**Лабораторная работа 11.**

**K-means. Практика**

**Для работы используйте юпитер ноутбук ClusteringAlgorithms**

Выберем оптимальное количество **кластеров** для наших данных: в нашем случае, это семь кластеров, один из которых в центре. Добавим случайным образом семь **центроид**. (центроиды — предполагаемые центры будущих кластеров) Для каждой точки мы можем посчитать, к какому центроиду она ближе и окрасить в соответствующий цвет. Перенесём центроид в центр выборки, к которой мы его отнесли: то есть, расположим его так. чтобы расстояния от объектов кластера до центроида были как можно меньше.. Повторим эти шаги, пока алгоритм не сойдётся.

Рассмотрим, с какими **наборами данных**будут работать наши алгоритмы. Первый набор данных — три хорошо разделимых кластера, второй — три кластера с добавлением шума, далее — кластеры с ленточной структурой. В нижнем ряду сложные случаи, когда кластеры вписываются друг в друга или не имеют структуру.

Посмотрим, как организован алгоритм K-means в **библиотеке Sklearn**. Импортируем данные, при инциализации можем указать ожидаемое число кластеров, способ инициализации центроид, максимальное число итераций.

**from** sklearn.cluster import KMeans

k\_means = KMeans(n\_clusters=8, #количество кластеров

**init**='k-means++', # 'k-means++', 'random', numpy.array

max\_iter=300 #максимальное количество итераций

)

Как же будет работать алгоритм на наших данных? Вызываем **метод k\_means.fit()**, который будет обучаться на нашем наборе данных. Далее строим графики для каждого из наших датасетов.

# создаём массив пар – датасета и соответствующих для него параметров алгоритма - количества кластеров

datasets\_params\_list = [

(blobs, {'n\_clusters': 3}),

(varied, {'n\_clusters': 3}),

(aniso, {'n\_clusters': 3}),

(noisy\_circles, {'n\_clusters': 2}),

(noisy\_moons, {'n\_clusters': 2}),

(no\_structure, {'n\_clusters': 3})]

**for** i, (X, k\_means\_params) **in** enumerate(datasets\_params\_list, start=1):

X = StandardScaler().fit\_transform(X)

k\_means = KMeans(n\_clusters=k\_means\_params['n\_clusters'])

k\_means.fit(X)

y\_pred = k\_means.labels\_.astype(np.int)

plt.subplot(f'23{i}')

plt.xticks([]); plt.yticks([])

colors = np.array(list(islice(cycle(['#377eb8', '#ff7f00', '#4daf4a',

'#f781bf', '#a65628', '#984ea3',

'#999999', '#e41a1c', '#dede00']),

int(max(y\_pred) + 1))))

plt.scatter(X[:, 0], X[:, 1], color=colors[y\_pred])

Для наших задач создадим **датасет**, в котором будет три кластера, а у каждого объекта будет два признака.

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

n\_samples = 1500

dataset = datasets.make\_blobs(n\_samples=n\_samples, centers=2, center\_box=(-7.0, 7.5),

cluster\_std=[1.4, 1.7],

random\_state=42)

X\_2, \_ = datasets.make\_blobs(n\_samples=n\_samples, random\_state=170, centers=[[-4, -3]], cluster\_std=[1.9])

transformation = [[1.2, -0.8], [-0.4, 1.7]]

X\_2 = np.dot(X\_2, transformation)

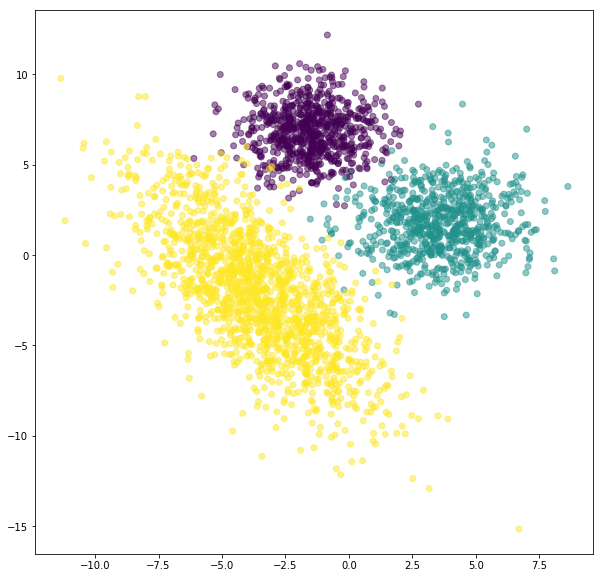
X, y = np.concatenate((dataset[0], X\_2)), np.concatenate((dataset[1], np.array([2] \* len(X\_2))))

