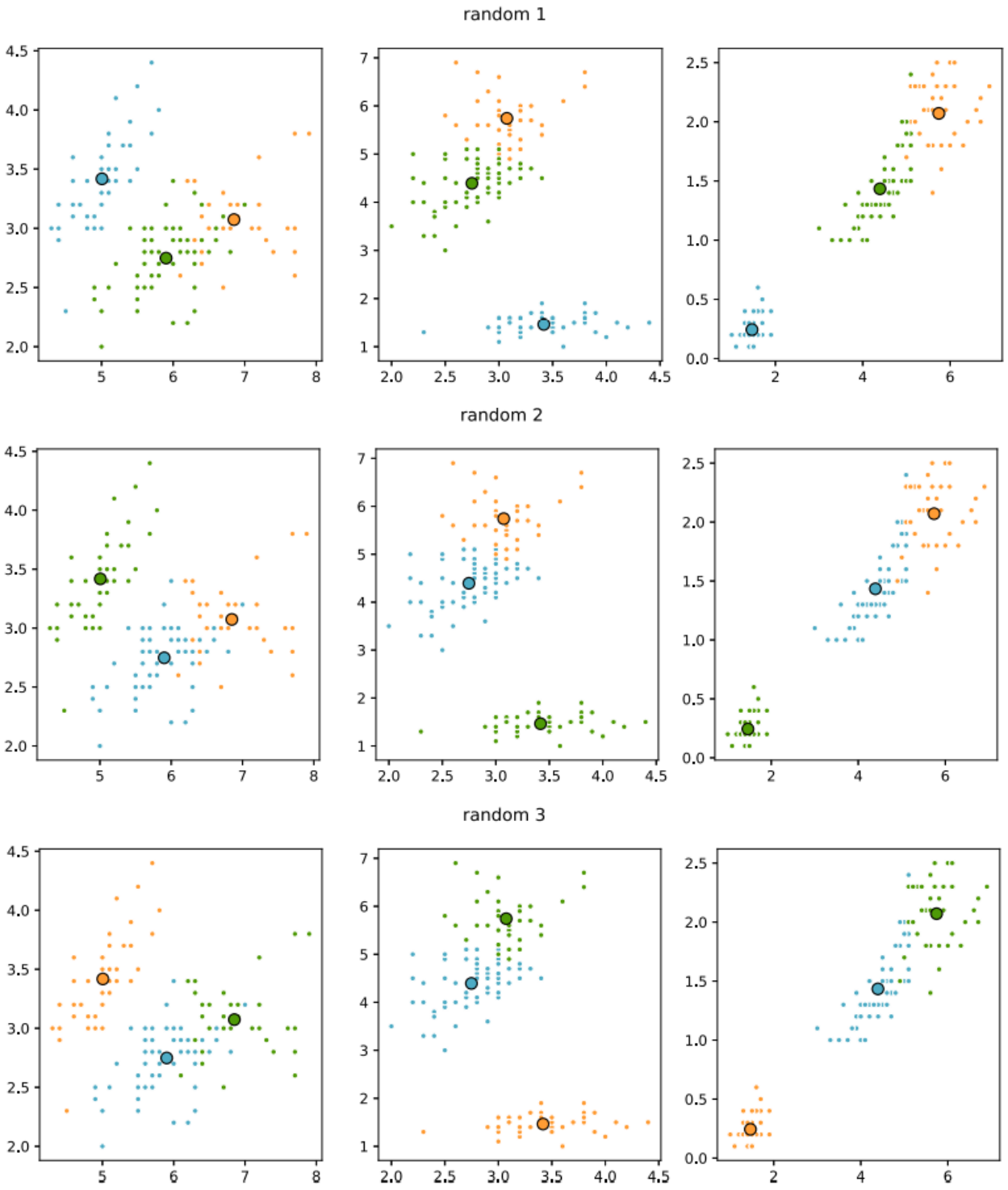

Рисунок 4 – Результаты k-средних для random

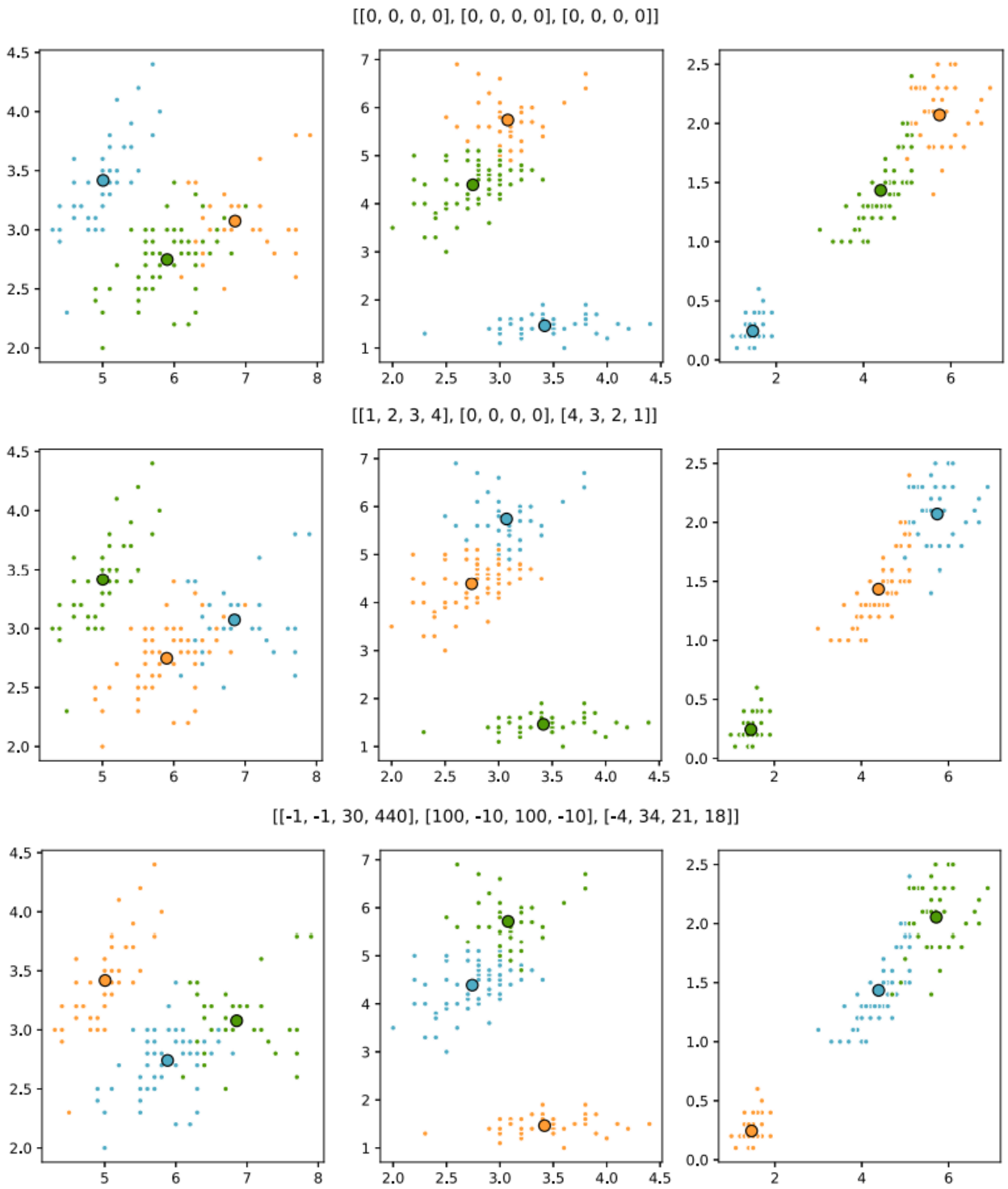

Рисунок 5 – Результаты k-средних для вручную выбранных начальных центроидов

При запусках k-средних с параметром *random* и при ручной настройке начальных значений центроидов меняется только порядок следования меток. Однако также замечено небольшое смещение итоговых центров кластеров для *n_init* = 1.

6. Определено наилучшее количество кластеров методом локтя. Результаты представлены на рис. 6.

