



# Специальные технологии баз данных и информационных систем

---

НИЯУ МИФИ, КАФЕДРА ФИНАНСОВОГО МОНИТОРИНГА

КУРС ЛЕКЦИЙ

В.Ю. РАДЫГИН. ЛЕКЦИЯ 4. СЕМЕСТР 2

# Библиотеки

---

В данной лекции будут рассмотрены примеры с использованием следующих библиотек:

- NumPy – <https://numpy.org/>
- Pandas – <https://pandas.pydata.org/>
- scikit-learn – <https://scikit-learn.org>
- Matplotlib – <https://matplotlib.org/>

# Часть 1

---

МЕТОДЫ МЕТОД СИГНАЛА РОДСТВА

# Affinity Propagation

---

Метод Affinity Propagation или по-русски метод сигнала родства (или метод распространения близости) – это один из алгоритмов кластеризации, основная задача которого – самостоятельно выяснить число кластеров.

# Основная идея

---

В основе алгоритма лежит расчёт трёх матриц: матрицы близости  $S$ , вычисляемой один раз в начале, а также матриц выразительности (responsibility)  $R$  и характерности (availability)  $A$ , вычисляемых итеративно до сходимости.

# Матрица близости

---

Матрица близости – это матрица, где каждый элемент  $s(i, j)$  показывает степень близости наблюдений  $i$  и  $j$ . Если задана какая-то функция расстояния между двумя наблюдениями, то степень близости будет равна квадрату значения данной функции, взятому со знаком минус:

$$s(i, j) = -\|x_i - x_j\|^2,$$

Если степень близости определяется, как декартово расстояние, то мы получаем:

$$s(i, j) = -\sum_{k=1}^N (x_{i,k} - x_{j,k})^2,$$

где  $N$  – размерность признакового пространства.

# Пример

---

	Мат. Ан.	Физика	Информ.
Мария	5	5	3
Петр	5	4	5
Ирина	3	4	3
Ангелина	3	3	5
Иван	2	5	5

	Мария	Петр	Ирина	Ангелина	Иван
Мария	0	-5	-5	-12	-13
Петр	-5	0	-8	-5	-10
Ирина	-5	-8	0	-5	-6
Ангелина	-12	-5	-5	0	-5
Иван	-13	-10	-6	-5	0

$$-12 = -((5 - 3)^2 + (5 - 3)^2 + (3 - 5)^2)$$

# Замечание

---

Матрица близости симметрична и имеет нулевую главную диагональ.

Чтобы алгоритм AP работал вместо нулей по диагонали записывается минимальный элемент матрицы близости.

	Мария	Петр	Ирина	Ангелина	Иван
Мария	-13	-5	-5	-12	-13
Петр	-5	-13	-8	-5	-10
Ирина	-5	-8	-13	-5	-6
Ангелина	-12	-5	-5	-13	-5
Иван	-13	-10	-6	-5	-13



# Матрица выразительности

---

Выразительность  $r(i, j)$  показывает насколько по сравнению с другими элементами элемент  $j$  выражает признаки элемента  $i$ :

$$r(i, j) = s(i, j) - \max_{j' \neq j} [a(i, j') + s(i, j')],$$

где  $a(i, j)$  – элемент матрицы характерности  $R$  в строке  $i$  и столбце  $j$ .

# Пример

	Мария	Петр	Ирина	Ангелина	Иван
Мария	-13	-5	-5	-12	-13
Петр	-5	-13	-8	-5	-10
Ирина	-5	-8	-13	-5	-6
Ангелина	-12	-5	-5	-13	-5
Иван	-13	-10	-6	-5	-13

	Мария	Петр	Ирина	Ангелина	Иван
Мария	-8	0	0	-7	-8
Петр	0	-8	-3	0	-5
Ирина	0	-3	-8	0	-1
Ангелина	-7	0	0	-8	0
Иван	-8	-5	-1	1	-8

$$-7 = -12 - \max(-5, -5, -13, -5) = -12 + 5$$

# Матрица характерности

---

Характерность  $a(i, j)$  показывает насколько элемент  $i$  характерен для элемента  $j$ . Для внедиагональных элементов формула следующая:

$$a(i, j) = \min \left[ 0, r(j, j) + \sum_{i' \neq i, j} r(i', j) \right].$$

Для диагональных элементов:

$$a(i, i) = \sum_{i' \neq i} \max[0, r(i', i)].$$

# Пример

	Мария	Петр	Ирина	Ангелина	Иван
Мария	<del>-8</del>	0	0	-7	-8
Петр	0	-8	-3	0	-5
Ирина	0	-3	-8	0	-1
Ангелина	-7	0	0	<del>-8</del>	0
Иван	-8	-5	-1	1	-8

	Мария	Петр	Ирина	Ангелина	Иван
Мария	0	-16	-12	-7	-14
Петр	-23	0	-9	-14	-17
Ирина	-23	-13	0	-14	-21
Ангелина	-16	-16	-12	1	-22
Иван	-15	-11	-11	-15	0

$$\text{-16} = \min(0, -8 + (0 + 0 + (-8)))$$

$$\text{-14} = \min(0, -8 + (-7 + 0 + 1))$$

$$1 = \max(0, -7) + \max(0, 0) + \max(0, 0) + \max(0, 1)$$

# Матрица критерия

---

После длительного повторения шагов расчёта матриц  $R$  и  $A$  (после того, как они перестанут изменяться) встаёт задача принятия решения. Оно осуществляется на основе матрицы критерия  $C$ :

$$c(i, j) = r(i, j) + a(i, j).$$

# Пример

	Мария	Петр	Ирина	Ангелина	Иван
Мария	-8	0	0	-7	-8
Петр	0	-8	-3	0	-5
Ирина	0	-3	-8	0	-1
Ангелина	-7	0	0	-8	0
Иван	-8	-5	-1	1	-8

	Мария	Петр	Ирина	Ангелина	Иван
Мария	0	-16	-12	-7	-14
Петр	-23	0	-9	-14	-17
Ирина	-23	-13	0	-14	-21
Ангелина	-16	-16	-12	1	-22
Иван	-15	-11	-11	-15	0

Максимальные значения каждой строки – это и есть признак кластера. Те, у кого это значение одинаковое, находятся в одном кластере.

	Мария	Петр	Ирина	Ангелина	Иван
Мария	-8	-16	-12	-14	-22
Петр	-23	-8	-12	-14	-22
Ирина	-23	-16	-8	-14	-22
Ангелина	-23	-16	-12	-7	-22
Иван	-23	-16	-12	-14	-8

# Недостатки алгоритма

---

1. Алгоритм имеет очень высокую сложность:

$O(N^2T)$  по времени ( $N$  – размер набора данных,  $T$  – число итераций);

$O(N^2)$  по памяти.