**Визуализируем наш датасет:**

plt.rcParams['figure.figsize'] = 10, 10

plt.scatter(X[:, 0], X[:, 1], c=y, alpha=0.5)

plt.show()



Посмотрим распределение классов в датасете:

unique, counts = np.unique(y, return\_counts=True)

dict(zip(unique, counts))

{0: 750, 1: 750, 2: 1500}

В дальнейшем в задачах с использованием этого датасета при использовании переменных X и y считать, что эти переменные содержат признаки объектов и классы объектов датасета соответственно.

Объект, который выполняет кластеризацию датасета с помощью алгоритма K-means, инициализруется так:

from sklearn.cluster **import** KMeans

k\_means = KMeans()

При создании объекта можно указать следующие параметры:

* + **n\_clusters** — число кластеров, на которое алгоритм будет делить набор данных;
  + **init** — способ начальной инициализации центроидов кластров;
  + **random\_state** — фиксирует генерацию случайного числа для инициализации центроидов.

Задание 7.5.1

При каком значении **параметра init** при создании объекта K-means **центроиды классов** будут инициализированы случайным образом в пространстве объектов датасета?

Для выполнения задания необходимо ознакомиться с [документацией](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html) к реализации K-means в пакете scikit-learn.

Задание 7.5.2

Для обучения модели K-means необходимо вызвать метод fit, передав входным параметром признаки объектов датасета:

k\_means.fit(X)

Обучив, мы можем получить центроиды кластеров:

k\_means.cluster\_centers\_

И узнать, в какой кластер попал каждый из объектов:

k\_means.labels\_

Обучите модель K-means с параметрами n\_clusters=3 и random\_state=42 на признаках исходного датасета.

**Какие центроиды будут у получившихся кластеров?** Введите ответ в виде массива. Каждое число в ответе округлите до ближайшего целого. Для округления можно воспользоваться функцией numpy.round:

**import** numpy **as** np

a = [0.4, 0.6, 1.7]

print(np.round(a).astype(np.int))

# Вывод

[0 1 2]

astype(np.int) конвертирует элементы массива в целые числа.

Задание 7.5.3

Подсчитайте количество элементов в каждом из получившихся кластеров. Запишите в форму ниже три числа через пробел(без запятых!): количество элементов в кластере 0, в кластере 1 и в кластере 2. Записывайте строго в таком порядке.

Для подсчёта элементов в списке можно воспользоваться функцией numpy.unique с параметром return\_counts=True:

importnumpyasnp

a = [0, 0, 0, 1, 1, 2]

\_, counts = np.unique(a, return\_counts=True)

**for** count **in** counts:

print(count, end=' ')

# Вывод

3 2 1

Для визуализации получившегося датасета можно использовать следующий код:

k\_means\_pred = k\_means.labels\_

**plt.scatter(X[:, 0], X[:, 1], c=k\_means\_pred, alpha=0.5)**

plt.show()

**ЗАДАЧА НА РЕАЛЬНЫХ ДАННЫХ: КЛАСТЕРИЗАЦИЯ УЧЕНИКОВ СТАРШЕЙ ШКОЛЫ**

Подгрузите данные snsdata и откройте их.

В датасете для учеников предоставлены следующие признаки:

* Год выпуска
* Пол
* Возраст
* Количество друзей
* 36 ключевых слов, которые встречаются в профилe *Facebook*(интересы, сообщества, встречи)

Мы будем пытаться выделить кластеры исключительно по интересам, поэтому в первую очередь удалите все признаки, кроме ключевых слов.

Нормализуйте оставшиеся признаки с помощью StandardScaler(). Помните, что кластеризация — обучение без учителя, поэтому разделение на обучающую и тренировочную выборку не требуется.

Используйте метод *k-means* для кластеризации, количество кластеров возьмите за 9, random\_state =123.

Задание 7.5.4

Укажите номер кластера, в который попало меньше всего учеников.