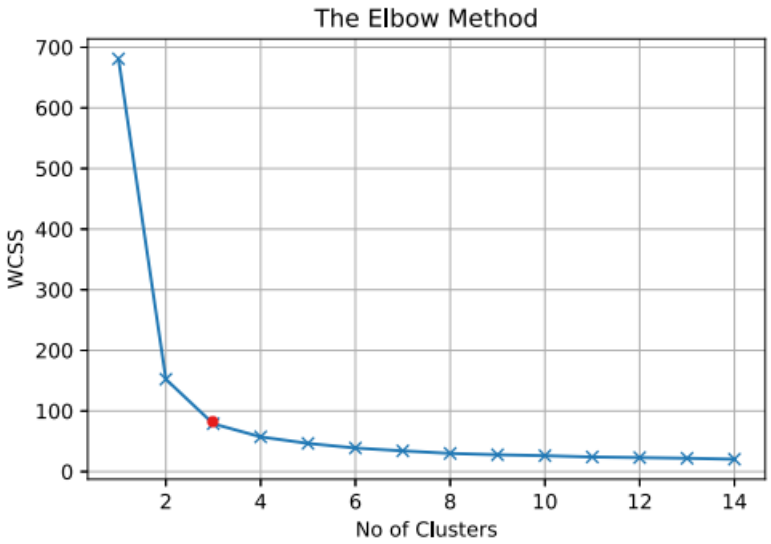

Рисунок 6 – Метод локтя

7. Выполнена кластеризация с помощью пакетной кластеризации k-средних. Пакетная кластеризация k-средних обрабатывает данные пакетами определенных размеров для уменьшения времени вычислений, что также приводит к снижению точности. Диаграммы рассеяния с отображением попадания точек в разные кластеры для разных методов представлены на рис. 7.

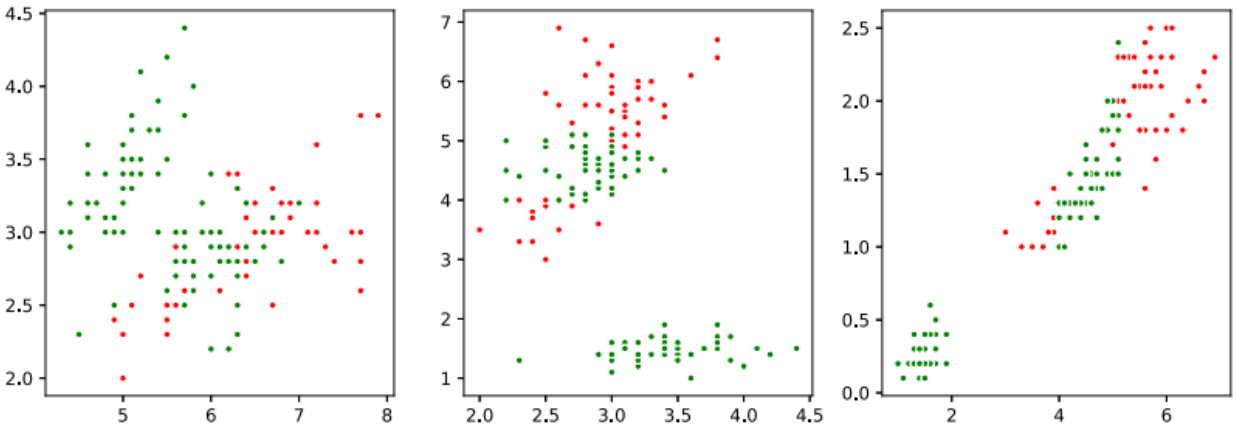

Рисунок 7 – Различие KMeans и MiniBatchKMeans (красные точки попали в разные кластеры)

Иерархическая кластеризация

1. Выполнена иерархическая кластеризация на тех же данных.
2. Результат кластеризации продемонстрирован на рис. 8.

