

«Машинное обучение»

Методы кластеризации

Александр Дьяконов



План

Задача кластеризации, типы кластеризации

k-средних (Lloyd's algorithm) + обобщения

Affinity propagation: кластеризация сообщениями между точками

Сдвиг среднего (Mean Shift): обнаружение мод плотности

Иерархическая кластеризация (Hierarchical clustering)

Графовые методы + MST

Spectral Clustering – Спектральная кластеризация

DBSCAN = Density-Based Spatial Clustering of Applications with Noise

BIRCH = Balanced Iterative Reducing Clustering using Hierarchies

Сравнение алгоритмов кластеризации

Кластеризация

– разбиение множества объектов на группы похожих
такие группы – кластеры

неформально:

маленькие внутрикластерные расстояния
большие межкластерные расстояния

Самое важное и «скользкое»

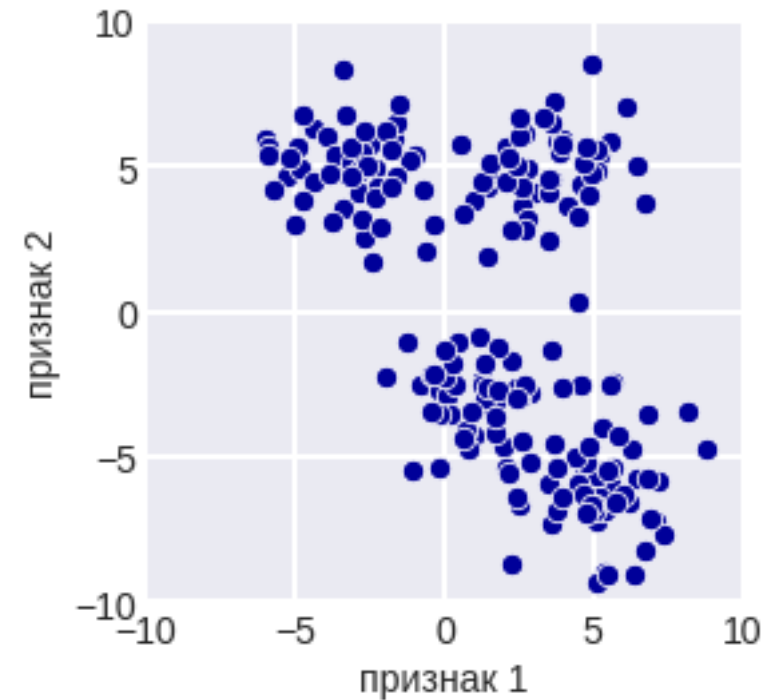
дальше предполагаем, что есть некоторая адекватная метрика!

С помощью её и будем осуществлять кластеризацию.

Входная информация для алгоритмов

- 1. (Feature-based) Признаковые описания объектов**
- 2. (Dis/similarity-based) Попарные сходства/различия**

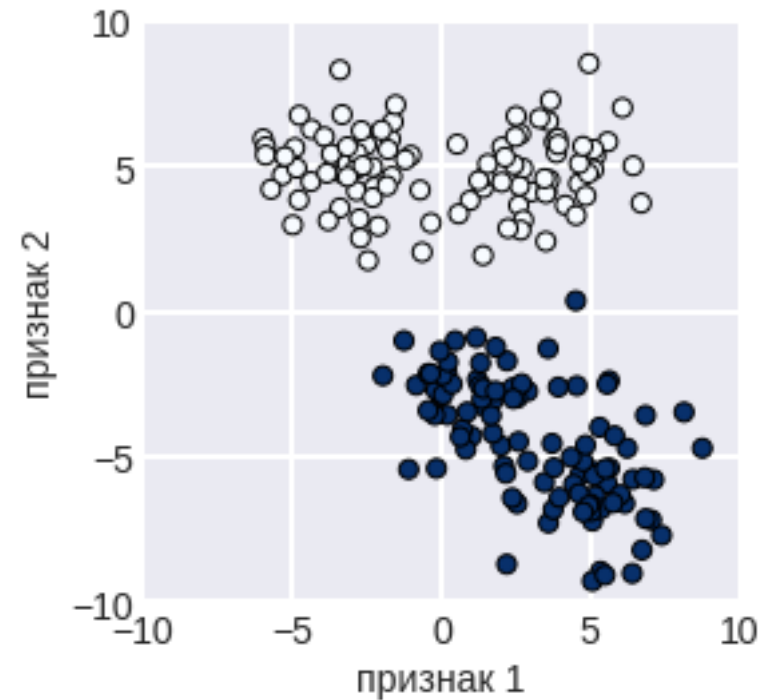
Модельная задача кластеризации



Даны объекты (без меток)

Надо разбить на группы похожих – кластеры

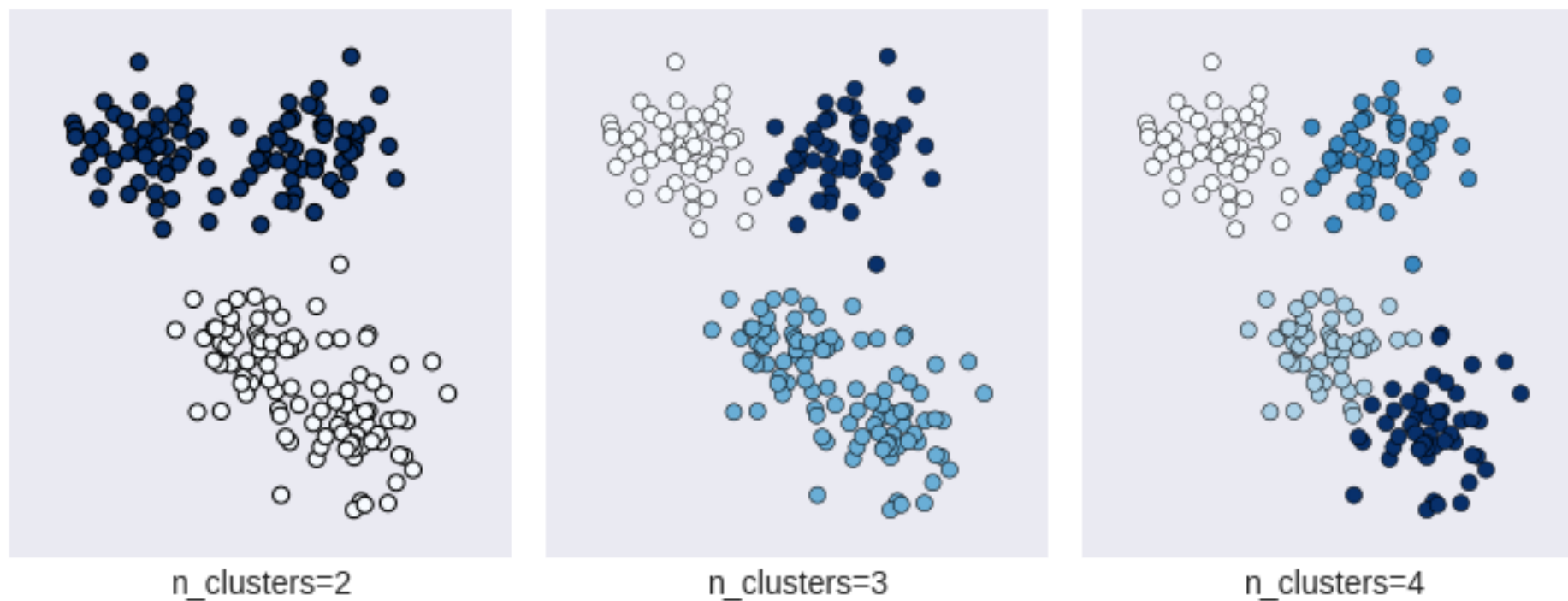
Модельная задача кластеризации



Даны объекты (без меток)

Надо разбить на группы похожих – кластеры

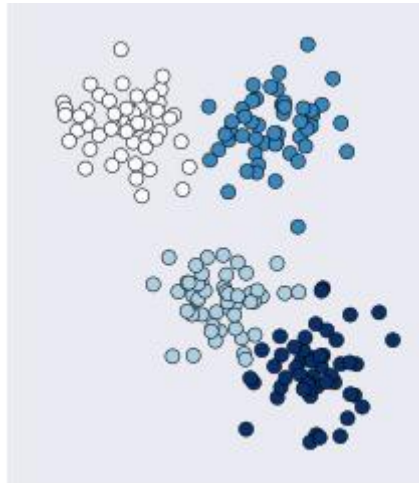
Модельная задача кластеризации



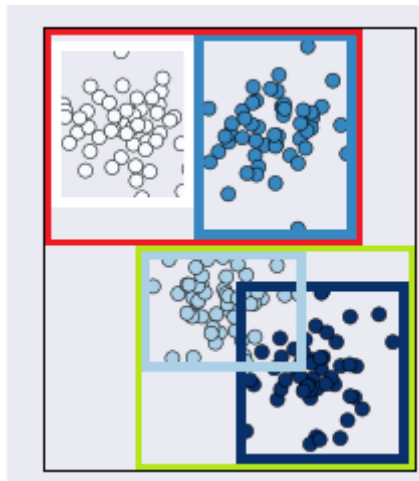
Много допустимых решений...

Нет правильного ответа!

Методы кластеризации



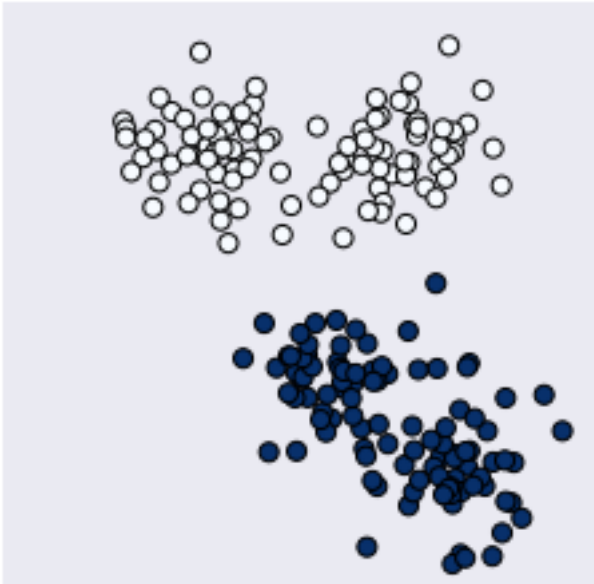
Плоские / разделяющие (Flat / Partitional) –
кластеризация на k непересекающихся кластеров



Иерархические (Hierarchical) – данные один большой кластер, далее рекурсивно «кластер = объединение подкластеров»

~ система каталогов

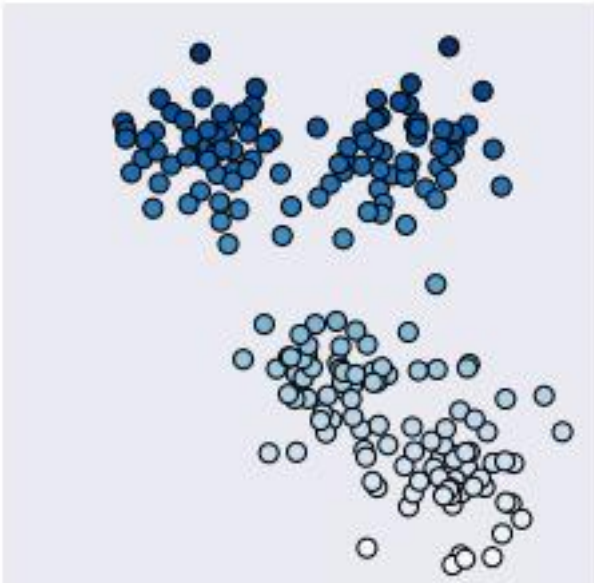
Типы кластеризации



чёткая (hard)

разбиение на непересекающиеся кластеры

$$\mathbb{X} = C_1 \cup \dots \cup C_k$$
$$i \neq j \Rightarrow C_i \cap C_j = \emptyset$$



нечёткая (мягкая, fuzzy)

определение степени принадлежности кластерам

$$x_i \rightarrow r_{it} \in [0, 1]$$
$$\forall i \in \{1, 2, \dots, m\} \sum_{t=1}^k r_{it} = 1$$

Зачем

- **кластеризация клиентов / товаров, иерархия сущностей (таксономия)**
(выработка таргетированной политики)

- **сжатие данных**

(при модерации проверять несколько представителей кластера, устранение однотипности вопросов, vector quantization – замена кластера центром)

- **сообщества клиентов**

(эффективное распространение новостей, предложение услуг)

- **анализ данных / признаков**

(классическая идея математики – факторизация)

- **важная составляющая решения других задач**

(semi-supervised, outlier detection, community detection)

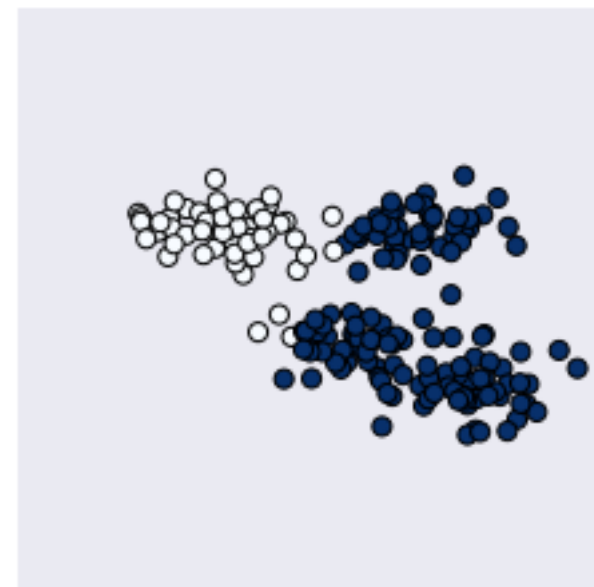
была в методах сэмплирования и много где ещё...