2. Может не сходиться к правильному решению, особенно, если есть несколько близких вариантов разбиения на кластеры.

Первую проблему решить нельзя! Вторую можно.

# Усовершенствования

---

1. Заполнять начальную матрицу с небольшим шумом. В Scikit-Learn этот шум порядка  $10^{-16}$ .
2. Использовать присваивание с экспоненциальным сглаживанием:  
$$r'_{t+1}(i, j) = \lambda r'_t(i, j) + (1 - \lambda) r_{t+1}(i, j)$$
$$a'_{t+1}(i, j) = \lambda a'_t(i, j) + (1 - \lambda) a_{t+1}(i, j)$$
$$0,5 \leq \lambda < 1 \text{ (0,5 по умолчанию)}$$
Данный фактор управляется при помощи параметра **damping**.
3. Эвристическая «подстройка» при которой в качестве центров выбираются значения, с критерием большим определенной величины (по умолчанию – это медианное значение). Данный фактор управляется при помощи параметра **preference**.



# Задача 1

---

Решим классическую задачу кластеризации. Кластеризацию ирисов Фишера [1]. Ирисы Фишера – это набор данных, собранных американским ботаником Эдгаром Андерсоном. Каждая запись данного набора состоит из длины наружной доли околоцветника или чашелистника (англ. sepal length), ширины наружной доли околоцветника или чашелистника (англ. sepal width), длины внутренней доли околоцветника или лепестка (англ. petal length), ширина внутренней доли околоцветника или лепестка (англ. petal width) и указания вида ириса (класса). Всего рассмотрено три вида ирисов: setosa, versicolor, и virginica. Первый из них линейно отделим от других.

На данной задаче часто проверяют качество методов кластеризации, сравнивая полученные результаты с реальным делением на классы (по видам ирисов).

# Ирисы Фишера на Википедии

Ирисы Фишера

Длина чашелистика ↕	Ширина чашелистика ↕	Длина лепестка ↕	Ширина лепестка ↕	Вид ириса ↕
5.1	3.5	1.4	0.2	<i>setosa</i>
4.9	3.0	1.4	0.2	<i>setosa</i>
4.7	3.2	1.3	0.2	<i>setosa</i>
4.6	3.1	1.5	0.2	<i>setosa</i>
5.0	3.6	1.4	0.2	<i>setosa</i>
5.4	3.9	1.7	0.4	<i>setosa</i>
4.6	3.4	1.4	0.3	<i>setosa</i>
5.0	3.4	1.5	0.2	<i>setosa</i>
4.4	2.9	1.4	0.2	<i>setosa</i>
4.9	3.1	1.5	0.1	<i>setosa</i>
5.4	3.7	1.5	0.2	<i>setosa</i>
4.8	3.4	1.6	0.2	<i>setosa</i>
4.8	3.0	1.4	0.1	<i>setosa</i>



*Iris setosa*



*Iris virginica*



# Импорт данных

---

Скопируем данную таблицу в текстовый файл (для однозначности назовём его `irises.txt`). Затем импортируем его и визуально изучим.

# Импорт и подготовка данных

```
example2-3runner.py - E:\Works\Victor\Students\STDB\Term2\Lecture3\example2-3runner.py (3.7.2)
File Edit Format Run Options Window Help
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
plt.get_current_fig_manager().window.wm_geometry('1400x750+50+50')

pd.set_option('display.max_columns', 2000)
pd.set_option('display.width', 2000)

table0 = pd.read_excel("../Lecture6/irises.xlsx")
table = table0.copy()

from sklearn import preprocessing
scaler_std = preprocessing.StandardScaler()
x = scaler_std.fit_transform(table[['sepal_length', 'sepal_width',
                                     'petal_length', 'petal_width']])
table[['sepal_length', 'sepal_width',
        'petal_length', 'petal_width']] = x

print(table)
```

Ln: 15 Col: 48

# Импорт и подготовка данных (текстом)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.get_current_fig_manager().window.wm_geometry('1400x750+50+50')
pd.set_option('display.max_columns', 2000)
pd.set_option('display.width', 2000)
table0 = pd.read_excel("../Lec6/irises.xlsx")
table = table0.copy()
from sklearn import preprocessing
scaler_std = preprocessing.StandardScaler()
x = scaler_std.fit_transform(table[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']])
table[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']] = x
print(table)
```

# Результат