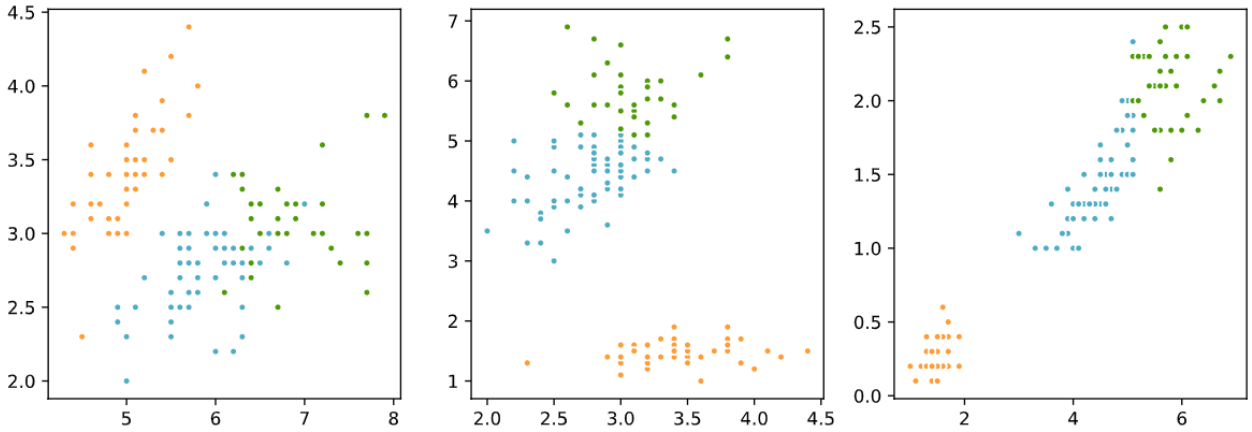

Рисунок 8 – Результат иерархической кластеризации для попарных признаков

AgglomerativeClustering выполняет иерархическую кластеризацию с использованием восходящего подхода: изначально каждое наблюдение располагается в своем собственном кластере, и ближайшие кластеры последовательно объединяются. При объединении кластеров расстояния пересчитываются с помощью определенных метрик.

В k-средних имеем начальные значения центроидов, количество которых равно количеству кластеров. Наблюдения распределяются по кластерам, исходя из близости к центроидам, затем центроиды пересчитываются и т.д.

3. Проведено исследование для различного размера кластеров (от 2 до 5). Результат продемонстрирован на рис. 9.