Задание 7.5.4.2

В один из кластеров попали ученики с интересами music, god, dance, hair, shopping, cute, band, rock, football, church. Сколько всего учеников попали в этот кластер?

**EM-алгоритм. Практика**

**Пример работы EM-алгоритма:**

* выбираем число кластеров, которое нам кажется оптимальным, в нашем случае их два;
* выбираем**параметр распределений**;
* для каждой точки набора считаем **вероятность** принадлежности каждому распределению;
* пересчитываем параметры распределения;
* повторяем шаги: снова присваиваем точку распределению и считаем параметры.

Рассмотрим реализацию алгоритмы в библиотеке **Sklearn**. Наш алгоритм лежит в пакете **GaussianMixture**. При инициализации мы можем указать число компонентов, максимальное число итераций и способ инициализации начальных параметров.

**from** sklearn.mixture **import** GaussianMixture

em\_gm = GaussianMixture(n\_components=1,

max\_iter=100,

init\_params='kmeans' # 'kmeans’, ‘random’

)

Посмотрим, как *EM*-алгоритм будет работать с нашим набором данных. Сделаем нормализацию признаков объектов, создадим объект, который будет кластеризовать, вызовем **методы fit()** и **predict()**и построим получившиеся кластеры для наших датасетов.

datasets\_params\_list = [

(blobs, {'n\_clusters': 3}),

(varied, {'n\_clusters': 3}),

(aniso, {'n\_clusters': 3}),

(noisy\_circles, {'n\_clusters': 2}),

(noisy\_moons, {'n\_clusters': 2}),

(no\_structure, {'n\_clusters': 3})]

**for** i, (X, em\_gm\_params) **in** enumerate(datasets\_params\_list, start=1):

X = StandardScaler().fit\_transform(X)

em\_gm = GaussianMixture(n\_components=em\_gm\_params['n\_clusters'])

em\_gm.fit(X)

y\_pred = em\_gm.predict(X)

plt.subplot(f'23{i}')

plt.xticks([]); plt.yticks([])

colors = np.array(list(islice(cycle(['#377eb8', '#ff7f00', '#4daf4a',

'#f781bf', '#a65628', '#984ea3',

'#999999', '#e41a1c', '#dede00']),

int(max(y\_pred) + 1))))

plt.scatter(X[:, 0], X[:, 1], color=colors[y\_pred])

В результате простейшие кластеры разделились хорошо, обнаружилась ленточная структура, но с более сложными примерами алгоритм запутался и нашёл кластеры там, где их не должно было быть.

Для решения задач используйте**датасет**, который мы создавали в теме «K-means. Практика»:

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

n\_samples = 1500

dataset = datasets.make\_blobs(n\_samples=n\_samples, centers=2, center\_box=(-7.0, 7.5),

cluster\_std=[1.4, 1.7],

random\_state=42)

X\_2, \_ = datasets.make\_blobs(n\_samples=n\_samples, random\_state=170, centers=[[-4, -3]], cluster\_std=[1.9])

transformation = [[1.2, -0.8], [-0.4, 1.7]]

X\_2 = np.dot(X\_2, transformation)

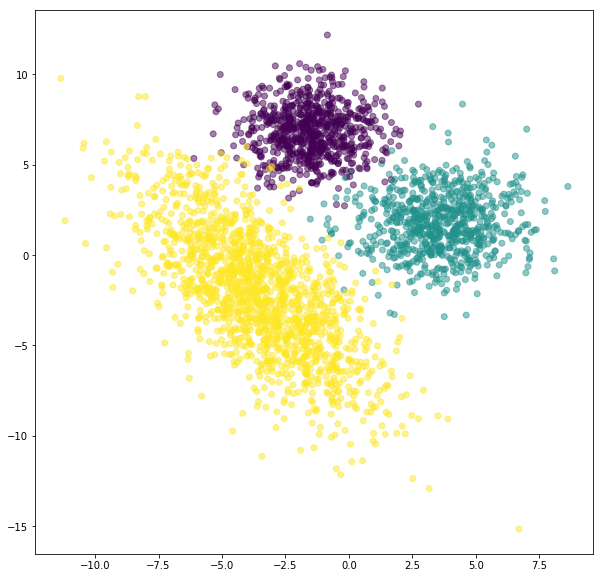
X, y = np.concatenate((dataset[0], X\_2)), np.concatenate((dataset[1], np.array([2] \* len(X\_2))))

**Визуализируем наш датасет:**

plt.rcParams['figure.figsize'] = 10, 10

plt.scatter(X[:, 0], X[:, 1], c=y, alpha=0.5)

plt.show()



Посмотрим распределение классов в датасете:

unique, counts = np.unique(y, return\_counts=True)

dict(zip(unique, counts))

{0: 750, 1: 750, 2: 1500}

При использовании переменных X и y считать, что эти переменные содержат признаки объектов и классы объектов датасета соответственно.