```
*Python 3.7.2 Shell*
File Edit Shell Debug Options Window Help
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:\Works\Victor\Students\STDB\Term2\Lec3\example2-3.py =====
      sepal_length  sepal_width  petal_length  petal_width  class_label
0      -0.900681      1.019004      -1.340227      -1.315444      setosa
1      -1.143017      -0.131979      -1.340227      -1.315444      setosa
2      -1.385353      0.328414      -1.397064      -1.315444      setosa
3      -1.506521      0.098217      -1.283389      -1.315444      setosa
4      -1.021849      1.249201      -1.340227      -1.315444      setosa
5      -0.537178      1.939791      -1.169714      -1.052180      setosa
6      -1.506521      0.788808      -1.340227      -1.183812      setosa
7      -1.021849      0.788808      -1.283389      -1.315444      setosa
8      -1.748856      -0.362176      -1.340227      -1.315444      setosa
Ln: 9 Col: 0
```

# Подготовка данных

---

Для эффективного решения задачи кластеризации необходимо убрать доминирование одних переменных над другими за счёт разницы абсолютных значений. Обычно для этого выполняют процедуру стандартизации данных. Данная задача в Python решается при помощи класса `StandardScaler` модуля `preprocessing` библиотеки `Scikit-Learn`.

Стандартизацию мы уже сделали сразу после импорта данных.

# Стандартизация данных

---

Стандартизация данных – это такое биективное отображение данных из пространства действительных чисел в пространство действительных чисел, при котором данные оказываются распределёнными вокруг 0 со стандартным отклонением 1:

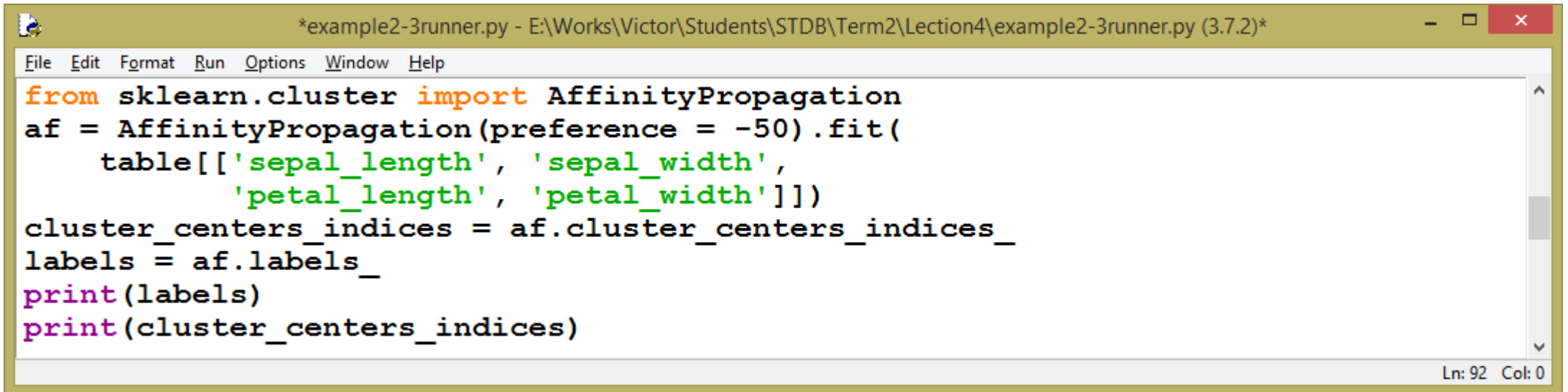
$$x' = \frac{x - M_x}{\sigma_x},$$

где  $M_x$  – математическое ожидание (среднее арифметическое) величины  $x$ , а  $\sigma_x$  – стандартное отклонение величины  $x$ .



# Affinity Propagation в Python

Для применения метода Affinity Propagation в Python используется класс AffinityPropagation модуля cluster библиотеки Scikit-Learn [3].

A screenshot of a Python script editor window titled '\*example2-3runner.py - E:\Works\Victor\Students\STDB\Term2\Lecture4\example2-3runner.py (3.7.2)\*'. The window contains a Python script that imports AffinityPropagation from sklearn.cluster, creates an instance with preference = -50, fits it to a table of features (sepal\_length, sepal\_width, petal\_length, petal\_width), and prints the resulting cluster\_centers\_indices and labels. The script is as follows:

```
from sklearn.cluster import AffinityPropagation
af = AffinityPropagation(preference = -50).fit(
    table[['sepal_length', 'sepal_width',
           'petal_length', 'petal_width']])
cluster_centers_indices = af.cluster_centers_indices_
labels = af.labels_
print(labels)
print(cluster_centers_indices)
```

The status bar at the bottom right indicates 'Ln: 92 Col: 0'.

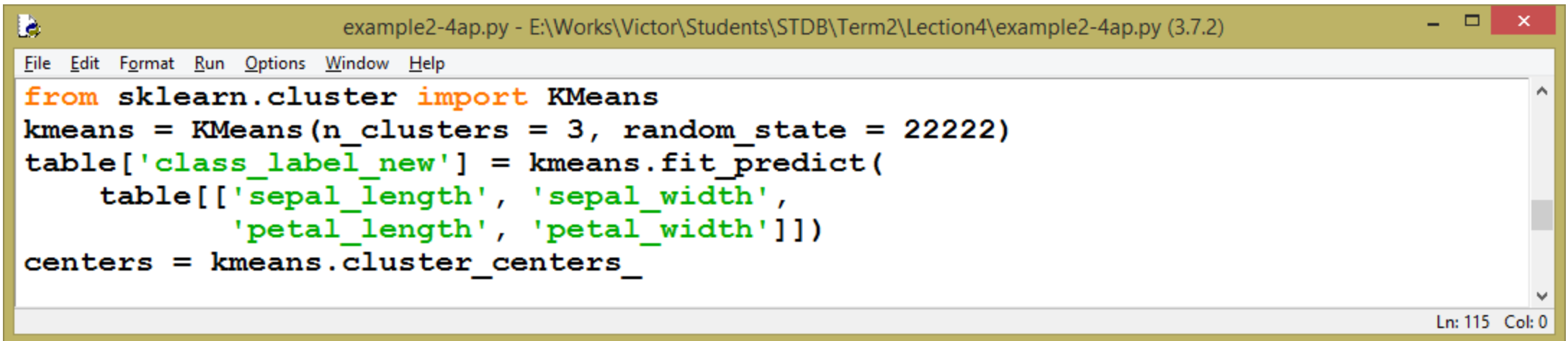
# Affinity Propagation в Python (текстом)

---

```
from sklearn.cluster import AffinityPropagation
af = AffinityPropagation(preference = -50).fit(
    table[['sepal_length', 'sepal_width',
           'petal_length', 'petal_width']])
cluster_centers_indices = af.cluster_centers_indices_
labels = af.labels_
print(labels)
print(cluster_centers_indices)
```

# Сравним с K-means

---

A screenshot of a Python IDE window titled 'example2-4ap.py - E:\Works\Victor\Students\STDB\Term2\Lecture4\example2-4ap.py (3.7.2)'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The main text area contains the following Python code:

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 3, random_state = 22222)
table['class_label_new'] = kmeans.fit_predict(
    table[['sepal_length', 'sepal_width',
            'petal_length', 'petal_width']])
centers = kmeans.cluster_centers_
```

The status bar at the bottom right shows 'Ln: 115 Col: 0'.

# Сравним с K-means (текстом)

---

```
from sklearn.cluster import KMeans  
kmeans = KMeans(n_clusters = 3, random_state = 22222)  
table['class_label_new'] = kmeans.fit_predict(  
    table[['sepal_length', 'sepal_width',  
           'petal_length', 'petal_width']])  
centers = kmeans.cluster_centers_
```

# Карты цветов и заголовки для рисования

```
example2-4ap.py - E:\Works\Victor\Students\STDB\Term2\Lection4\example2-4ap.py (3.7.2)
File Edit Format Run Options Window Help
titles = ['sepal length', 'sepal width',
          'petal length', 'petal width']
colors_map = {'virginica': 'red',
              'setosa': 'green', 'versicolor': 'blue'}
table['color'] = table['class_label']
table['color'] = table['color'].apply(
    lambda x: colors_map[x])

colors_map_new = {0: 'red', 1: 'green', 2: 'blue'}
table['color_km'] = table['class_label_new']
table['color_km'] = table['color_km'].apply(
    lambda x: colors_map_new[x])
table['color_af'] = labels
table['color_af'] = table['color_af'].apply(
    lambda x: colors_map_new[x])
y = table.iloc[:, 1]
x = table.iloc[:, 2]
```