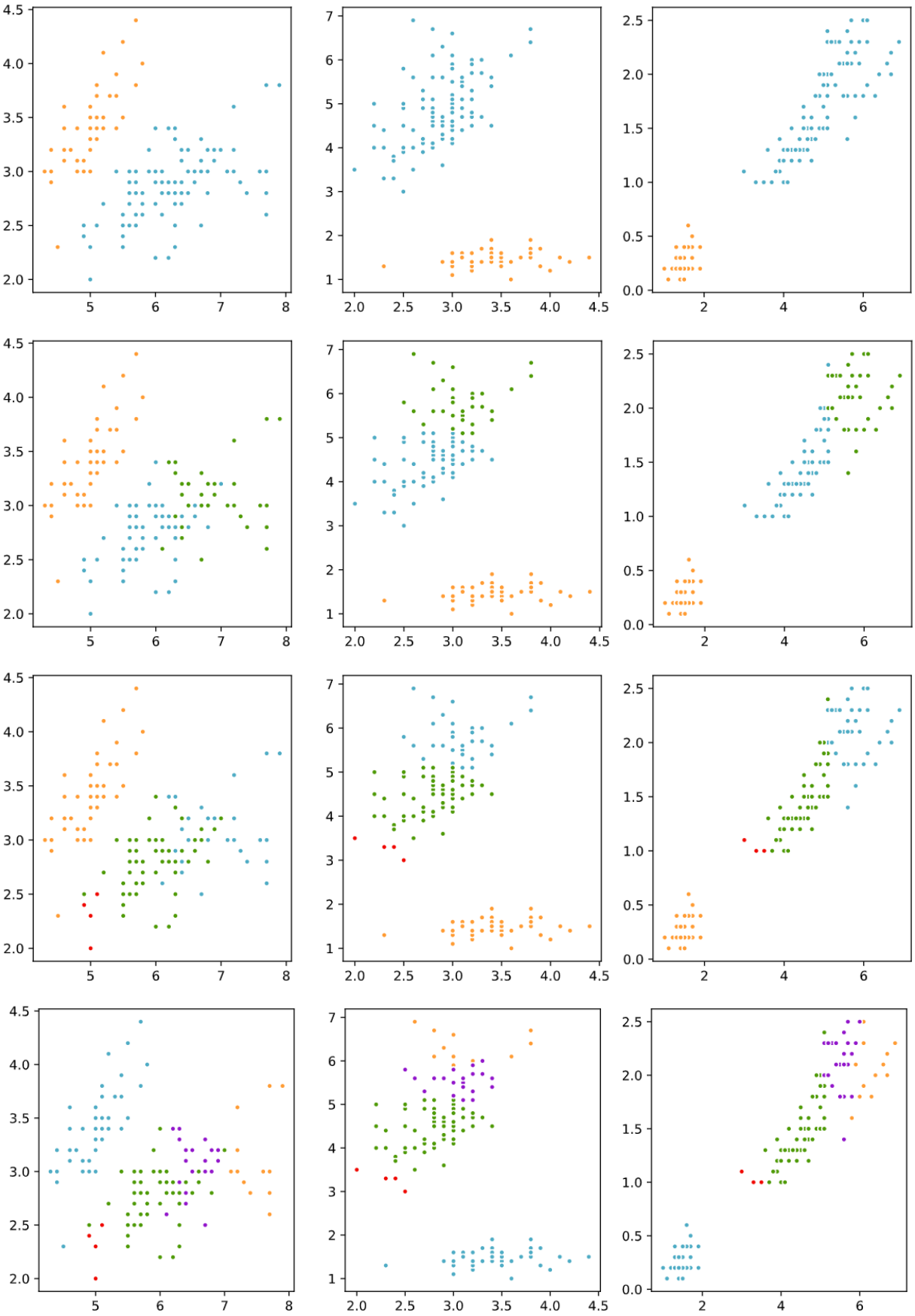

Рисунок 9 – AgglomerativeClustering для количества кластеров от 2 до 5

4. Иерархия кластеров представлена в виде дерева (дендрограммы). Корень дерева – уникальный кластер, который содержит все наблюдения, а листья – кластеры только с одним наблюдением. Дендрограмма представлена на рис. 10.