Объект, который выполняет кластеризацию датасета с помощью EM-алгоритма, инициализируется так:

from sklearn.mixture **import** GaussianMixture

gm = GaussianMixture()

В отличие от остальных рассматриваемых алгоритмов, EM-алгоритм лежит не в модуле **cluster**, а в модуле **mixture**. EM-алгоритм определяет **смеси распределений** в выборке, а не просто проводит кластеризацию.

Полный список параметров доступен в [документации](https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html) (англ.).

Для обучения модели **GaussianMixture** необходимо вызвать **метод fit**, передав входным параметром признаки объектов датасета:

gm.fit(X)

Получить предсказания можно с помощью **метода predict**после метода fit:

gm.fit(X)

y\_pred = gm.predict(X)

Можно совместить эти два шага при помощи **метода fit\_predict**:

y\_pred = gm.fit\_predict(X)

Обучив, мы можем получить параметры распределений кластеров, например, средние:

means = gm.means\_

Задание 7.6.1

В алгоритме *k-means* за число кластеров отвечал параметр n\_clusters. Какой параметр задаёт число кластеров в *EM*-алгоритме?

Задание 7.6.2

Обучите **модель GaussianMixture** с параметрами n\_components=3 и random\_state=42 на признаках исходного датасета.

**Какие средние значения будут у получившихся распределений кластеров?** Каждое число в ответе округлите до ближайшего целого. Для округления можно воспользоваться функцией numpy.round:

**import** numpy **as** np

a = [0.4, 0.6, 1.7]

print(np.round(a).astype(np.int))

[0 1 2]

Задание 7.6.3

Подсчитайте количество элементов в каждом из получившихся кластеров. Выведите три числа через пробел: количество элементов в кластере 0, в кластере 1 и в кластере 2. Числа вводите целые, без точек и запятых.

Для подсчёта элементов в списке можно воспользоваться **функцией numpy.unique** с параметром return\_counts=True:

importnumpyasnp

a = [0, 0, 0, 1, 1, 2]

\_, counts = np.unique(a, return\_counts=True)

**for** count **in** counts:

print(count, end=' ')

# Вывод

3 2 1

Задание 7.6.4

Возьмите данные про учеников старшей школы из прошлого урока (c такой же обработкой) и примените к ним *EM*-алгоритм , количество кластеров — 9, random\_state = 123. К какому кластеру принадлежит восьмой (по списку из изначального датасета) ученик? Введите номер кластера целым числом без точек и десятичной части.

Сколько итераций до схождения осуществил алгоритм? Вопрос относится к датасету об учениках старшей школы.

**Агломеративная кластеризация. Практика**

Будем поочередно объединять близлежащие объекты (кластеры), пока весь датасет не войдёт в кластеры. Посмотрим, как реализуется агломеративная кластеризация в пакете **Sklearn.**При **инициализации** можно задать количество кластеров, функцию расстояния, способ определения ближайшего соседа:

**from** sklearn.cluster **import** AgglomerativeClustering

ac = AgglomerativeClustering(n\_clusters=2,

affinity='euclidean', # “euclidean”, “l1”, “l2”, “manhattan”,

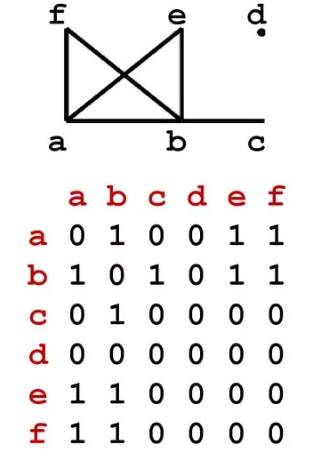
# “cosine”, or “precomputed”

linkage='ward', # “ward”, “complete”, “average”, “single”

)

Посмотрим, как агломеративная кластеризация будет работать с нашими наборами данных. Мы будем варьировать только **число кластеров**. Будем также использовать **матрицу смежности**, для которой нужен **параметр n\_neighbors**, то есть число ближайших соседей.