Ln: 151 Col: 20

# Карты цветов и заголовки для рисования (текстом)

---

```
titles = ['sepal length', 'sepal width', 'petal length', 'petal width']
colors_map = {'virginica': 'red', 'setosa': 'green', 'versicolor': 'blue'}
table['color'] = table['class_label']
table['color'] = table['color'].apply(lambda x: colors_map[x])
colors_map_new = {0: 'red', 1: 'green', 2: 'blue'}
table['color_km'] = table['class_label_new']
table['color_km'] = table['color_km'].apply(lambda x: colors_map_new[x])
table['color_af'] = labels
table['color_af'] = table['color_af'].apply(lambda x: colors_map_new[x])
y = table.iloc[:, 1]
x = table.iloc[:, 2]
```

# Рисуем 3 карты-проекции

```
example2-4ap.py - E:\Works\Victor\Students\STDB\Term2\Lesson4\example2-4ap.py (3.7.2)
File Edit Format Run Options Window Help

ax = plt.subplot(1, 3, 1)
ax.set_title("K-means", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color_km'],
            s = 10)
plt.scatter(centers[0][2], centers[0][1],
            c = ['lime'], s = 200, marker = 'X')
plt.scatter(centers[1][2], centers[1][1],
            c = ['lime'], s = 200, marker = 'X')
plt.scatter(centers[2][2], centers[2][1],
            c = ['lime'], s = 200, marker = 'X')
ax = plt.subplot(1, 3, 2)
ax.set_title("Real", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color'], s = 10)
ax = plt.subplot(1, 3, 3)
ax.set_title("AP", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color_af'], s = 10)
plt.scatter(table.iloc[cluster_centers_indices[0], 2],
            table.iloc[cluster_centers_indices[0], 1],
            c = ['lime'], s = 200, marker = 'X')
plt.scatter(table.iloc[cluster_centers_indices[1], 2],
            table.iloc[cluster_centers_indices[1], 1],
            c = ['lime'], s = 200, marker = 'X')
plt.scatter(table.iloc[cluster_centers_indices[2], 2],
            table.iloc[cluster_centers_indices[2], 1],
            c = ['lime'], s = 200, marker = 'X')

plt.show()
```

Ln: 152 Col: 0

# Рисуем 3 карты-проекции (текстом)

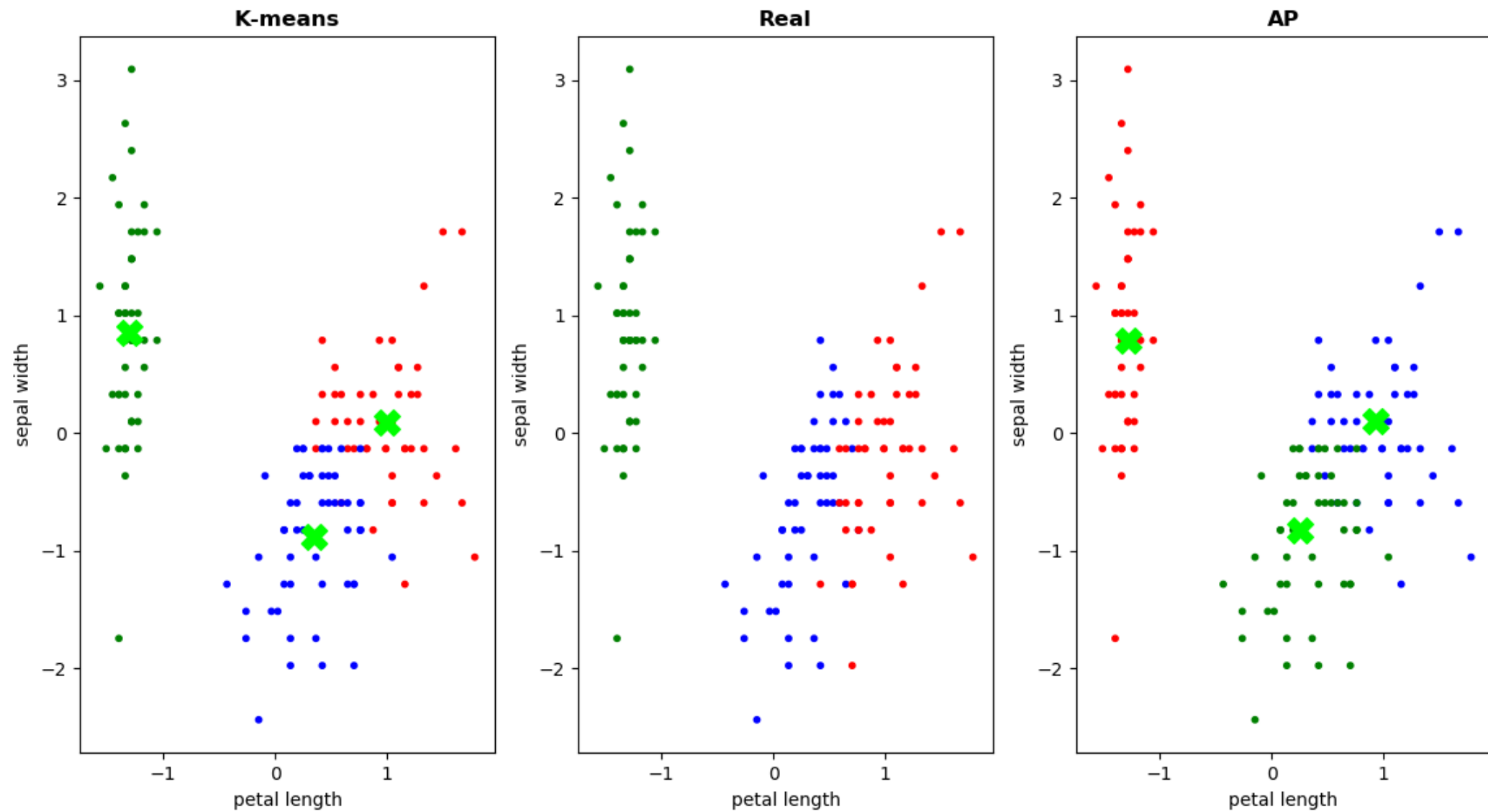
---

```
ax = plt.subplot(1, 3, 1)
ax.set_title("K-means", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color_km'], s = 10)
plt.scatter(centers[0][2], centers[0][1], c = ['lime'], s = 200, marker = 'X')
plt.scatter(centers[1][2], centers[1][1], c = ['lime'], s = 200, marker = 'X')
plt.scatter(centers[2][2], centers[2][1], c = ['lime'], s = 200, marker = 'X')
ax = plt.subplot(1, 3, 2)
ax.set_title("Real", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color'], s = 10)
```

```
ax = plt.subplot(1, 3, 3)
ax.set_title("AP", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color_af'], s = 10)
plt.scatter(table.iloc[cluster_centers_indices[0], 2],
            table.iloc[cluster_centers_indices[0], 1], c = ['lime'], s = 200, marker = 'X')
plt.scatter(table.iloc[cluster_centers_indices[1], 2],
            table.iloc[cluster_centers_indices[1], 1], c = ['lime'], s = 200, marker = 'X')
plt.scatter(table.iloc[cluster_centers_indices[2], 2],
            table.iloc[cluster_centers_indices[2], 1], c = ['lime'], s = 200, marker = 'X')
plt.show()
```



Figure 1



# Часть 2

---

DBSCAN

# DBSCAN

---

DBSCAN расшифровывается как Density-based spatial clustering of applications with noise или по-русски плотностной алгоритм пространственной кластеризации с присутствием шума.