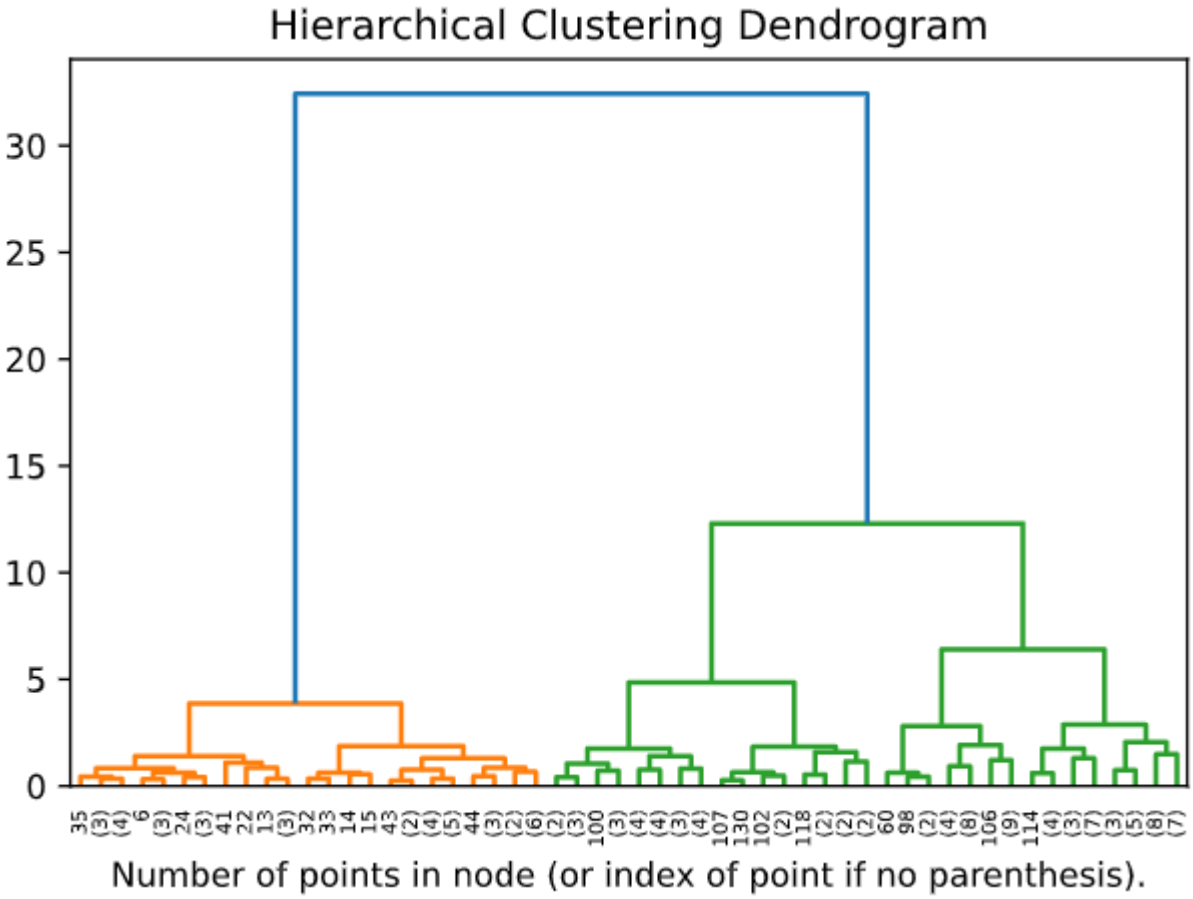

Рисунок 10 - Дендрограмма

5. Сгенерированы случайные данные в виде двух колец.
6. Проведена иерархическая кластеризация с метрикой Уорда для определения расстояния, результат представлен на рис. 11.