**Примечание**. Матрица смежности — это матрица, которая соответствует графу, и которая отражает, с какими соседями объект соединен или нет. Например, на рисунке ниже объект а соединен с *е*, *f*, *b*, поэтому на пересечении  строки(столбца) *а*со столбцами (строками) *e*, *f*, *b*стоят единички.



Вместо единичек, которые показывают, есть связь или нет, могут быть другие характеристики. Например, расстояние между объектами.

Мы используем здесь матрицу смежности по той причине,что агломеративная кластеризация строится в качестве одного из вариантов именно по такой матрице, которая показывает, сколько ближайших соседей находится рядом с объектом.

**import** warnings

**from** sklearn.neighbors **import** kneighbors\_graph

datasets\_params\_list = [

(blobs, {'n\_clusters': 3, 'n\_neighbors': 10}),

(varied, {'n\_clusters': 3, 'n\_neighbors': 2}),

(aniso, {'n\_clusters': 3, 'n\_neighbors': 2}),

(noisy\_circles, {'n\_clusters': 2, 'n\_neighbors': 10}),

(noisy\_moons, {'n\_clusters': 2, 'n\_neighbors': 10}),

(no\_structure, {'n\_clusters': 3, 'n\_neighbors': 10})]

**for** i, (X, ac\_params) **in** enumerate(datasets\_params\_list, start=1):

X = StandardScaler().fit\_transform(X)

# строим матрицу смежности

connectivity = kneighbors\_graph(X,

n\_neighbors=ac\_params['n\_neighbors'],

include\_self=False)

# делаем матрицу смежности симметричной

connectivity = 0.5 \* (connectivity + connectivity.T)

ac = AgglomerativeClustering(n\_clusters=ac\_params['n\_clusters'],

linkage='average',

connectivity=connectivity)

**with** warnings.catch\_warnings():

warnings.filterwarnings(

"ignore",

message="Error",

category=UserWarning)

ac.fit(X)

y\_pred = ac.labels\_.astype(np.int)

plt.subplot(f'23{i}')

plt.xticks([]); plt.yticks([])

colors = np.array(list(islice(cycle(['#377eb8', '#ff7f00', '#4daf4a',

'#f781bf', '#a65628', '#984ea3',

'#999999', '#e41a1c', '#dede00']),

int(max(y\_pred) + 1))))

plt.scatter(X[:, 0], X[:, 1], color=colors[y\_pred])

В алгоритме мы сначала нормализуем признаки объектов, а затем строим матрицу смежности. Так мы получаем **расстояние между объектами**датасета. Далее создаём объект агломеративной кластеризации и передаём в него матрицу. Обучаем алгоритм,**игнорируем предупреждения** и строим предсказания.

В результате агломеративная кластеризация справилась лучше, чем EM-алгоритм со сложными кластерами, с ленточными, наоборот, хуже. При отсутствии кластерной структуры агломеративная кластеризация более явно определяет это.

Для решения задач используйте**датасет**, который мы создавали в теме «K-means. Практика»:

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

n\_samples = 1500

dataset = datasets.make\_blobs(n\_samples=n\_samples, centers=2, center\_box=(-7.0, 7.5),

cluster\_std=[1.4, 1.7],

random\_state=42)

X\_2, \_ = datasets.make\_blobs(n\_samples=n\_samples, random\_state=170, centers=[[-4, -3]], cluster\_std=[1.9])

transformation = [[1.2, -0.8], [-0.4, 1.7]]

X\_2 = np.dot(X\_2, transformation)