Как видно из названия, алгоритм выполняет кластеризацию, исходя из плотности наблюдений. При этом данный алгоритм изначально рассчитан на данные, в которых есть кластеры (области с высокой плотностью) и выбросы (области с низкой плотностью). Поэтому помимо самих кластеров алгоритм выявляет ещё и точки выбросов.

# Основная идея

---

Основная идея алгоритм хорошо представлена в [5]. Мы используем похожую аналогию для объяснения.

Представим что у нас есть зал, в котором проходит праздник. Люди каким-то образом размещены в зале. Кто-то общается в компании, кто-то танцует вместе с другими, кто-то играет в игры, ну а кто-то бродит по двое или по одному. Попробуем выявить в зале «толпы». Толпой будем называть ситуацию, когда несколько людей находятся друг от друга на небольшом расстоянии.

Возникает две метрики и одна функция.

Функция расстояния между двумя объектами (людьми) –  $\rho(x, y)$ .

Ограничение окрестности элемента, определяющую близкие объекты (людей) –  $\varepsilon$ .

Минимальное число людей в толпе (иначе это не толпа) –  $m$ .

# Основная идея

---

Тогда всех людей, для которых в пределах  $\varepsilon$  находится (по расстоянию  $\rho$ ) не менее  $m$  других людей, будем считать находящимися в центре толпы и помечать зелёным цветом.

Всех людей, для которых в пределах  $\varepsilon$  находится (по расстоянию  $\rho$ ) менее  $m$  людей и при этом менее  $m$ , но больше 0 «зелёных» людей, будем считать находящимися на границе толпы и помечать жёлтым цветом.

Всех людей, для которых в пределах  $\varepsilon$  находится (по расстоянию  $\rho$ ) менее  $m$  людей и ни одного «зелёного» человека, будем считать выбросами и помечать красным.

# Формальные определения

---

Все элементы  $y$  для которых расстояние  $\rho(x, y) \leq \varepsilon$  образуют  $\varepsilon$ -окрестность  $x$  ( $E(x)$ ).

Корневым объектом степени  $m$  называется объект,  $E(x)$  которого содержит не менее  $m$  элементов.

Объект  $p$  непосредственно плотно-достижим из объекта  $q$ , если  $p \in E(q)$  и  $q$  – корневой объект.

Объект  $p$  плотно-достижим из объекта  $q$ , если  $\exists p_1, p_2, \dots, p_n$ , такие что  $p_1 = q$ ,  $p_n = p$  и для любого  $i \in \{1, n\}$  выполняется  $p_{i+1}$  непосредственно достижим из  $p_i$ .

Используя данные определения можно построить простейший итеративный алгоритм, размечающий всех людей.

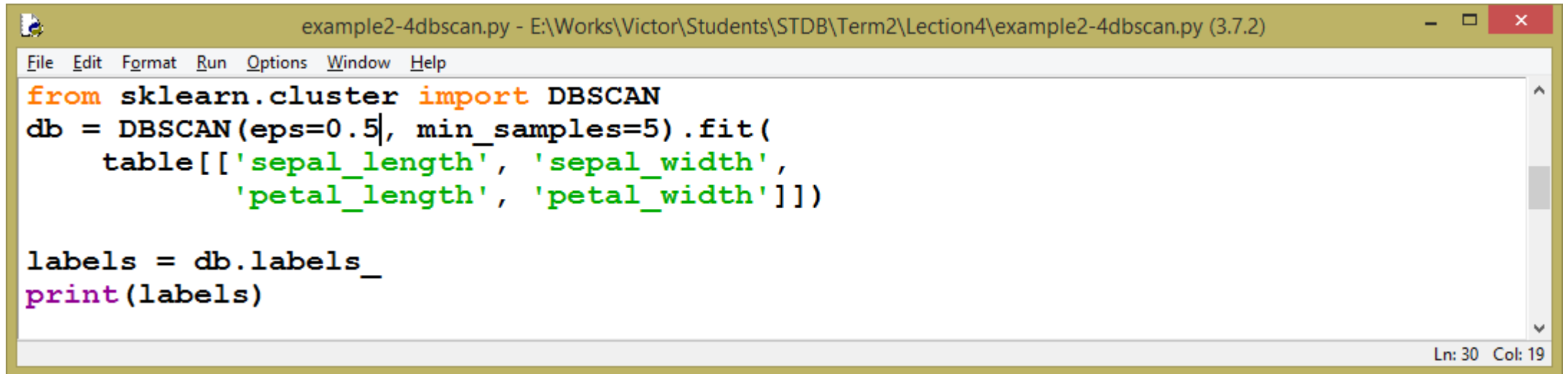
# Сложность

---

В идеальной ситуации временная сложность алгоритма  $O(N)$ . В худшей ситуации  $O(N^2)$ , в среднем  $O(N \log_2 N)$ .

# DBSCAN в Python

Для применения метода DBSCAN в Python используется класс DBSCAN модуля cluster библиотеки Scikit-Learn [3].

A screenshot of a Python script editor window titled 'example2-4dbscan.py - E:\Works\Victor\Students\STDB\Term2\Lection4\example2-4dbscan.py (3.7.2)'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The code is as follows:

```
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.5, min_samples=5).fit(
    table[['sepal_length', 'sepal_width',
           'petal_length', 'petal_width']])

labels = db.labels_
print(labels)
```

The status bar at the bottom right shows 'Ln: 30 Col: 19'.



# DBSCAN в Python (текстом)

---

```
from sklearn.cluster import DBSCAN

db = DBSCAN(eps=0.5, min_samples=5).fit(
    table[['sepal_length', 'sepal_width',
           'petal_length', 'petal_width']])

labels = db.labels_
print(labels)
```

# Сравним с K-means

---

Код запуска алгоритма K-means у нас уже был в лекции ранее. Но вот при визуализации сравнения двух алгоритмов надо помнить, что DBSCAN помимо кластеров выявляет ещё и выбросы. Выбросы помечаются меткой -1.