Редко бывает самостоятельной задачей! Но важный этап

Проблемы с расстоянием



сжали по одному признаку



сжали по второму признаку

Часто используют стандартизацию, но она может всё портить...

В кластеризации считаем, что метрика адекватна

Как делать кластеризацию?

$$\sum_t \frac{1}{|C_t|} \sum_{x_i, x_j \in C_t} \rho(x_i, x_j)^\gamma \rightarrow \min_{\{C_t\}} \text{ – NP-полная задача}$$

Пусть с каждым кластером C_t ассоциирован центр μ_t

Заменим задачу такой:

$$\sum_t \frac{1}{|C_t|} \sum_{x_i \in C_t} \rho(x_i, \mu_t)^\gamma \rightarrow \min_{\{C_t\}, \{\mu_t\}}$$

но тут больше параметров...

k-средних (Lloyd's algorithm)

$$\sum_t \frac{1}{|C_t|} \sum_{x_i \in C_t} \|x_i - \mu_t\|^2 \rightarrow \min_{\{C_t\}, \{\mu_t\}}$$

Если зафиксировать $\{C_t\}$, то оптимальное решение – центроид кластера

$$\mu_t = \frac{1}{|C_t|} \sum_{x_i \in C_t} x_i$$

Если зафиксировать $\{\mu_t\}$, то оптимальное решение – ближайшие точки

$$C_t = \{i \mid \|x_i - \mu_t\| = \min_j \|x_i - \mu_j\|\}$$

Такую минимизацию делают итеративно

k-средних (Lloyd's algorithm)**Вход:**

$$\{x_1, \dots, x_m\} \subseteq \mathbf{R}^n$$

Инициализация:

$$k \text{ центров кластеров } \{\mu_1, \dots, \mu_k\} \subseteq \mathbf{R}^n$$

случайные точки или случайные объекты
из обучающей выборки

Итерация:

повторять итерации до сходимости (пока
центры станут неподвижными)
«assignment»

**1. Каждый объект приписать к тому
кластеру, к центру которого он ближе:**

$$C_t = \{i \mid \|x_i - \mu_t\| = \min_j \|x_i - \mu_j\|\}$$

Ничьи разрешаются произвольно

2. Пересчитать центры кластеров:

$$\mu_t = \frac{1}{|C_t|} \sum_{i \in C_t} x_i$$

«update»

при неопределённости вида «деление на
ноль» оставляем центр неизменным

метод ~ координатный спуск

Итерации метода к-средних

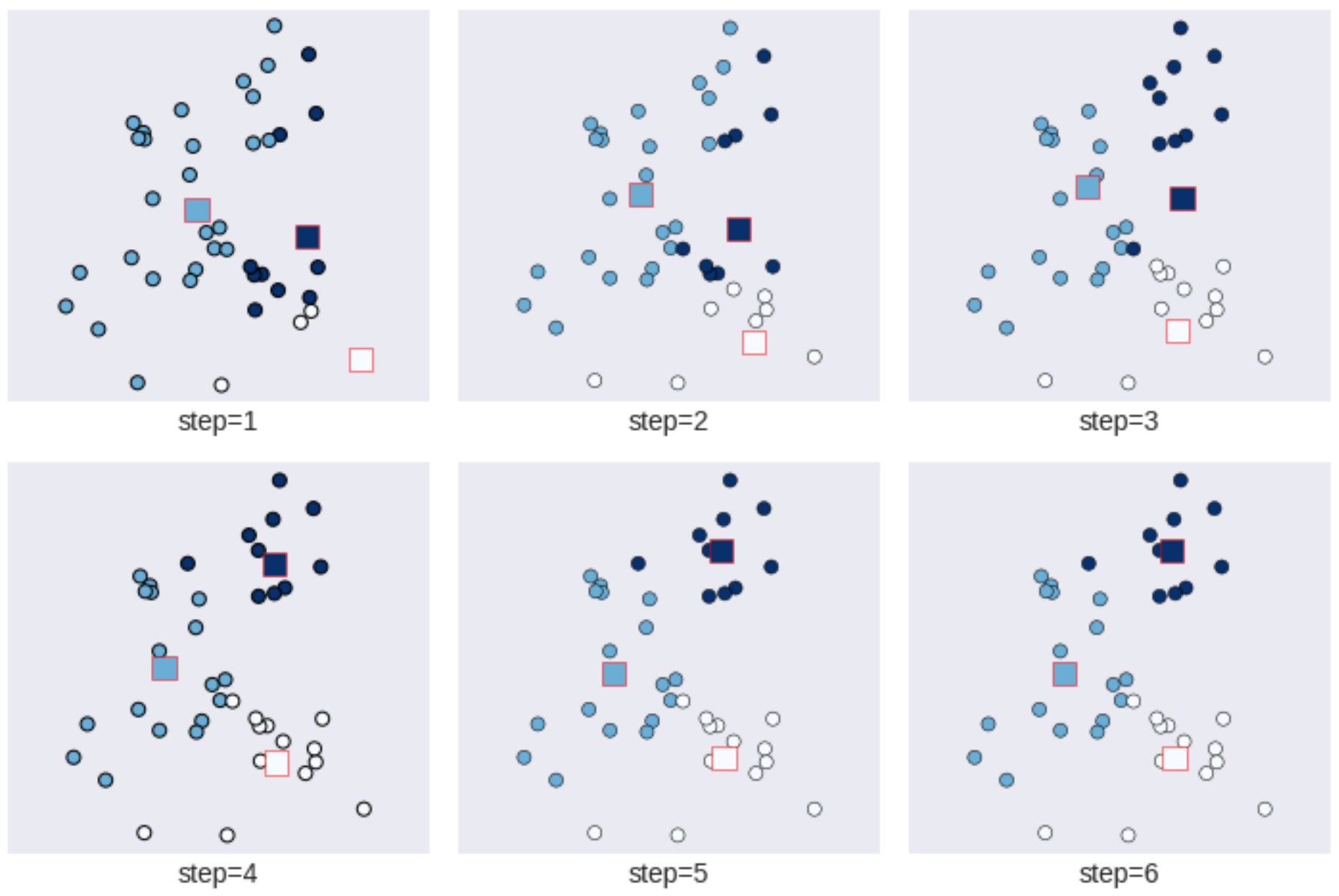
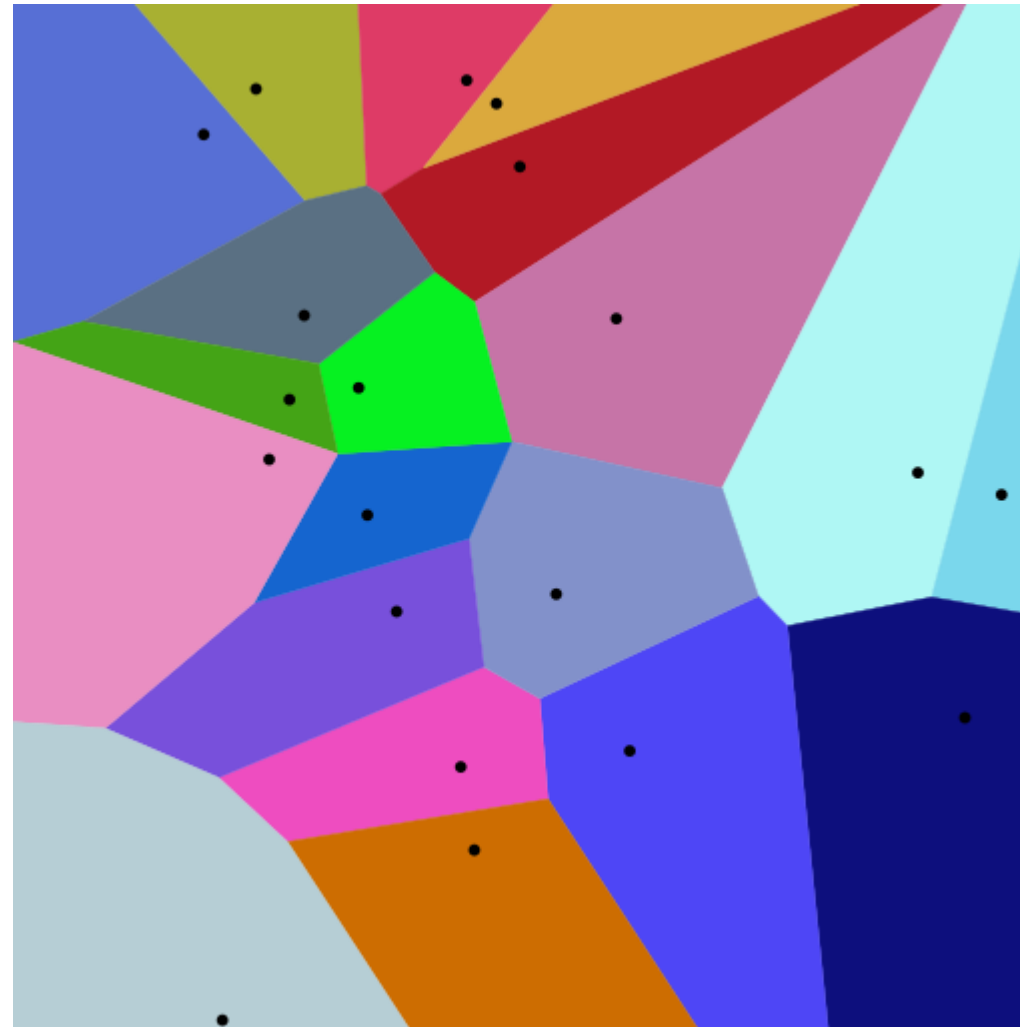


Диаграмма Вороного (Voronoi diagram)



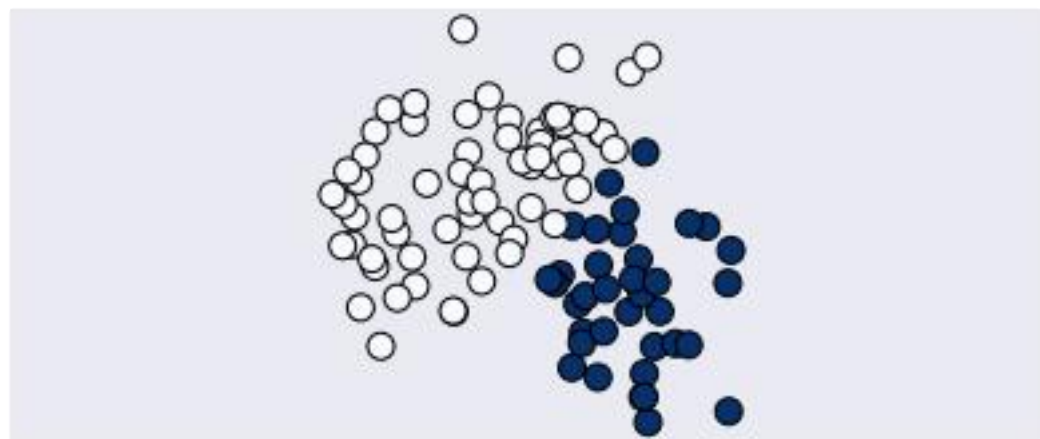
https://en.wikipedia.org/wiki/Voronoi_diagram