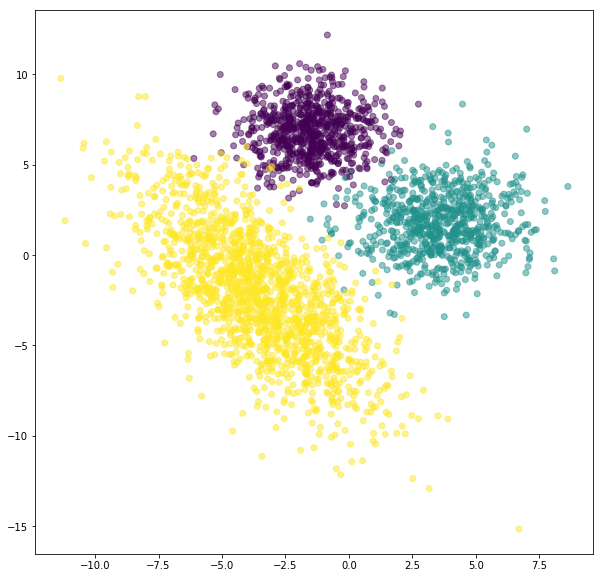
X, y = np.concatenate((dataset[0], X\_2)), np.concatenate((dataset[1], np.array([2] \* len(X\_2))))

**Визуализируем наш датасет:**

plt.rcParams['figure.figsize'] = 10, 10

plt.scatter(X[:, 0], X[:, 1], c=y, alpha=0.5)

plt.show()



Посмотрим распределение классов в датасете:

unique, counts = np.unique(y, return\_counts=True)

dict(zip(unique, counts))

{0: 750, 1: 750, 2: 1500}

При использовании переменных  и  считать, что эти переменные содержат признаки объектов и классы объектов датасета соответственно.

Объект, который выполняет агломеративную кластеризацию датасета, инициализируется следующим образом:

from sklearn.cluster **import** AgglomerativeClustering

ac = AgglomerativeClustering()

В отличие от *k-means*,*AgglomerativeClustering*не имеет **параметра random\_state**.

Полный список параметров доступен в [документации](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html) (англ.).

Обучение модели *AgglomerativeClustering* проходит аналогично обучению модели *k-means*:

ac.fit(X)

Обучив, мы можем узнать, в какой кластер попал каждый из объектов:

ac.labels\_

Задание 7.7.1

При каком значении параметра linkage при создании объекта *AgglomerativeClustering* критерием объединения кластеров будет среднее расстояние между объектами объединяемых кластеров?

Обучите модель *AgglomerativeClustering* с параметром n\_clusters=3 на признаках исходного датасета. Напишите число листьев в иерархическом дереве, полученном при обучении:

Задание 7.7.2

Подсчитайте количество элементов в каждом из получившихся кластеров. Запишите в форму ниже три числа через пробел: количество элементов в кластере 0, в кластере 1 и в кластере 2.

Для подсчёта элементов в списке можно воспользоваться функцией numpy.unique с параметром return\_counts=True:

**import** numpy **as** np

a = [0, 0, 0, 1, 1, 2]

counts = np.unique(a, return\_counts=True)

**for** count **in** counts:

print(count, end=' ')

# Вывод

3 2 1

Задание 7.7.3

При создании модели AgglomerativeClustering можно указать **матрицу смежности**, передав её в параметр connectivity. Построить матрицу смежности можно с помощью следующего кода:

**from** sklearn.neighbors **import** kneighbors\_graph

connectivity = kneighbors\_graph(X, n\_neighbors=6, include\_self=False)

connectivity = 0.5 \* (connectivity + connectivity.T)

В переменной connectivity будет хранится матрица смежности для ненаправленного графа на основе датасета.

Обучите **модель AgglomerativeClustering** с параметром n\_clusters=3 и составленной матрицей смежности на признаках исходного датасета. Подсчитайте количество элементов в каждом из получившихся кластеров. Выведите три числа через пробел: количество элементов в кластере 0, в кластере 1 и в кластере 2.

**Подсказка:** При решении данной задачи стандартизация должна быть выключена.

Задание 7.7.4

Постройте **дендрограмму** с помощью пакета scipy:

**from** scipy.cluster.hierarchy **import** dendrogram, linkage

# подготовим данные для построения дендрограммы

# ещё один способ выполнить агломеративную кластеризацию

Z = linkage(X, "ward")