Кроме того, DBSCAN не выявляет центры кластеров. Поэтому код визуализации немного поменяется.

# Карты цветов и заголовки для рисования

```
example2-4dbscan.py - E:\Works\Victor\Students\STDB\Term2\Lecture4\example2-4dbscan.py (3.7.2)
File Edit Format Run Options Window Help

titles = ['sepal length', 'sepal width',
          'petal length', 'petal width']
colors_map = {'virginica': 'red',
              'setosa': 'green', 'versicolor': 'blue'}
table['color'] = table['class_label']
table['color'] = table['color'].apply(
    lambda x: colors_map[x])

colors_map_new = {0: 'red', 1: 'green', 2: 'blue', -1: 'lime'}
table['color_km'] = table['class_label_new']
table['color_km'] = table['color_km'].apply(
    lambda x: colors_map_new[x])
table['color_db'] = labels
table['color_db'] = table['color_db'].apply(
    lambda x: colors_map_new[x])
y = table.iloc[:, 1]
x = table.iloc[:, 2]
```

Ln: 32 Col: 43

# Карты цветов и заголовки для рисования (текстом)

---

```
titles = ['sepal length', 'sepal width', 'petal length', 'petal width']
colors_map = {'virginica': 'red', 'setosa': 'green', 'versicolor': 'blue', -1: 'lime'}
table['color'] = table['class_label']
table['color'] = table['color'].apply(lambda x: colors_map[x])
colors_map_new = {0: 'red', 1: 'green', 2: 'blue'}
table['color_km'] = table['class_label_new']
table['color_km'] = table['color_km'].apply(lambda x: colors_map_new[x])
table['color_db'] = labels
table['color_db'] = table['color_db'].apply(lambda x: colors_map_new[x])
y = table.iloc[:, 1]
x = table.iloc[:, 2]
```

# Рисуем 3 карты-проекции

```
example2-4dbscan.py - E:\Works\Victor\Students\STDB\Term2\Lecture4\example2-4dbscan.py (3.7.2)
File Edit Format Run Options Window Help

ax = plt.subplot(1, 3, 1)
ax.set_title("K-means", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color_km'],
            s = 10)
plt.scatter(centers[0][2], centers[0][1],
            c = ['lime'], s = 200, marker = 'X')
plt.scatter(centers[1][2], centers[1][1],
            c = ['lime'], s = 200, marker = 'X')
plt.scatter(centers[2][2], centers[2][1],
            c = ['lime'], s = 200, marker = 'X')
ax = plt.subplot(1, 3, 2)
ax.set_title("Real", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color'], s = 10)
ax = plt.subplot(1, 3, 3)
ax.set_title("DBSCAN", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color_db'], s = 10)

plt.show()
```

Ln: 81 Col: 0

# Рисуем 3 карты-проекции (текстом)

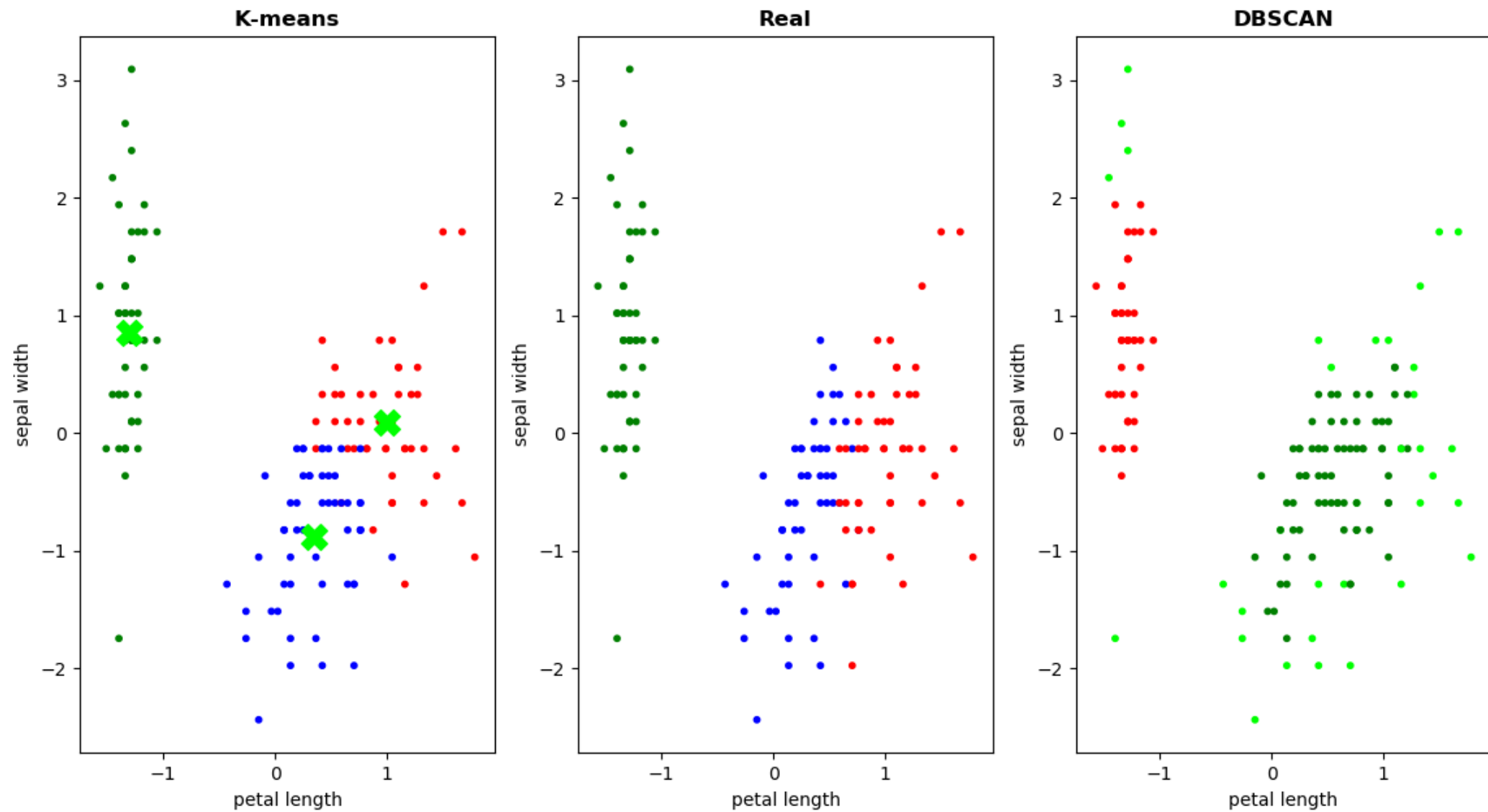
---

```
ax = plt.subplot(1, 3, 1)
ax.set_title("K-means", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color_km'],
            s = 10)
plt.scatter(centers[0][2], centers[0][1],
            c = ['lime'], s = 200, marker = 'X')
plt.scatter(centers[1][2], centers[1][1],
            c = ['lime'], s = 200, marker = 'X')
plt.scatter(centers[2][2], centers[2][1],
            c = ['lime'], s = 200, marker = 'X')
```

```
ax = plt.subplot(1, 3, 2)
ax.set_title("Real", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color'], s = 10)
ax = plt.subplot(1, 3, 3)
ax.set_title("DBSCAN", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color_db'], s = 10)

plt.show()
```