Код

```
# создание датасета
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=100, n_features=2, centers=3, random_state=0)

# запуск алгоритма
from sklearn.cluster import KMeans
a = KMeans(n_clusters=2, random_state=0).fit_predict(X)

# визуализация
import matplotlib.pyplot as plt
plt.scatter(X[:, 0], X[:, 1], c=a)
```



k-Means – реализация

`sklearn.cluster.KMeans`

`n_clusters` – **число кластеров**

`init` – **метод инициализации** :«k-means++», «random», ndarray

`n_init` – **сколько раз алгоритм запускается (выбирается лучший ответ)**

`max_iter` – **максимальное число итерация для каждого алгоритма**

`tol` – **порог для определения сходимости**

`precompute_distances` – **предварительное вычисление расстояний, «auto», True, False (для скорости, но требует память)**

`verbose` – **отчёт о работе**

`random_state`

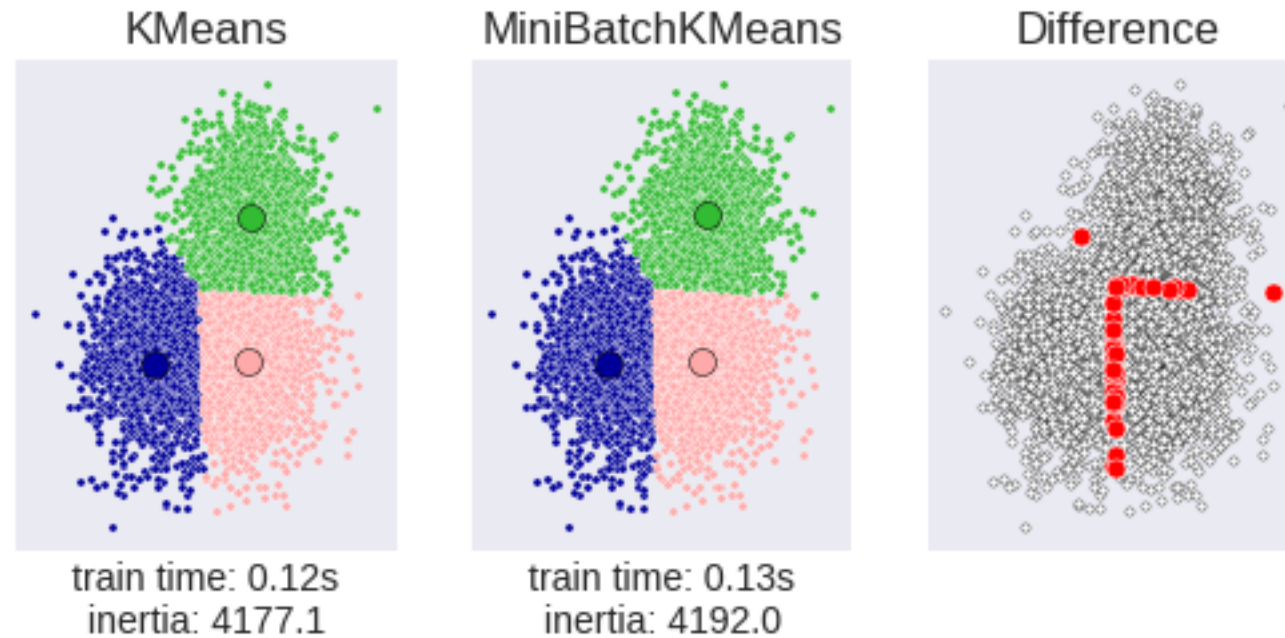
`copy_x` – **когда precompute_distances можно ли портить данные**

`n_jobs`

`algorithm` – **какой алгоритм использовать: «lloyd», «elkan» (использует неравенство треугольника, но не подходит для разреженных данных)**

k-Means с мини-батчами

`sklearn.cluster.MinibatchKMeans`



Кстати, inertia – в sklearn $\sum_{i=1}^m \min_t \|x_i - \mu_t\|^2$

D. Sculley «Web-Scale K-Means Clustering» // WWW 2010: Proceedings of the 19th Annual International World Wide Web Conference. April, 2010. <http://www.eecs.tufts.edu/~dsculley/papers/fastkmeans.pdf>

k-Means с мини-батчами

не обязательно точно решать задачу на итерации,
можно вычислять центр градиентным алгоритмом $\mu_t \leftarrow \mu_t + \eta(x_i - \mu_t)$

а можно делать это по мини-батчам...

Algorithm 1 Mini-batch k -Means.

```
1: Given:  $k$ , mini-batch size  $b$ , iterations  $t$ , data set  $X$ 
2: Initialize each  $c \in C$  with an  $x$  picked randomly from  $X$ 
3:  $v \leftarrow 0$ 
4: for  $i = 1$  to  $t$  do
5:    $M \leftarrow b$  examples picked randomly from  $X$ 
6:   for  $x \in M$  do
7:      $d[x] \leftarrow f(C, x)$  // Cache the center nearest to  $x$ 
8:   end for
9:   for  $x \in M$  do
10:     $c \leftarrow d[x]$  // Get cached center for this  $x$ 
11:     $v[c] \leftarrow v[c] + 1$  // Update per-center counts
12:     $\eta \leftarrow \frac{1}{v[c]}$  // Get per-center learning rate
13:     $c \leftarrow (1 - \eta)c + \eta x$  // Take gradient step
14:   end for
15: end for
```

k-Means с мини-батчами – реализация`sklearn.cluster.MinibatchKMeans``batch_size` – размер батча`max_iter` – ограничение на число итераций`init` – инициализация («k-means++» или «random»)`compute_labels` – вычислять ли метки и качество в конце (True)`max_no_improvement` – для остановки`init_size` – число сэмплов для ускорения инициализации

(начальные центры будут вычислены на этой подвыборке)

`n_init` – число случайных инициализаций

метод запускается один раз (!) на лучшей инициализации

`reassignment_ratio` – максимальная доля вычислений для пересчёта центра (0.01)