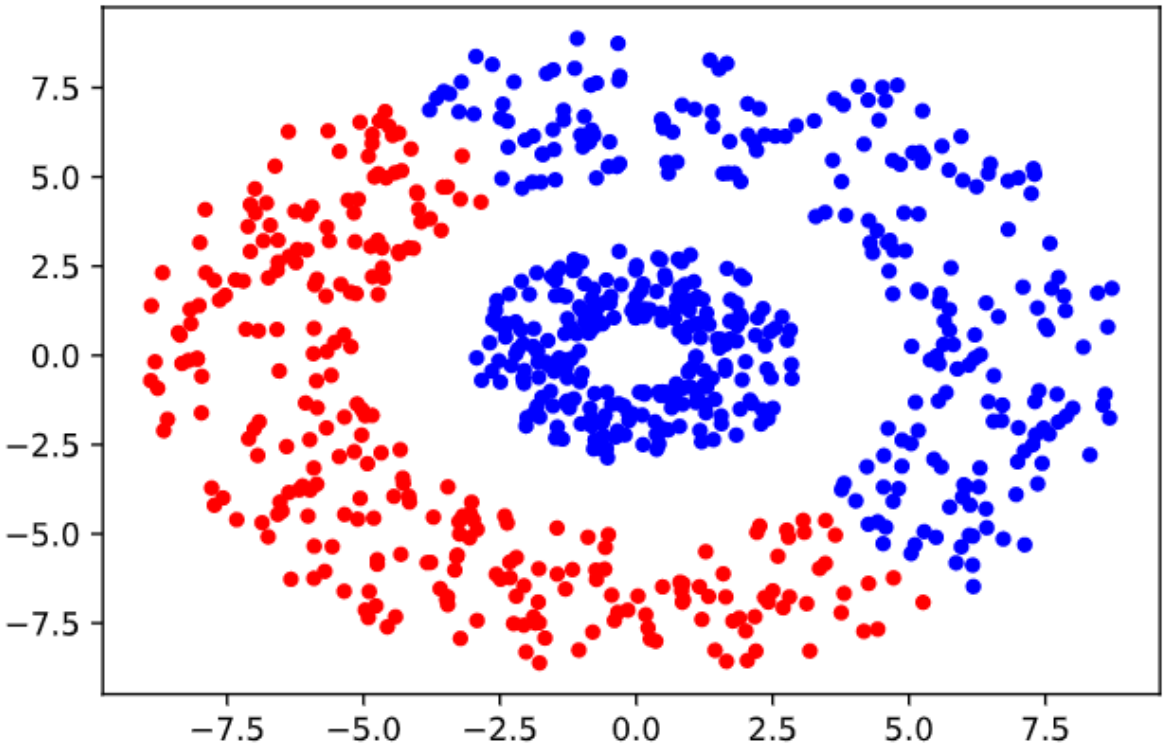

Рисунок 11 – AgglomerativeClustering с методом Уорда для определения расстояния

7. AgglomerativeClustering исследован при всех остальных параметрах linkage. Результаты представлены на рис. 12.

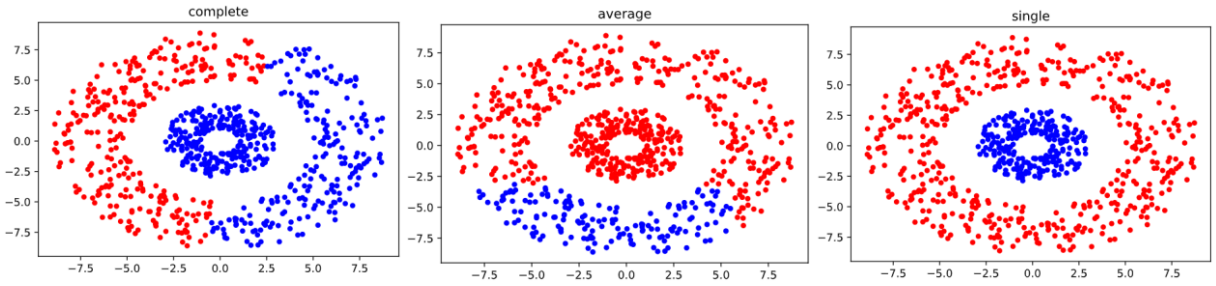

Рисунок 12 – AgglomerativeClustering при различных linkage

Таким образом, для разбиения данных на 2 кольца подходит только Single Link метрика.

Выводы

В ходе лабораторной работы изучены такие методы кластеризации модуля *Sklearn*, как KMeans и MiniBatchKMeans (KMeans с уменьшением времени вычислений и потерей точности), а также метод иерархической кластеризации AgglomerativeClustering, с помощью которого выявлена нелинейная зависимость в данных с использованием метрики Single Link.