# строим дендрограмму

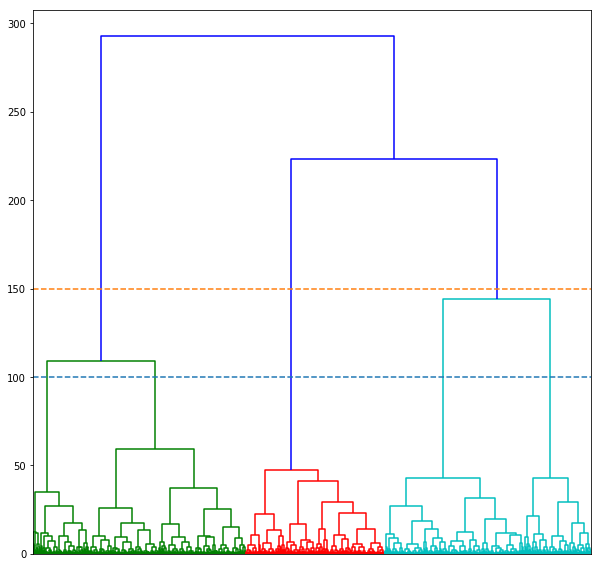
dendrogram(Z, leaf\_rotation=90.)

На выходе должно получиться изображение иерархического дерева.

Постройте дендрограмму на исходном наборе данных. **Сколько получится кластеров, если граничным расстоянием для разделения кластеров взять 150.**

  Задание 7.7.5

Сколько кластеров получится, если граничным расстоянием для разделения кластеров взять 100?



Рассмотрим [данные](https://lms-cdn.skillfactory.ru/assets/courseware/v1/674de4a149becd117e895dc5bf1e36f4/asset-v1:Skillfactory+DST-WEEKLY-2.0+08JULY2020+type@asset+block/food.txt) food , содержащие информацию о составе различных продуктов.

Попробуем построить на них иерархическую кластеризацию:

from scipy.cluster.hierarchy **import** dendrogram, linkage

Z = linkage(X, method='average', metric='euclidean')

names = df.Name.values

dend = dendrogram(Z, color\_threshold=0, labels=names,

orientation='left')

from scipy.cluster.hierarchy **import** fcluster

t = 2.3

labels = fcluster(Z, t, criterion='distance')

В алгоритме данные нужно **нормализовать**.

Задание 7.7.6

Сколько всего кластеров получилось для продуктов?  нет ответа

Введите номер кластера, в который попали продукты с содержанием кальция от 150 до 160.

Введите номер кластера, в который попал продукт с максимальной жирностью.

**DBSCAN. Практика**

Из набора данных случайным образом выбираем точку и смотрим её **окрестности**. Если там не менее четырёх точек, то мы помечаем скопление кластером и далее для каждой точки из окрестности рассматриваем такое же окружение. Повторяем до момента окончания точек в наборе.

Посмотрим, как реализуется алгоритм **DBSCAN**в пакете **Sklearn.**При **инициализации** можно задать размер окрестностей **eps** и минимальное число точек в этой окрестности **min\_sample**:

from sklearn.cluster **import** DBSCAN

dbscan = DBSCAN(eps=0.5,

min\_samples=5)

Посмотрим, как *DBSCAN* будет работать с нашими наборами данных. Мы будем варьировать только **размер окрестностей:**

datasets\_params\_list = [

(blobs, {'eps': 0.3}),

(varied, {'eps': 0.18}),

(aniso, {'eps': 0.184}),

(noisy\_circles, {'eps': 0.3}),

(noisy\_moons, {'eps': 0.3}),

(no\_structure, {'eps': 0.3})]

**for** i, (X, dbscan\_params) **in** enumerate(datasets\_params\_list, start=1):

X = StandardScaler().fit\_transform(X)

dbscan = DBSCAN(eps=dbscan\_params['eps'])

dbscan.fit(X)

y\_pred = dbscan.labels\_.astype(np.int)

plt.subplot(f'23{i}')

plt.xticks([]); plt.yticks([])

colors = np.array(list(islice(cycle(['#377eb8', '#ff7f00', '#4daf4a',

'#f781bf', '#a65628', '#984ea3',

'#999999', '#e41a1c', '#dede00']),

int(max(y\_pred) + 1))))

# чёрным цветом отметим выбросы

colors = np.append(colors, ["#000000"])

plt.scatter(X[:, 0], X[:, 1], color=colors[y\_pred])

В алгоритме мы сначала нормализуем данные, затем создаём объект *DBSCAN*, обучаем его при помощи **метода .fit()**и получаем предсказания. Выбросы будем отмечать чёрным цветом.