Разная начальная инициализация



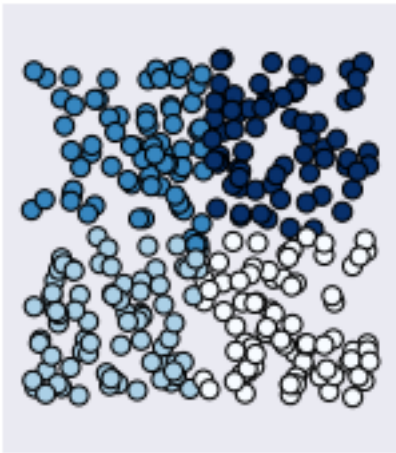
random_state=0



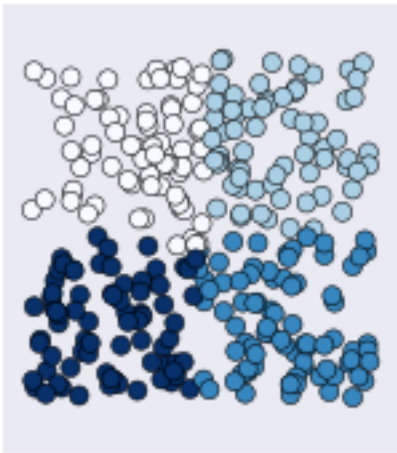
random_state=1



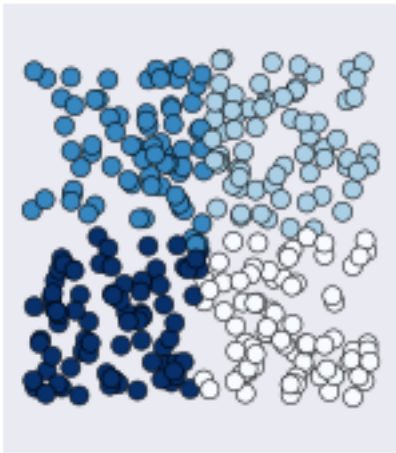
random_state=2



random_state=0

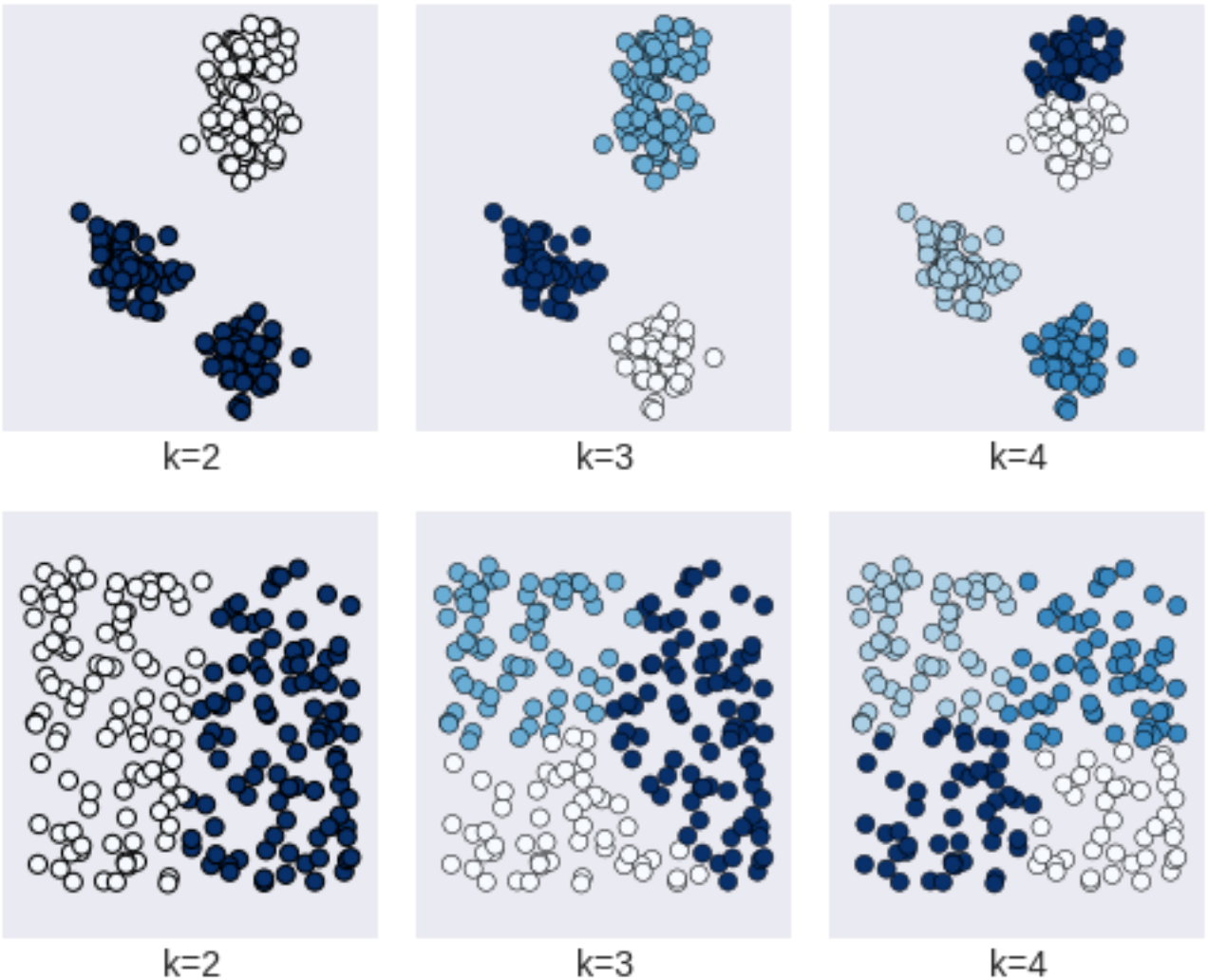


random_state=1

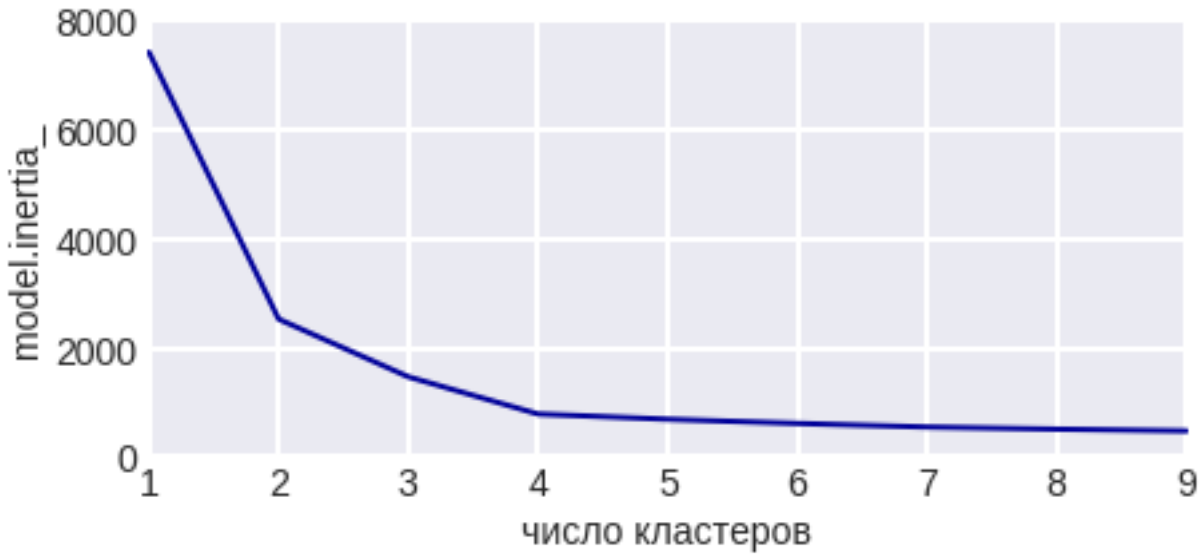


random_state=2

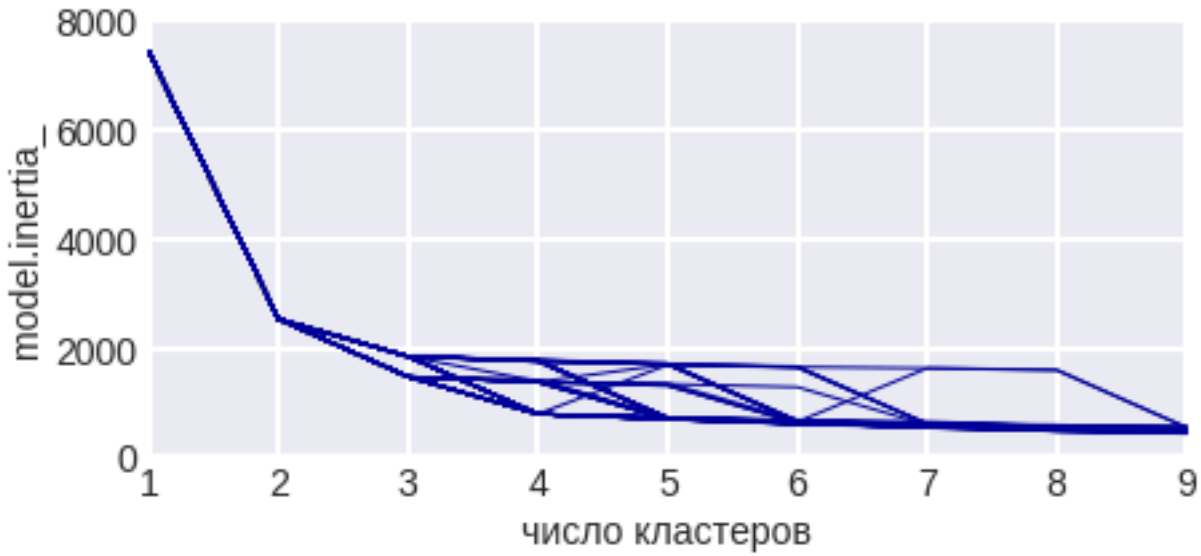
Разное число кластеров k



Вариант для выбора k



выбираем лучшую инициализацию из 5



много запусков

Сходимость метода

– теоретически может сходиться долго, но на практике быстро

средняя сложность $O(kmn_{\text{iter}})$

худший случай $O(m^{(k+2/n)})$

D. Arthur, S. Vassilvitskii «How slow is the k-means method?» SoCG2006

– сходимость к локальному минимуму

– чувствителен к начальной инициализации (могут получаться разные ответы):

- качество кластеризации
- скорость сходимости

– нужен выбор k

Реализация на практике и обобщения

несколько разных инициализаций и перезапусков

эвристика: 1й центр – один из объектов, 2й – максимально удалённый от первого, 3й – максимально удалённый от предыдущих центров и т.д.

на самом деле тут вероятностный выбор, поэтому разные перезапуски
– **k-means++**

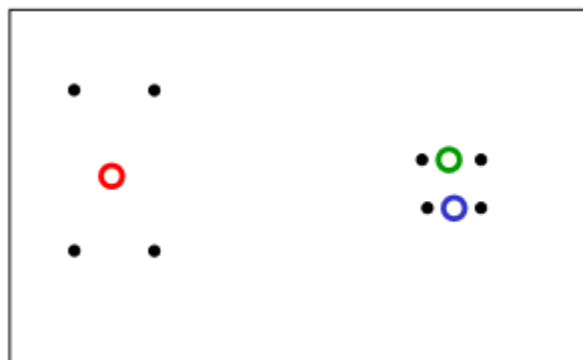
Fuzzy / soft k-means – вместо чёткой принадлежности – нечёткая **дальше**

k-medians (medoids) – другие усреднения (для борьбы с выбросами)

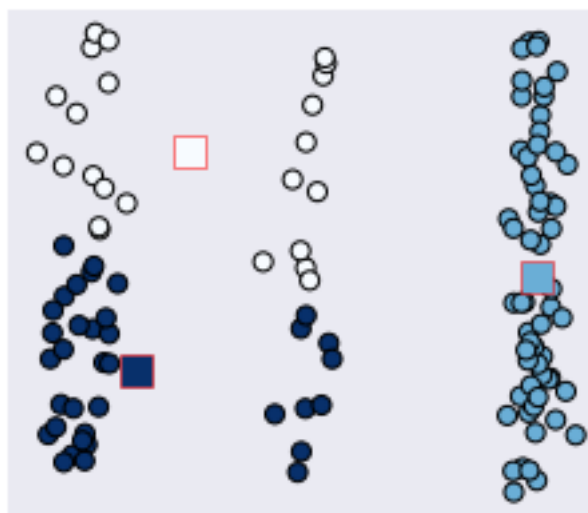
k-means + kernel tricks

k-means: ограничения

- хорошо работает для примерно равномогущих кластеров с «шаровой формой»
будут рисунки, потом поймём почему (GMM)
- оптимизируемый функционал не выпуклый

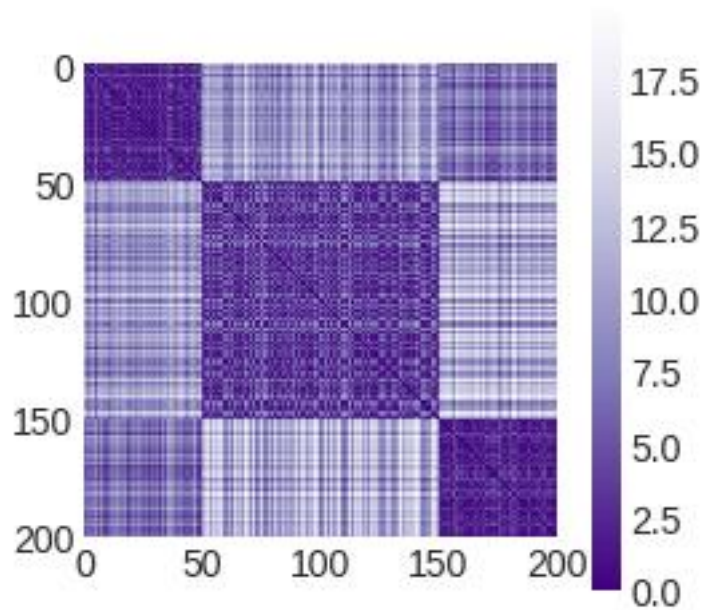
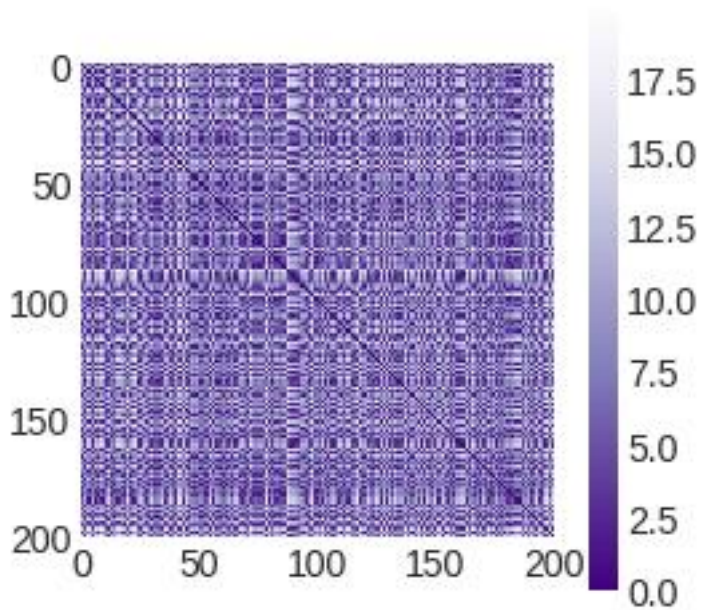


много локальных
минимумов

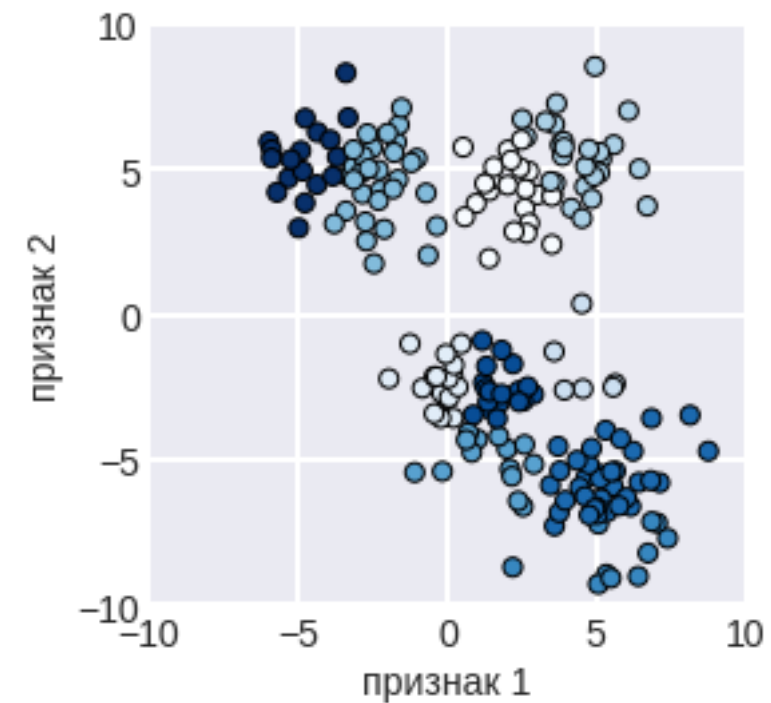
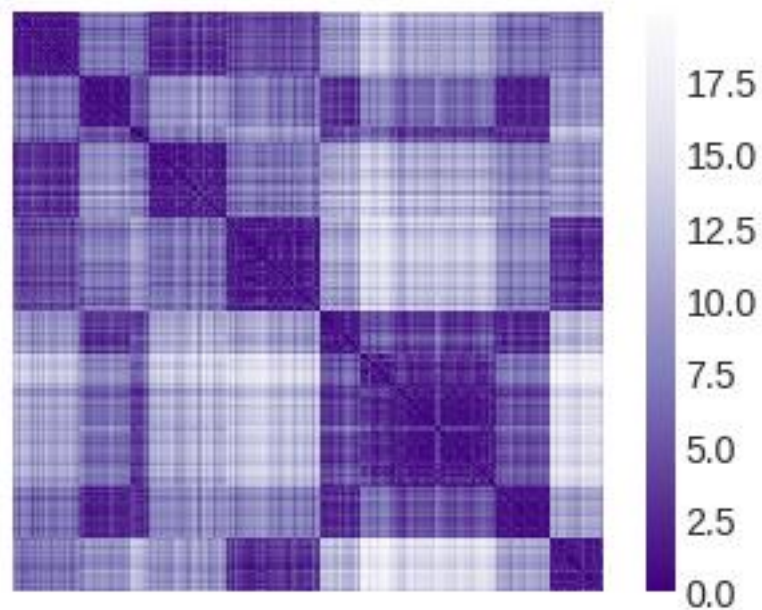