Приложение А

Код программы на python

```
# To add a new cell, type '# %%'
# To add a new markdown cell, type '# %% [markdown]'
# %%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, MiniBatchKMeans, AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram
from sklearn.metrics.pairwise import pairwise_distances_argmin
from sklearn.decomposition import PCA
import random
import math

# %%
data = pd.read_csv('iris.data',header=None)
data

# %%
no_labeled_data = data.drop(4, axis=1)
no_labeled_data

# %%
k_means = KMeans(init='k-means++', n_clusters=3, n_init=15)
k_means.fit(no_labeled_data)

# %%
k_means.n_iter_

# %%
k_means.cluster_centers_

# %%
k_means.labels_

# %%
def plot_kmeans(k_means, data, title=''):
    f, ax = plt.subplots(1, 3, figsize=(12, 4))
    f.suptitle(title)
    colors = ['#4EACC5', '#FF9C34', '#4E9A06']
    print(ax)
    for i in range(3):
        my_members = k_means.labels_ == i
        cluster_center = k_means.cluster_centers_[i]
        for j in range(3):
            ax[j].plot(data[my_members][j], data[my_members][j+1], 'w', markerfacecol
or=colors[i], marker='o', markersize=4, lw=0)
            ax[j].plot(cluster_center[j], cluster_center[j+1], 'o', markerfacecolor=c
olors[i], markeredgecolor='k', markersize=8)
    plt.show()
plot_kmeans(k_means, no_labeled_data)

# %%
pca = PCA(n_components = 2)
reduced_data = pca.fit_transform(no_labeled_data)

# %%
kmeans = KMeans(init='k-means++', n_clusters=3)
kmeans.fit(reduced_data)
```

```
# Step size of the mesh. Decrease to increase the quality of the VQ.
h = .02     # point in the mesh [x_min, x_max]x[y_min, y_max].

# Plot the decision boundary. For that, we will assign a color to each
x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].max() + 1
y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Obtain labels for each point in mesh. Use last trained model.
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1)
plt.clf()
plt.imshow(Z, interpolation='nearest',
           extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap=plt.cm.Paired,
           aspect='auto', origin='lower')

plt.plot(reduced_data[:, 0], reduced_data[:, 1], 'k.', markersize=2)
# Plot the centroids as a white X
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1],
            marker='x', s=169, linewidths=3,
            color='w', zorder=10)
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()

# %%
k_means = KMeans(init='random', n_clusters=3).fit(no_labeled_data)
plot_kmeans(k_means, no_labeled_data, 'random 1')

# %%
k_means = KMeans(init='random', n_clusters=3).fit(no_labeled_data)
plot_kmeans(k_means, no_labeled_data, 'random 2')

# %%
k_means = KMeans(init='random', n_clusters=3).fit(no_labeled_data)
plot_kmeans(k_means, no_labeled_data, 'random 3')

# %%
k_means = KMeans(init=np.array([[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]), n_cluster
s=3).fit(no_labeled_data)
plot_kmeans(k_means, no_labeled_data, '[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]')

# %%
k_means = KMeans(init=np.array([[1, 2, 3, 4], [10, 10, 10, 10], [4, 3, 2, 1]]), n_clu
sters=3).fit(no_labeled_data)
plot_kmeans(k_means, no_labeled_data, '[[1, 2, 3, 4], [10, 10, 10, 10], [4, 3, 2, 1]]
')

# %%
k_means = KMeans(init=np.array([[-1, -1, 30, 440], [100, -10, 100, -10], [-
4, 34, 21, 18]]), n_clusters=3).fit(no_labeled_data)
plot_kmeans(k_means, no_labeled_data, '[[-1, -1, 30, 440], [100, -10, 100, -10], [-
4, 34, 21, 18]]')

# %%
```

```
wcss=[]
for i in range(1,15):
    kmean = KMeans(n_clusters=i,init="k-means++")
    kmean.fit_predict(no_labeled_data)
    wcss.append(kmean.inertia_)

plt.plot(range(1,15), wcss, marker='x')
plt.title('The Elbow Method')
plt.xlabel("No of Clusters")
plt.ylabel("WCSS")
plt.grid()
plt.show()

# %%
batch_km = MiniBatchKMeans(n_clusters=3, batch_size=10, n_init=1)
batch_km.fit(no_labeled_data)

# %%
km = KMeans(n_clusters=3, n_init=1)
km.fit(no_labeled_data)

# %%
diff_labels = np.array([km_l != batch_km_l for km_l, batch_km_l in zip(km.labels_, ba
tch_km.labels_)])
diff_labels

# %%
f, ax = plt.subplots(1, 3, figsize=(12, 4))
for j in range(3):
    ax[j].plot(no_labeled_data[diff_labels][j], no_labeled_data[diff_labels][j+1], 'w
', markerfacecolor='r', marker='o', markersize=4, lw=0)
    ax[j].plot(no_labeled_data[~diff_labels][j], no_labeled_data[~diff_labels][j+1],
'w', markerfacecolor='g', marker='o', markersize=4, lw=0)
plt.show()

# %%
hier = AgglomerativeClustering(n_clusters=5, linkage='average')
hier = hier.fit(no_labeled_data)
hier_labels = hier.labels_
hier_labels

# %%
f, ax = plt.subplots(1, 3, figsize=(12, 4))
colors = ['#4EACC5', '#FF9C34', '#4E9A06', '#F00800', '#9215CB']
for i in range(5):
    my_members = hier_labels == i
    for j in range(3):
        ax[j].plot(no_labeled_data[my_members][j], no_labeled_data[my_members][j+1],
'w', markerfacecolor=colors[i], marker='o', markersize=4, lw=0)
plt.show()

# %%
def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
```

```
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)

# %%
# setting distance_threshold=0 ensures we compute the full tree.
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None)
model = model.fit(no_labeled_data)
plt.title('Hierarchical Clustering Dendrogram')
# plot the top three levels of the dendrogram
plot_dendrogram(model, truncate_mode='level', p=5)
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()

# %%
data1 = np.zeros([250,2])
for i in range(250):
    r = random.uniform(1, 3)
    a = random.uniform(0, 2 * math.pi)
    data1[i,0] = r * math.sin(a)
    data1[i,1] = r * math.cos(a)
    data2 = np.zeros([500,2])
for i in range(500):
    r = random.uniform(5, 9)
    a = random.uniform(0, 2 * math.pi)
    data2[i,0] = r * math.sin(a)
    data2[i,1] = r * math.cos(a)
data = np.vstack((data1, data2))

# %%
hier = AgglomerativeClustering(n_clusters=2, linkage='single')
hier = hier.fit(data)
hier_labels = hier.labels_

# %%
my_members = hier_labels == 0
plt.plot(data[my_members, 0], data[my_members, 1], 'w', marker='o', markersize=4, col
or='red',linestyle='None')
my_members = hier_labels == 1
plt.plot(data[my_members, 0], data[my_members, 1], 'w', marker='o', markersize=4, col
or='blue',linestyle='None')
plt.title('single')
plt.show()

# %%
```