В результате при хорошо разделимых кластерах *DBSCAN* справляется отлично, он определил **сложные случаи** и ленточную структуру, правда, нашёл **лишние выбросы**. Это единственный алгоритм, по сравнению с предыдущими, который определил, что кластерной структуры в последнем датасете нет. Добавление **шума** смущает алгоритм: в одном из примеров кластеры склеились между собой.

Для решения задач используйте**датасет**, который мы создавали в теме «K-means. Практика»:

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

n\_samples = 1500

dataset = datasets.make\_blobs(n\_samples=n\_samples, centers=2, center\_box=(-7.0, 7.5),

cluster\_std=[1.4, 1.7],

random\_state=42)

X\_2, \_ = datasets.make\_blobs(n\_samples=n\_samples, random\_state=170, centers=[[-4, -3]], cluster\_std=[1.9])

transformation = [[1.2, -0.8], [-0.4, 1.7]]

X\_2 = np.dot(X\_2, transformation)

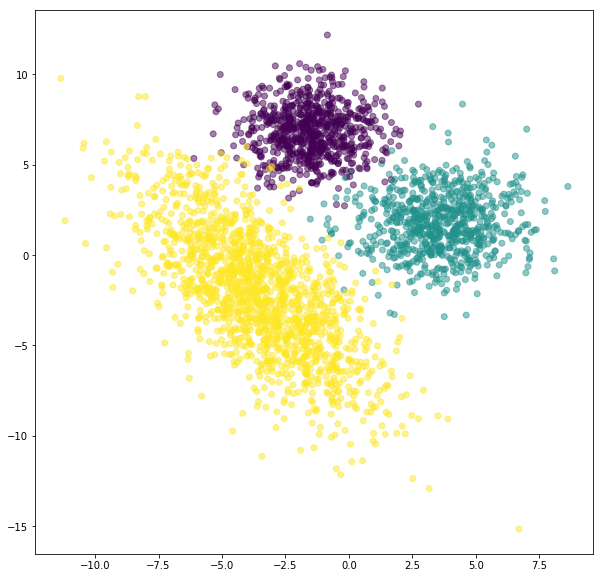
X, y = np.concatenate((dataset[0], X\_2)), np.concatenate((dataset[1], np.array([2] \* len(X\_2))))

**Визуализируем наш датасет:**

plt.rcParams['figure.figsize'] = 10, 10

plt.scatter(X[:, 0], X[:, 1], c=y, alpha=0.5)

plt.show()



Посмотрим распределение классов в датасете:

unique, counts = np.unique(y, return\_counts=True)

dict(zip(unique, counts))

{0: 750, 1: 750, 2: 1500}

При использовании переменных X и y считать, что эти переменные содержат признаки объектов и классы объектов датасета соответственно.

Объект, который выполняет кластеризацию датасета с помощью алгоритма DBSCAN, инициализируется так:

from sklearn.cluster **import** DBSCAN

dbscan = DBSCAN()

В отличие от *k-means* и *AgglomerativeClustering*, класс *DBSCAN* не имеет параметра **n\_clusters**, поскольку *DBSCAN* автоматически определяет число кластеров в выборке.

Полный список параметров доступен в [документации](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html) (англ.).

Обучение модели *DBSCAN* проходит аналогично обучению модели *k-means*:

dbscan.fit(X)

Обучив, мы можем узнать, в какой кластер попал каждый из объектов:

dbscan.labels\_

Задание 7.8.1

Какой параметр DBSCAN устанавливает число объектов в окрестности, которое определяет, является ли объект основным или нет?

  Задание 7.8.2

Обучите модель DBSCAN с параметрами eps=0.9 и min\_samples=35 на признаках объектов исходного датасета. Выведите число получившихся кластеров.

Обратите внимание, что DBSCAN определяет выбросы, относя их к кластеру -1. Кластер выбросов учитывать в ответе не нужно.

Задание 7.8.3

Сколько объектов выборки было отмечено как выбросы в прошлой задаче?

Задание 7.8.4

Проверьте, как настройка параметров влияет на результат работы DBSCAN. Обучите модель DBSCAN с параметрами eps=0.8 и min\_samples=35 на признаках объектов исходного датасета.

Выведите число объектов выборки, которые были отмечены как выбросы.