Иллюстрация

Матрица попарных расстояний до и после кластеризации
(объекты упорядочены по кластерам)

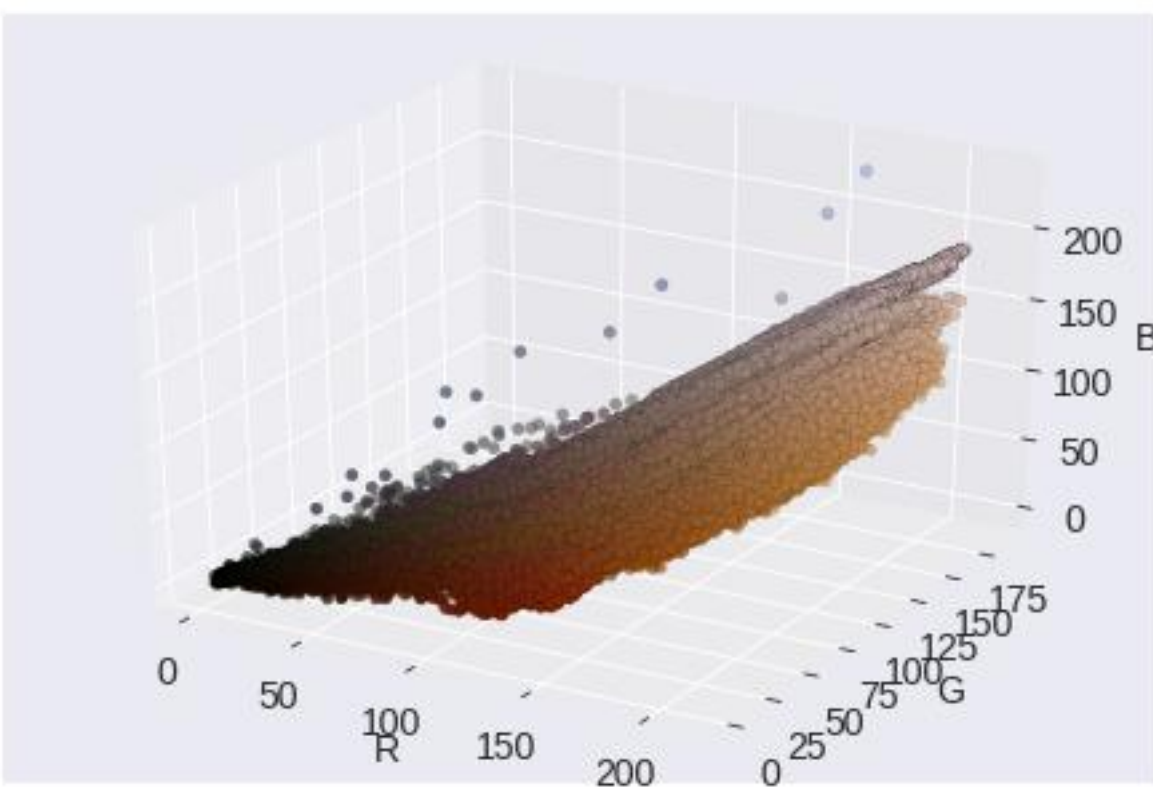
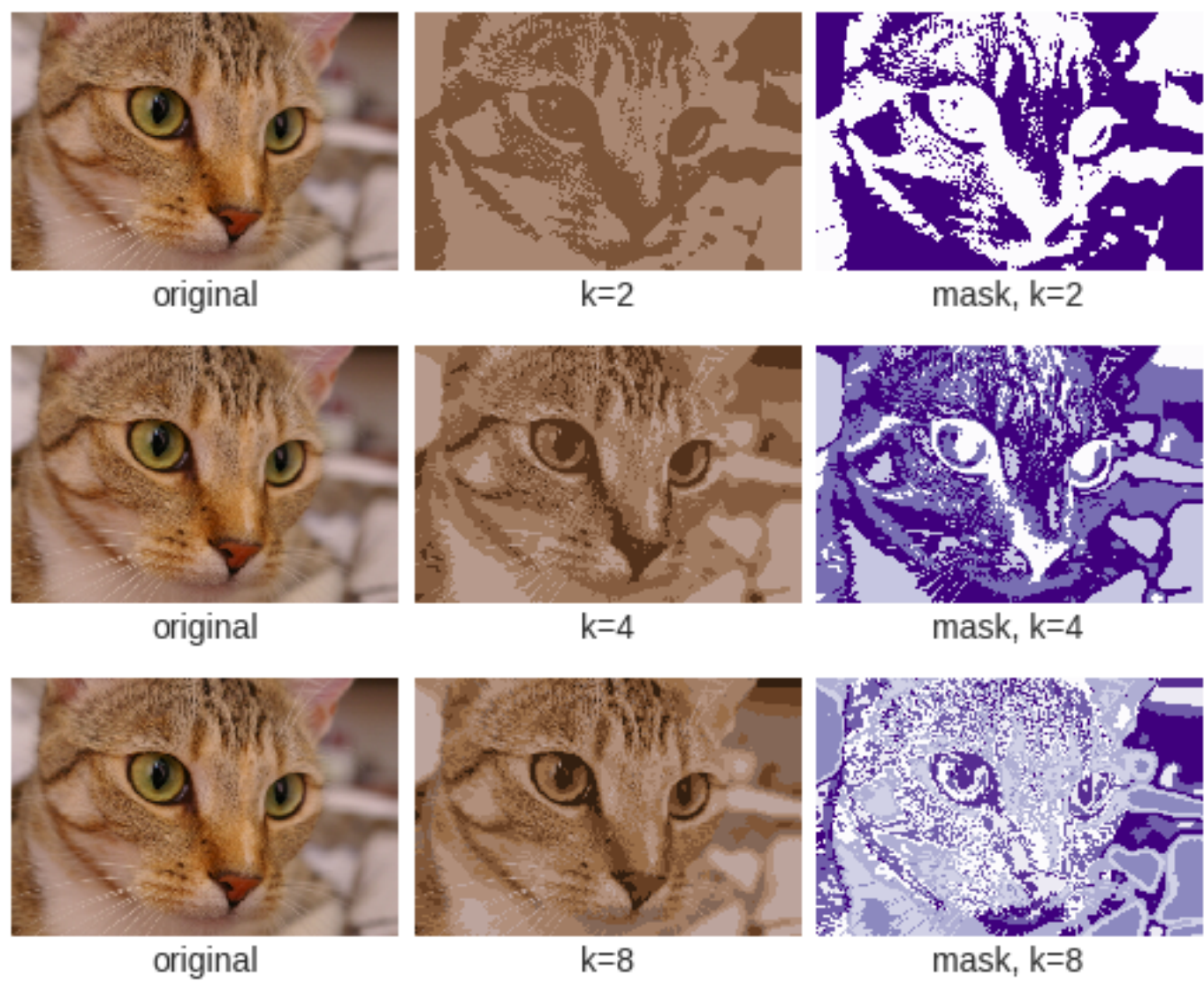


Иллюстрация



Д3 Как логично упорядочить кластеры?

Пример k-means: сегментации в изображениях



Не надо хранить все цвета, а только k – хорошее сжатие

Пример k-means: сегментации в изображениях

```
def simplify_image(image, k=10):  
    image2 = image.copy()  
    image2.resize([image.shape[0]*image.shape[1], 3])  
    model = KMeans(n_clusters=k, random_state=11)  
    model.fit(image2)  
    result = model.cluster_centers_.round().astype(int)[model.labels_, :]  
    result.resize([image.shape[0], image.shape[1], 3])  
    mask = model.labels_  
    mask.resize([image.shape[0], image.shape[1]])  
    return result, mask
```



original



k=2



k=4

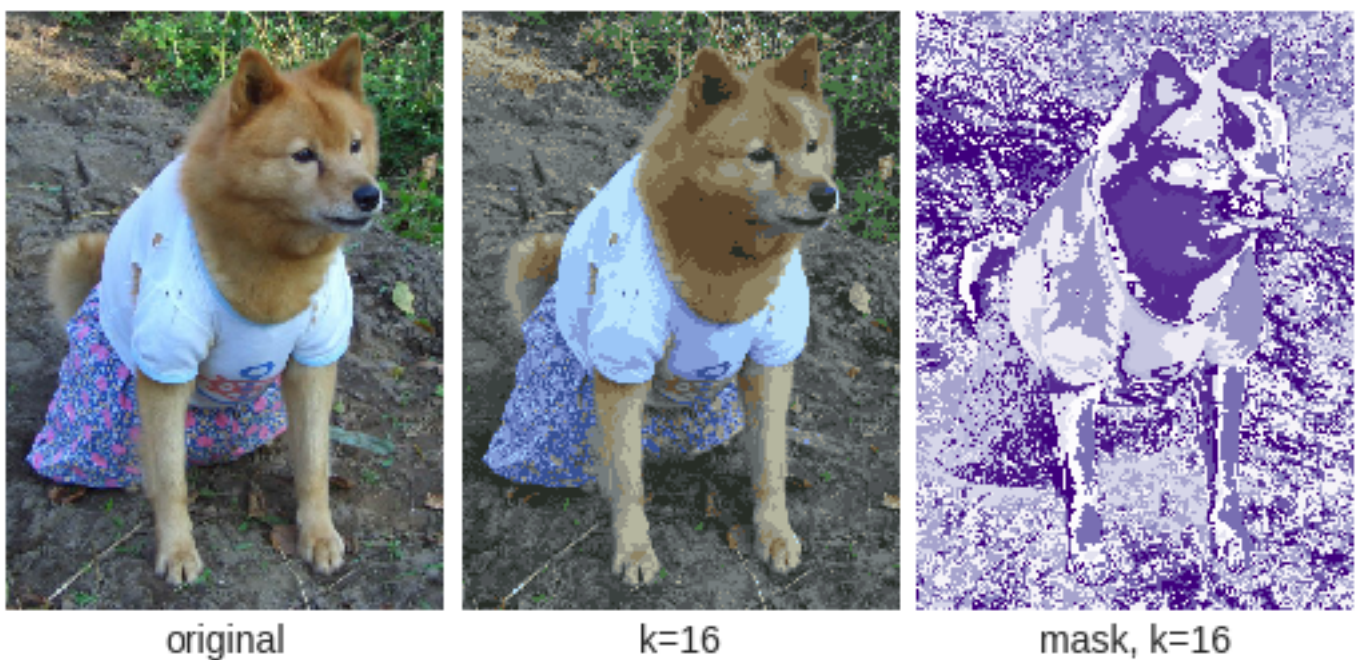


k=8



k=16

Пример k-means: сегментации в изображениях



k-Means – немного перепишем алгоритм

Вход:

$$\{x_1, \dots, x_m\} \subseteq \mathbf{R}^n$$

Изменения:

Инициализация:

k центров кластеров $\{\mu_1, \dots, \mu_k\} \subseteq \mathbf{R}^n$

Итерация:

1. Каждый объект приписать к тому кластеру, к центру которого он ближе:

$$C_t = \{i \mid \|x_i - \mu_t\| = \min_j \|x_i - \mu_j\|\}$$

2. Пересчитать центры кластеров:

$$\mu_t = \frac{1}{|C_t|} \sum_{i \in C_t} x_i$$

$$r_{it} = I[\|x_i - \mu_t\| = \min_s \|x_i - \mu_s\|]$$

считаем, что минимум единственный

$$\mu_t = \frac{1}{\sum_{i=1}^m r_{it}} \sum_{i=1}^m r_{it} x_i$$

soft k-Means**Вход:**

$$\{x_1, \dots, x_m\} \subseteq \mathbf{R}^n$$

Изменения:**Инициализация:**

k центров кластеров $\{\mu_1, \dots, \mu_k\} \subseteq \mathbf{R}^n$

Итерация:

1. Для каждого объекта – оценку принадлежности к каждому кластеру

$$r_{it} = \frac{\exp(-\beta \|x_i - \mu_t\|)}{\sum_j \exp(-\beta \|x_i - \mu_j\|)}$$

2. Пересчитать центры кластеров:

$$\mu_t = \frac{1}{\sum_{i=1}^m r_{it}} \sum_{i=1}^m r_{it} x_i$$

soft k-Means

+ мягкая кластеризация

+ иногда работает лучше

– выбор параметра «бета»

– остались проблемы

**«продолговатые кластеры»
кластеры разной мощности**

<http://qaru.site/questions/611120/minibatchkmeans-parameters>

Распространение близости (Affinity propagation): кластеризация сообщениями между точками

пусть близость (similarity) между точками x_i и x_j – $s(i, j)$

формулы пересчёта изначально нулевых:

ответственности (responsibility) – $r(i, k) \leftarrow s(i, k) - \max\{a(i, k') + s(i, k') \mid k' \neq k\}$

(k может быть лидером для i – лидер должен быть похожим)

доступности (availability) – $a(i, k) \leftarrow \min(0, r(k, k) + \sum_{i' \notin \{i, k\}} r(i', k))$

(i может подчиняться k – лидер должен быть лидером для многих)

тут есть варианты...