Figure 1



# Оценка числа кластеров

---

Для выявления необходимого числа кластеров с помощью метода DBSCAN используются такие же подходы, как и в случае метода K-means. Например, метод силуэта или локтя. Само число кластеров изменяется за счёт изменения параметров **eps** и **min\_samples**.



# Часть 3

---

## СПЕКТРАЛЬНАЯ КЛАСТЕРИЗАЦИЯ

# Спектральная кластеризация

---

Спектральная кластеризация (Spectral Clustering) – это алгоритм, выполняющий кластеризацию на основе матрицы близости наблюдений.

Данный алгоритм обычно очень эффективен для задач с неклассической формой кластеров. То есть в ситуациях, когда кластеры сильно отличаются от выпуклых оболочек.

# Основная идея

---

Основная идея алгоритма довольно проста. Мы рассчитываем матрицу близости наблюдений. Она может быть рассчитана также, как и в алгоритме Affinity Propagation или другими методами. Наиболее часто используются радиальные базисные функции (RBF). Можно также использовать известные нам полиномы и сигмоиды.

Затем в данной матрице вычисляются собственные значения (спектры). Анализ собственных значений позволяет поделить наблюдения на кластеры.

Таким образом, суть алгоритма сводится к численным методам нахождения собственных значений матрицы.

# Spectral Clustering в Python

---

Для применения метода Spectral Clustering в Python используется класс SpectralClustering модуля cluster библиотеки Scikit-Learn [7].

# Spectral Clustering в Python

```
example2-4sc.py - E:\Works\Victor\Students\STDB\Term2\Lecture4\example2-4sc.py (3.7.2)
File Edit Format Run Options Window Help

from sklearn.cluster import SpectralClustering
sc1 = SpectralClustering(n_clusters = 3, affinity = 'rbf',
                        gamma = 0.9, random_state = 22222).fit(
    table[['sepal_length', 'sepal_width',
           'petal_length', 'petal_width']])

labels1 = sc1.labels_
print(labels1)

sc2 = SpectralClustering(n_clusters = 3, affinity = 'poly',
                        degree = 3, random_state = 22222).fit(
    table[['sepal_length', 'sepal_width',
           'petal_length', 'petal_width']])

labels2 = sc2.labels_
print(labels2)
```

Ln: 36 Col: 13

# Spectral Clustering в Python (текстом)

---

```
from sklearn.cluster import SpectralClustering

sc1 = SpectralClustering(n_clusters = 3, affinity = 'rbf',
                        gamma = 0.9, random_state = 22222).fit(
    table[['sepal_length', 'sepal_width',
           'petal_length', 'petal_width']])
labels1 = sc1.labels_
print(labels1)

sc2 = SpectralClustering(n_clusters = 3, affinity = 'poly',
                        degree = 3, random_state = 22222).fit(
    table[['sepal_length', 'sepal_width',
           'petal_length', 'petal_width']])
labels2 = sc2.labels_
print(labels2)
```

# Сравним с K-means

---

Код запуска алгоритма K-means у нас уже был в лекции ранее. Не будем его повторять на слайдах. А вот при визуализации сравним спектральный анализ сразу двух разных ядер.

# Карты цветов и заголовки для рисования

```
example2-4sc.py - E:\Works\Victor\Students\STDB\Term2\Lecture4\example2-4sc.py (3.7.2)
File Edit Format Run Options Window Help

titles = ['sepal length', 'sepal width',
          'petal length', 'petal width']
colors_map = {'virginica': 'red',
              'setosa': 'green', 'versicolor': 'blue'}
table['color'] = table['class_label']
table['color'] = table['color'].apply(
    lambda x: colors_map[x])

colors_map_new = {0: 'red', 1: 'green', 2: 'blue'}
table['color_km'] = table['class_label_new']
table['color_km'] = table['color_km'].apply(
    lambda x: colors_map_new[x])
table['color_sc1'] = labels1
table['color_sc1'] = table['color_sc1'].apply(
    lambda x: colors_map_new[x])
table['color_sc2'] = labels2
table['color_sc2'] = table['color_sc2'].apply(
    lambda x: colors_map_new[x])
y = table.iloc[:, 1]
x = table.iloc[:, 2]
```

Ln: 44 Col: 0



# Карты цветов и заголовки (текстом)

---

```
titles = ['sepal length', 'sepal width',  
         'petal length', 'petal width']  
colors_map = {'virginica': 'red',  
             'setosa': 'green', 'versicolor': 'blue'}  
table['color'] = table['class_label']  
table['color'] = table['color'].apply(  
    lambda x: colors_map[x])  
  
colors_map_new = {0: 'red', 1: 'green', 2: 'blue'}
```

```
table['color_km'] = table['class_label_new']  
table['color_km'] = table['color_km'].apply(  
    lambda x: colors_map_new[x])  
table['color_sc1'] = labels1  
table['color_sc1'] = table['color_sc1'].apply(  
    lambda x: colors_map_new[x])  
table['color_sc2'] = labels2  
table['color_sc2'] = table['color_sc2'].apply(  
    lambda x: colors_map_new[x])  
y = table.iloc[:, 1]  
x = table.iloc[:, 2]
```