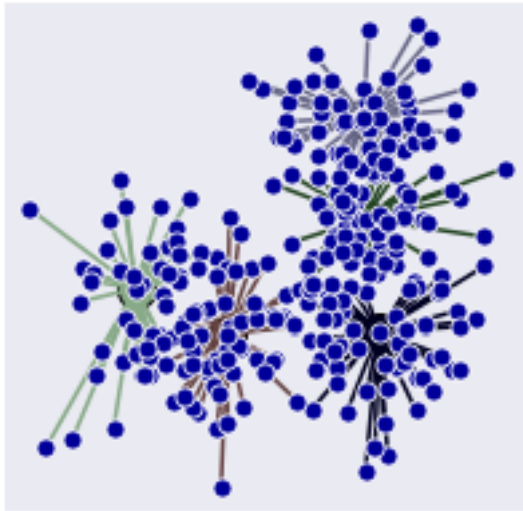
Обычно пересчитывают «с инерцией»:

$$r^{(t+1)}(i, k) \leftarrow \lambda r^{(t)}(i, k) + (1 - \lambda) r^{\text{new}}(i, k)$$

$$C_t = \{x_i \mid t = \arg \max(a(i, \bullet) + r(i, \bullet))\}$$

т.е. потенциально меток столько, сколько точек

Распространение близости (Affinity propagation)



– высокая сложность $O(m^2 n_{\text{iter}})$

– требования к памяти $O(m^2)$

+ не нужно задавать число кластеров

Brendan J. Frey, Delbert Dueck «Clustering by Passing Messages Between Data Points»
Science Feb. 2007 <https://www.icmla-conference.org/icmla07/FreyDueckScience07.pdf>

Есть хорошее объяснение <https://habr.com/ru/post/321216/>

Affinity propagation – реализация

`sklearn.cluster.AffinityPropagation`

`damping` – **фактор забывания значений параметров**

`max_iter` – **максимальное число итераций**

`convergence_iter` – **число итераций без изменений, после которого останов**
`copy`

`preference` – **вектор вероятностей для каждой точки стать экземпляром (т.е. породить кластер)**

`affinity` – **п.в. эвклидово расстояние**

`verbose`

`random_state`

Сдвиг среднего (Mean Shift): обнаружение мод плотности

Идея: рассмотрим точку x и её окрестность $N(x)$

оценим «центр масс» в этой окрестности

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x - x_i) x_i}{\sum_{x_i \in N(x)} K(x - x_i)}$$

сдвинем точку в центр: $x \leftarrow x + \eta(m(x) - x)$

хотим быть в центре «сгустка»

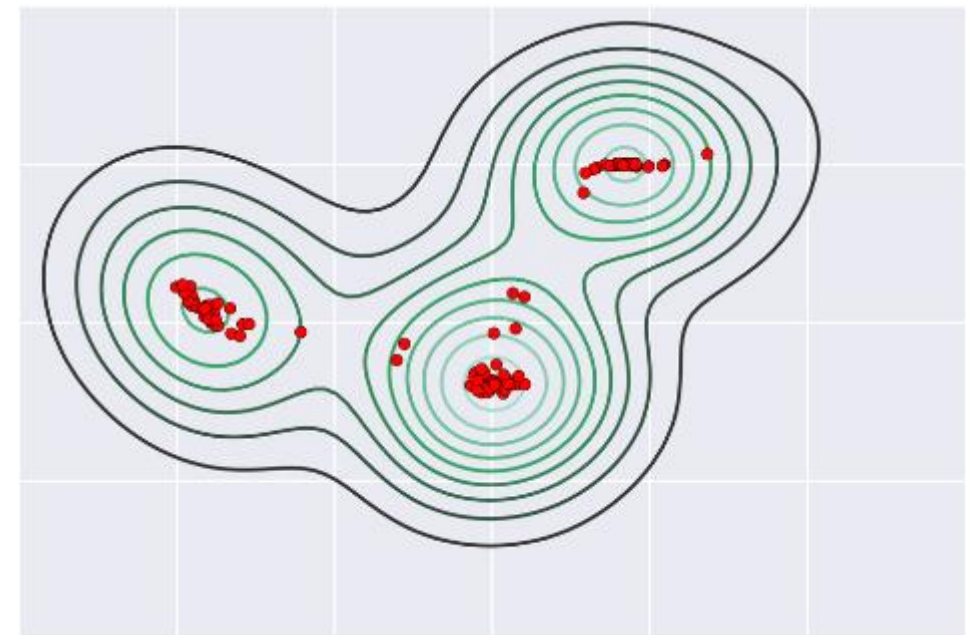
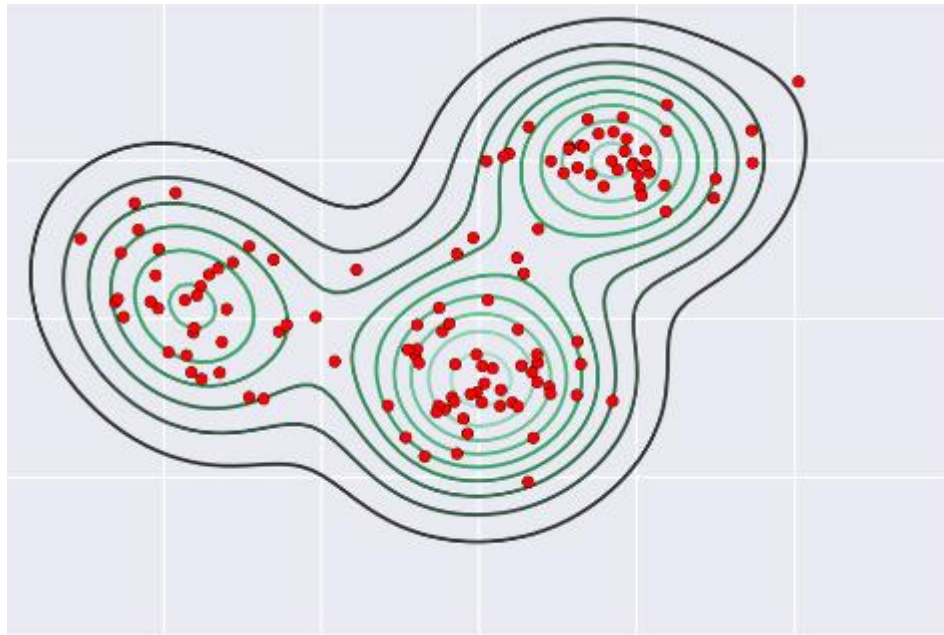
**Можно итерационно повторять, сдвигая все точки выборки,
пока они не притянутся (с ε -точностью) к общим центрам...**

каждый полученный центр – свой кластер

$$K(z) = \exp(-\|z\|/h)$$
$$N(x) = \{x_i \mid \rho(x, x_i) \leq d\}$$

Потом фильтрация для устранения почти-дубликатов (отличаются на ε).

Сдвиг среднего (Mean Shift): обнаружение мод плотности



все точки, которые «стекли» в одну – один кластер

ϵ , d и h влияют на число кластеров, которые получатся

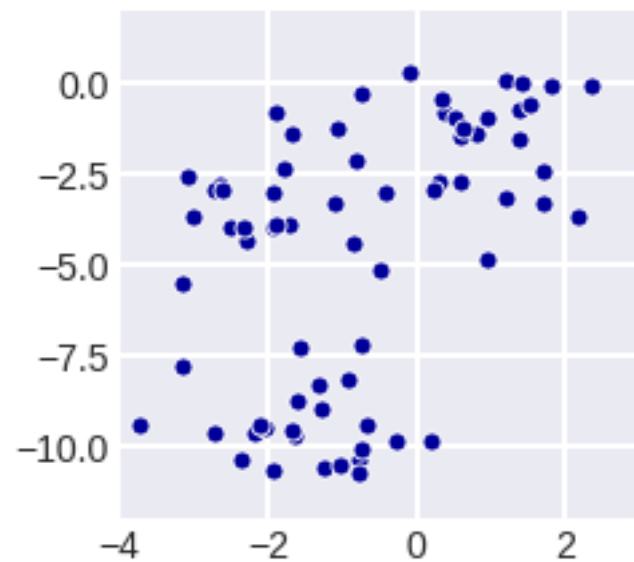
<https://spin.atomicobject.com/2015/05/26/mean-shift-clustering/>

В общем случае нет доказательства сходимости:

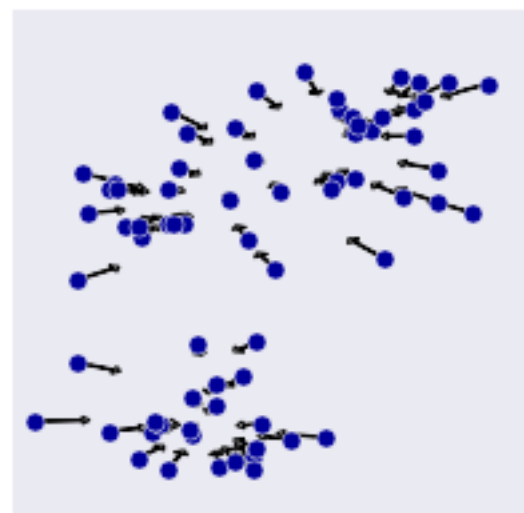
https://en.wikipedia.org/wiki/Mean_shift

Сдвиг среднего (Mean Shift): обнаружение мод плотности

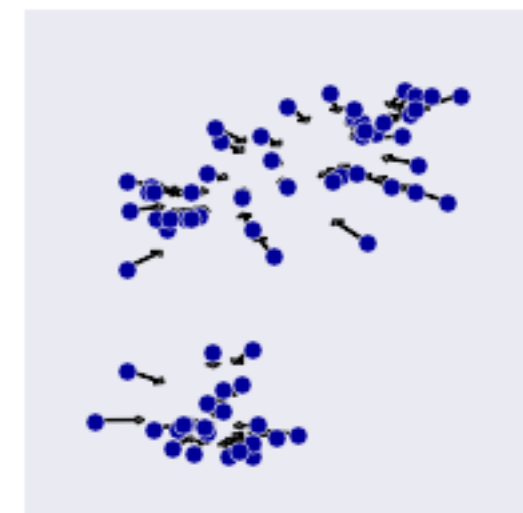
Выборка



Шаг 1

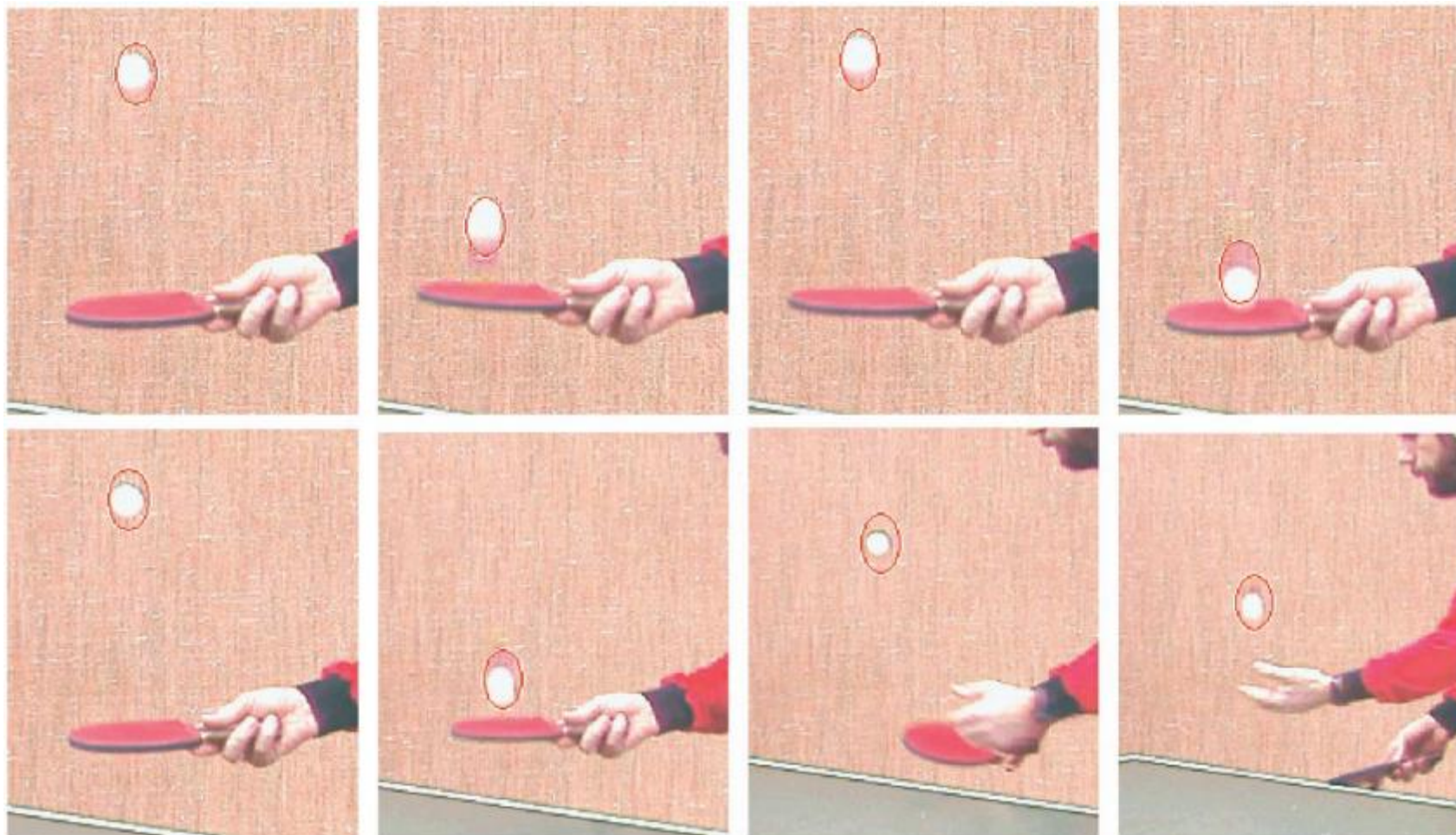


Шаг 2



Сдвиг среднего (Mean Shift): обнаружение мод плотности

Кроме кластеризации подходит для задачи «отслеживания»



D. Comaniciu, V. Ramesh, P. Meer «Kernel-based Object Tracking» // IEEE Transactions on Pattern Analysis and Machine Intelligence. — IEEE, 2003.
D. Comaniciu, P. Meer, «Mean Shift: A robust approach toward feature space analysis» IEEE Transactions on Pattern Analysis and Machine Intelligence. 2002. pp. 603-619.

Сдвиг среднего (Mean Shift): аналоги

Можно выписать плотность

$$\rho(x) \propto \sum_{i=1}^m K\left(\frac{|x - x_i|}{h}\right)$$

и градиентный метод

$$x \leftarrow x + \eta \nabla \rho(x)$$

для всех точек выборки

(можно не задаваясь окрестностью)

MeanShift – реализация

`sklearn.cluster.MeanShift`

`bandwidth` – параметр RBF-ядра

`seeds` – для инициализации ядер

`bin_seeding` – если включить опцию, изначально алгоритм работает в огрублённом пространстве (дискретизация исходных данных)

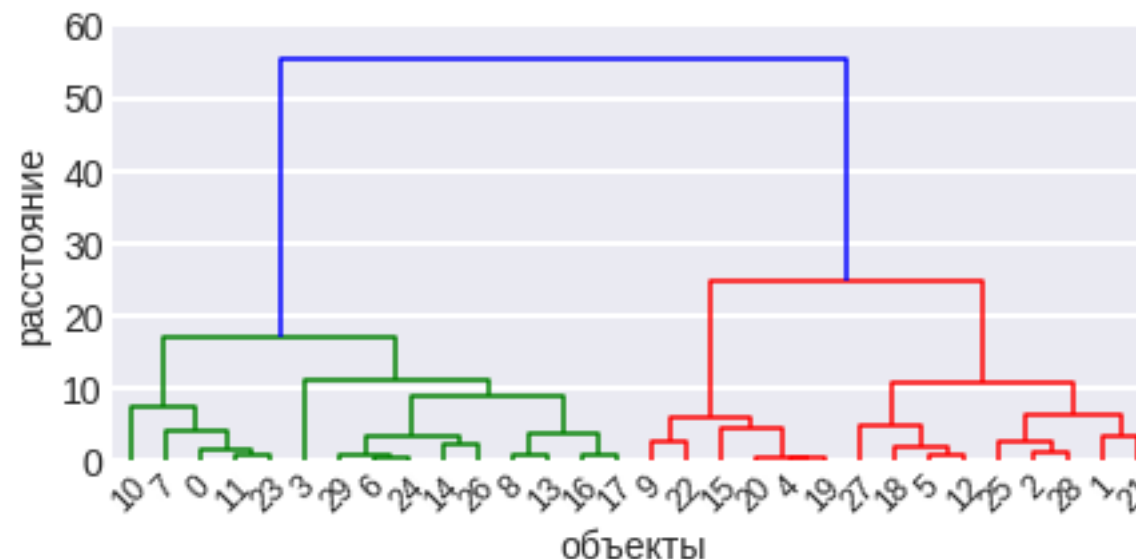
`min_bin_freq` – для увеличения скорости принимать бины только с таким количеством точек

`cluster_all` – кластеризовать ли все точки. Есть возможность оставлять «сирот» (`orphans`) без кластеров (в окрестности которых нет других точек)

`n_jobs`

Иерархическая кластеризация (Hierarchical clustering)

ИК в отличие от плоской (Flat Clustering) получает серию вложенных друг в друга кластеризаций (\Rightarrow не требует заранее заданного числа кластеров)



ИК может быть долгой...

https://en.wikipedia.org/wiki/Hierarchical_clustering

Два подхода к иерархической кластеризации

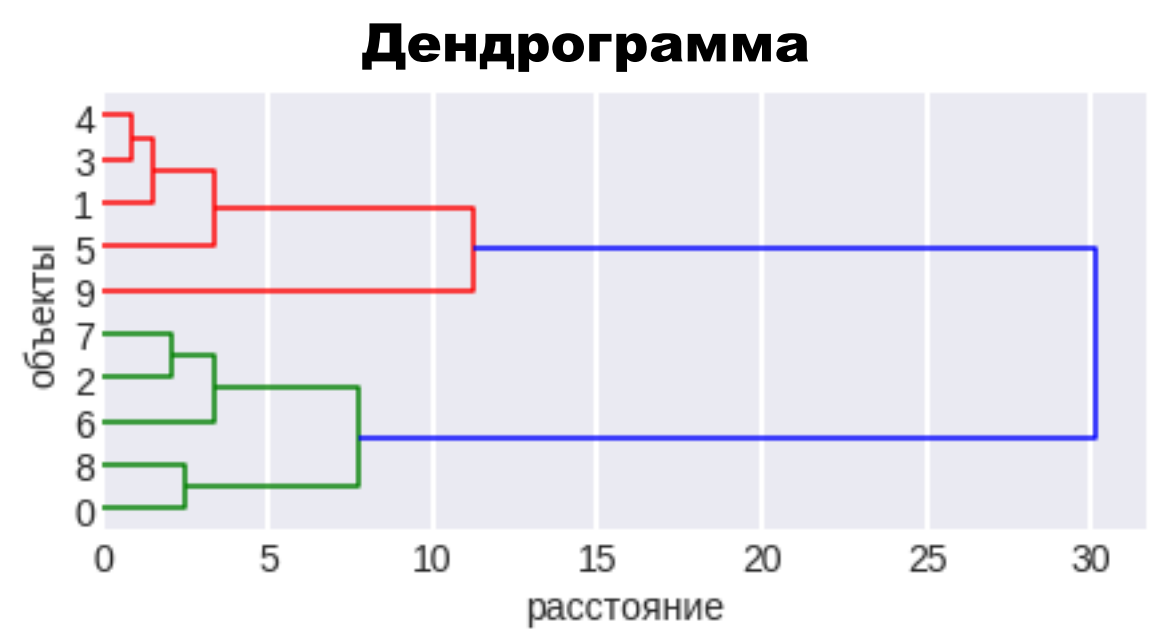
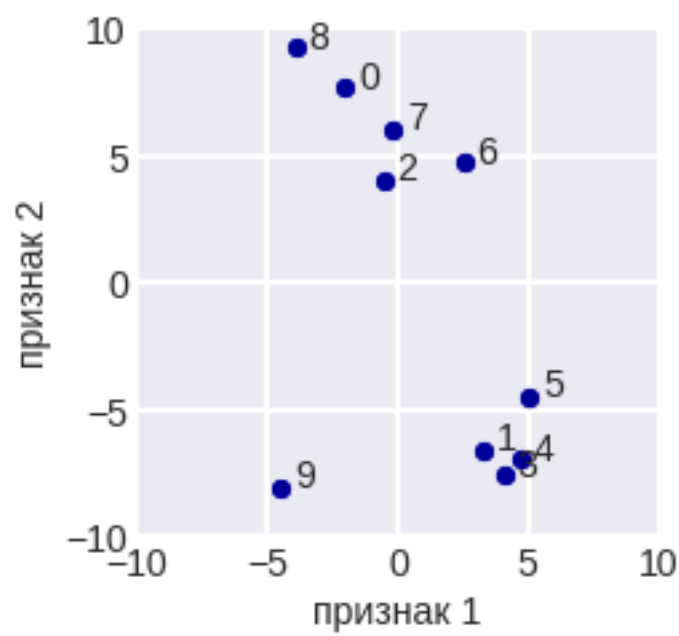
Агломеративный (Agglomerative / bottom-up)

- изначально число кластеров совпадает с числом объектов, каждый кластер содержит один объект обучения
- на каждом шаге объединяем два наиболее похожих кластера
- останавливаемся, если остаётся один кластер
- чаще используют (проще реализовать)

Дивизионный (Divisive / top-down)

- изначально есть один кластер, содержащий все объекты обучения
- расщепить кластер **с максимальными внутрикластерными расстояниями** на два
- останавливаемся, когда число кластеров совпадает с числом объектов

Визуализация иерархической кластеризации



Корень дерева – кластер, который содержит все объекты

Визуализация иерархической кластеризации

```
from scipy.cluster.hierarchy import dendrogram
from sklearn.cluster import AgglomerativeClustering

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)

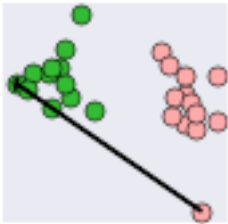
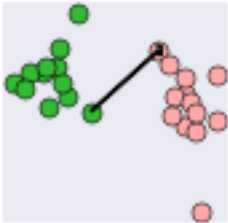

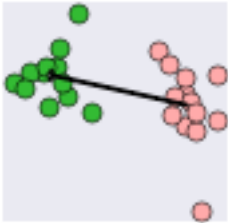
    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs, orientation = 'right')
    return linkage_matrix
```

```
model = AgglomerativeClustering(
    distance_threshold=0,
    n_clusters=None)

model = model.fit(X)

plot_dendrogram(model,
    truncate_mode='level',
    p=5,
    leaf_font_size=14)
```