# Рисуем 4 карты-проекции

```
example2-4sc.py - E:\Works\Victor\Students\STDB\Term2\Lecture4\example2-4sc.py (3.7.2)
File Edit Format Run Options Window Help

ax = plt.subplot(1, 4, 1)
ax.set_title("K-means", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color_km'],
            s = 10)
plt.scatter(centers[0][2], centers[0][1],
            c = ['lime'], s = 200, marker = 'X')
plt.scatter(centers[1][2], centers[1][1],
            c = ['lime'], s = 200, marker = 'X')
plt.scatter(centers[2][2], centers[2][1],
            c = ['lime'], s = 200, marker = 'X')
ax = plt.subplot(1, 4, 2)
ax.set_title("Real", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color'], s = 10)
ax = plt.subplot(1, 4, 3)
ax.set_title("Spectral RBF", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color_sc1'], s = 10)
ax = plt.subplot(1, 4, 4)
ax.set_title("Spectral Poly", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color_sc2'], s = 10)

plt.show()
```

# Рисуем 4 карты-проекции (текстом)

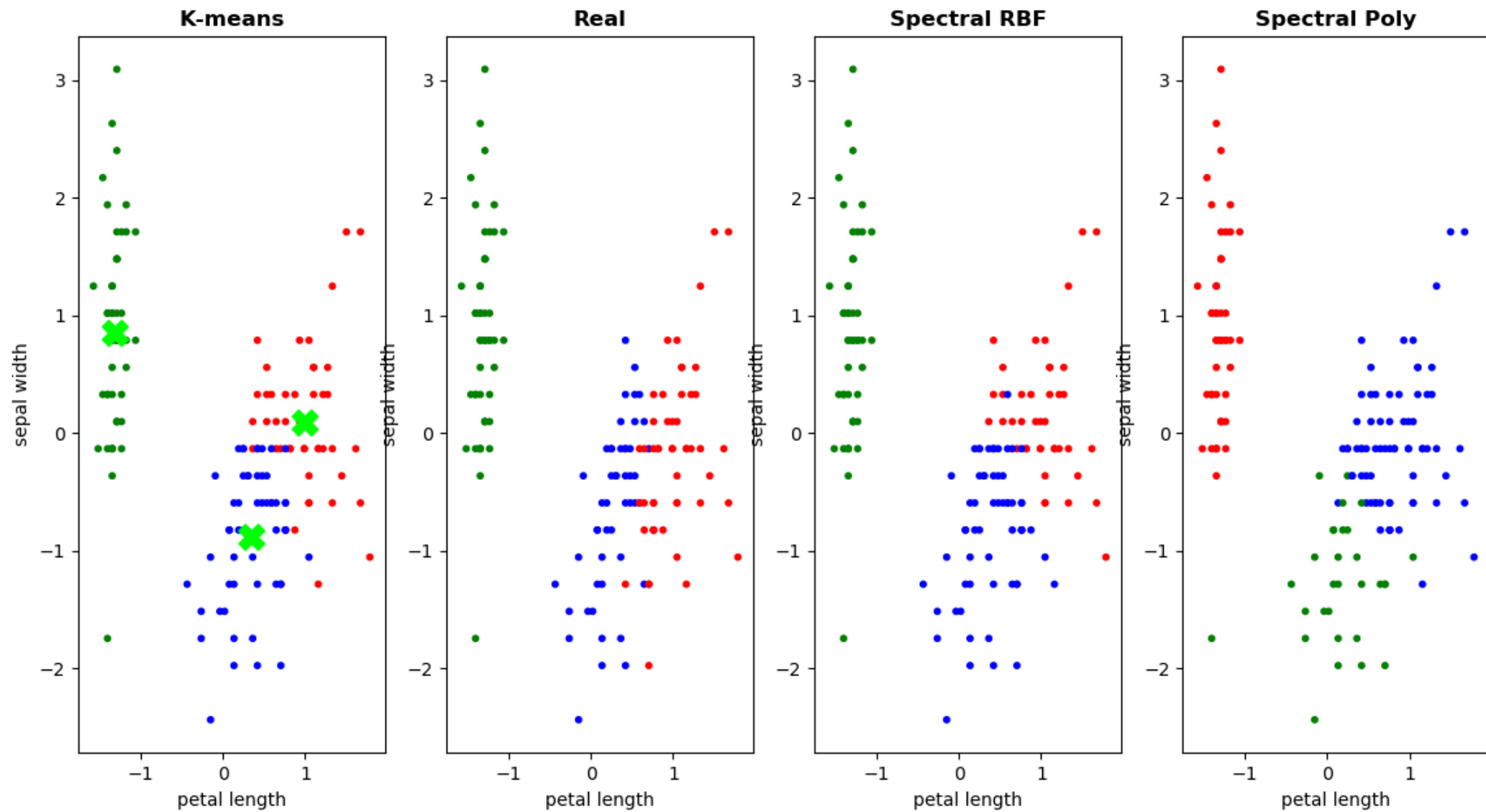
---

```
ax = plt.subplot(1, 4, 1)
ax.set_title("K-means", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color_km'], s = 10)
plt.scatter(centers[0][2], centers[0][1], c = ['lime'], s = 200, marker = 'X')
plt.scatter(centers[1][2], centers[1][1], c = ['lime'], s = 200, marker = 'X')
plt.scatter(centers[2][2], centers[2][1], c = ['lime'], s = 200, marker = 'X')
ax = plt.subplot(1, 4, 2)
ax.set_title("Real", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
```

```
plt.scatter(x, y, c = table['color'], s = 10)
ax = plt.subplot(1, 4, 3)
ax.set_title("Spectral RBF", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color_sc1'], s = 10)
ax = plt.subplot(1, 4, 4)
ax.set_title("Spectral Poly", fontweight='bold')
ax.set_xlabel(titles[2])
ax.set_ylabel(titles[1])
plt.scatter(x, y, c = table['color_sc2'], s = 10)

plt.show()
```

Figure 1



# Оценка числа кластеров

---

Для выявления необходимого числа кластеров с помощью метода Spectral Clustering используются такие же подходы, как и в случае метода K-means. Например, метод силуэта или локтя.

# Интернет ресурсы и литература

---

1. [https://ru.wikipedia.org/wiki/%D0%98%D1%80%D0%B8%D1%81%D1%8B\\_%D0%A4%D0%B8%D1%88%D0%B5%D1%80%D0%B0](https://ru.wikipedia.org/wiki/%D0%98%D1%80%D0%B8%D1%81%D1%8B_%D0%A4%D0%B8%D1%88%D0%B5%D1%80%D0%B0) – википедия об ирисах Фишера.
2. <https://towardsdatascience.com/unsupervised-machine-learning-affinity-propagation-algorithm-explained-d1fef85f22c8>
3. <https://scikit-learn.org/stable/modules/clustering.html#affinity-propagation>
4. <https://habr.com/ru/post/321216/>
5. <https://habr.com/ru/post/322034/>
6. <https://scikit-learn.org/stable/modules/clustering.html#dbscan>
7. <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html#sklearn.cluster.SpectralClustering>