Как измерить расстояния между кластерами – типы Linkage

Linkage		Описание
Complete		Maximal inter-cluster dissimilarity $\max_{x \in A, x' \in B} \text{dis}(x, x')$
Single		Minimal inter-cluster dissimilarity $\min_{x \in A, x' \in B} \text{dis}(x, x')$
Average		Mean inter-cluster dissimilarity $\frac{1}{ A \cdot B } \sum_{x \in A, x' \in B} \text{dis}(x, x')$
Centroid		Dissimilarity between the centroid for cluster A and B $\text{dis}(\bar{x}_A, \bar{x}_B)$

Как измерить расстояния между кластерами – типы Linkage

Расстояние Варда (Ward's measure)

изменение суммы квадратов

$$\lambda(X) = \sum_{x \in X} \|x - \bar{X}\|^2$$

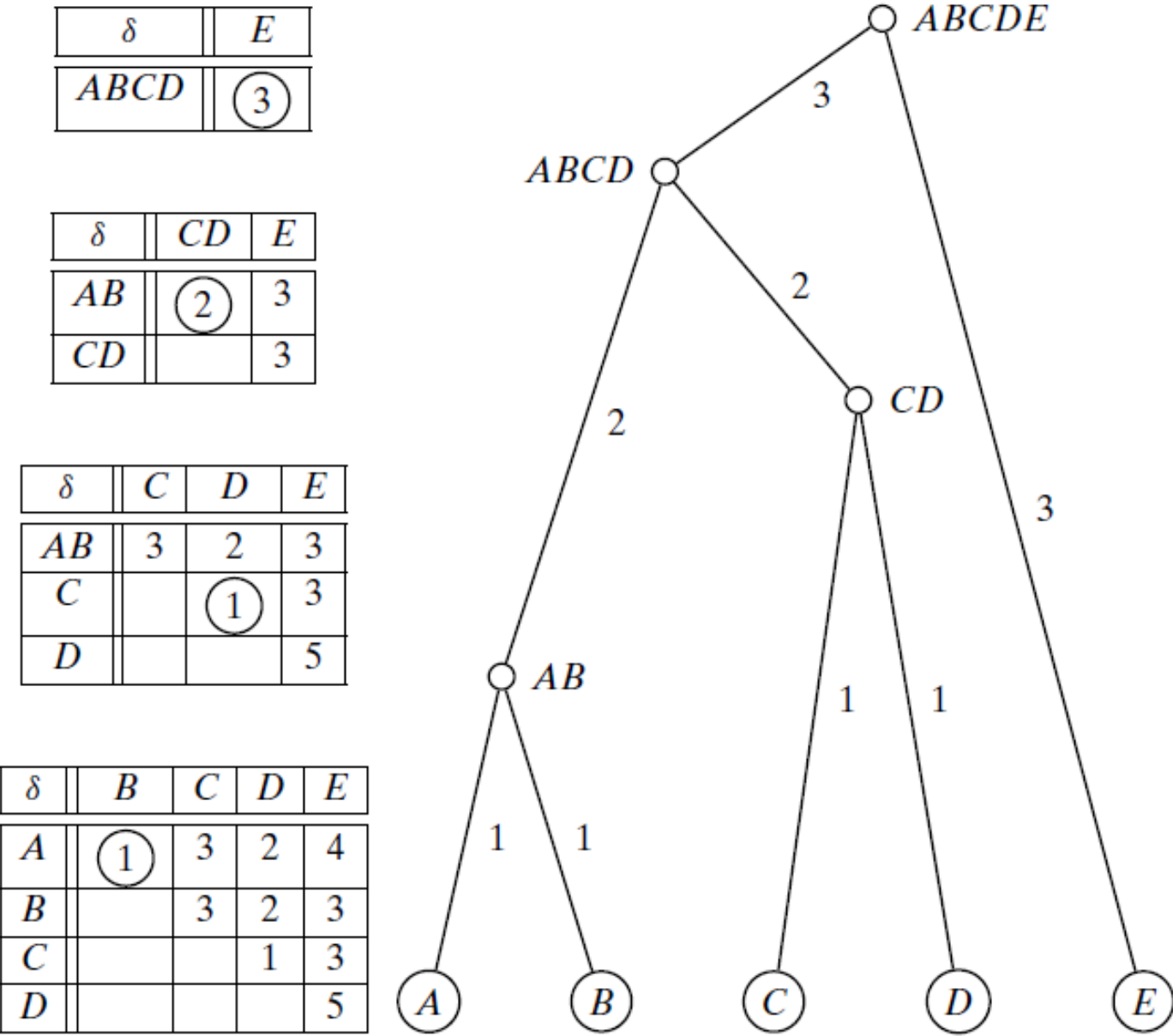
после объединения кластеров:

$$\lambda(A \cup B) - \lambda(A) - \lambda(B) = \frac{1}{\frac{1}{|A|} + \frac{1}{|B|}} \|\bar{A} - \bar{B}\|^2$$

ДЗ можно доказать...

https://en.wikipedia.org/wiki/Hierarchical_clustering

Single Link



Lance–Williams formula

**«Lance–Williams formula» при объединении кластеров
не будем рассматривать**

$$\delta(C_{ij}, C_r) = \alpha_i \cdot \delta(C_i, C_r) + \alpha_j \cdot \delta(C_j, C_r) + \\ \beta \cdot \delta(C_i, C_j) + \gamma \cdot |\delta(C_i, C_r) - \delta(C_j, C_r)|$$

Measure	α_i	α_j	β	γ
Single link	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$
Complete link	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
Group average	$\frac{n_i}{n_i + n_j}$	$\frac{n_j}{n_i + n_j}$	0	0
Mean distance	$\frac{n_i}{n_i + n_j}$	$\frac{n_j}{n_i + n_j}$	$\frac{-n_i \cdot n_j}{(n_i + n_j)^2}$	0
Ward's measure	$\frac{n_i + n_r}{n_i + n_j + n_r}$	$\frac{n_j + n_r}{n_i + n_j + n_r}$	$\frac{-n_r}{n_i + n_j + n_r}$	0

https://en.wikipedia.org/wiki/Ward's_method

Агломеративная кластеризация

`sklearn.cluster.AgglomerativeClustering`

`n_clusters` – число кластеров

`affinity` – «euclidean», «l1», «l2», «manhattan», «cosine», «precomputed», функция

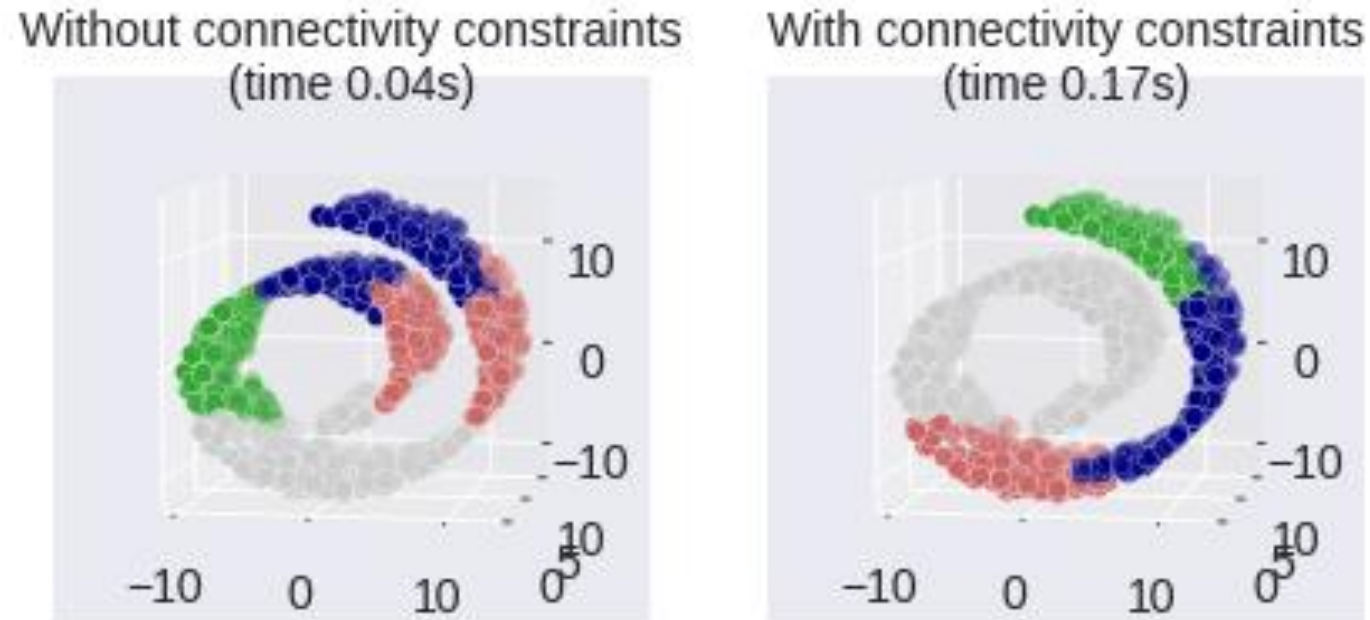
`memory` – параметр связан с кэшированием дерева

`connectivity` – матрица связности (матрица или функция), чтобы только соседние кластеры объединялись, по умолчанию нет

`compute_full_tree` – строить ли всё дерево (обычно до `n_clusters`)

`linkage` – критерий слияния: «ward», «complete», «average», «single»,

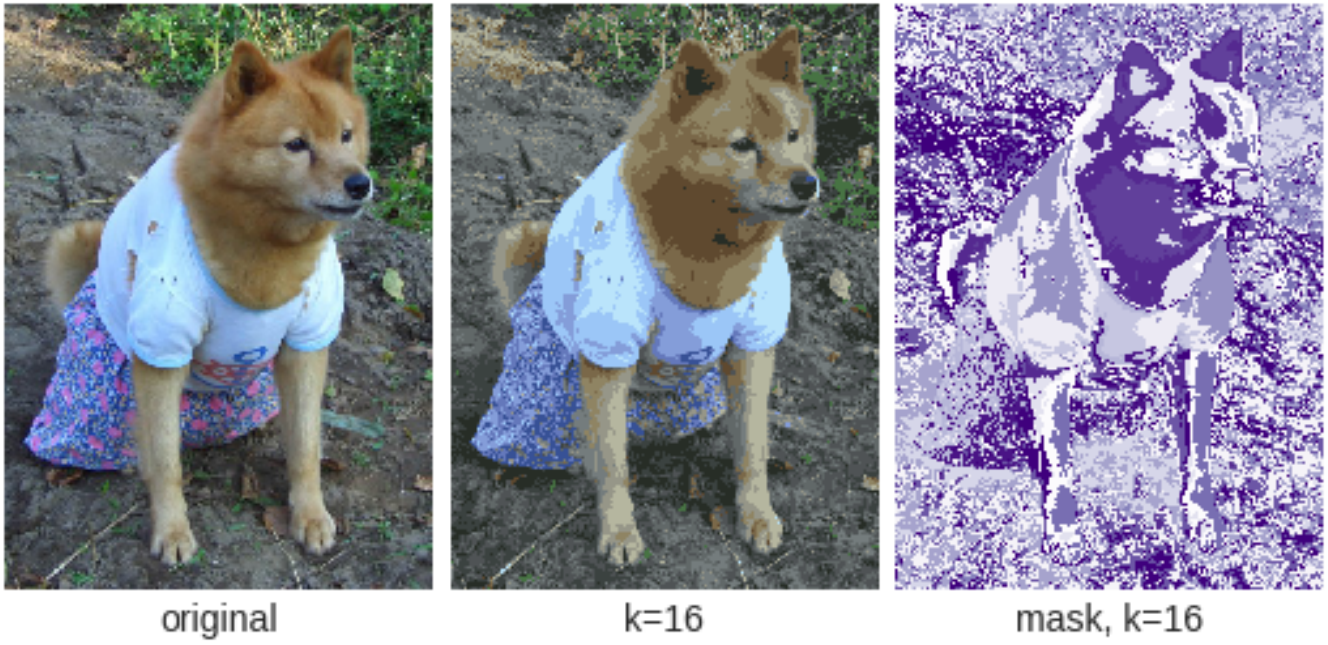
Использование матрицы связности



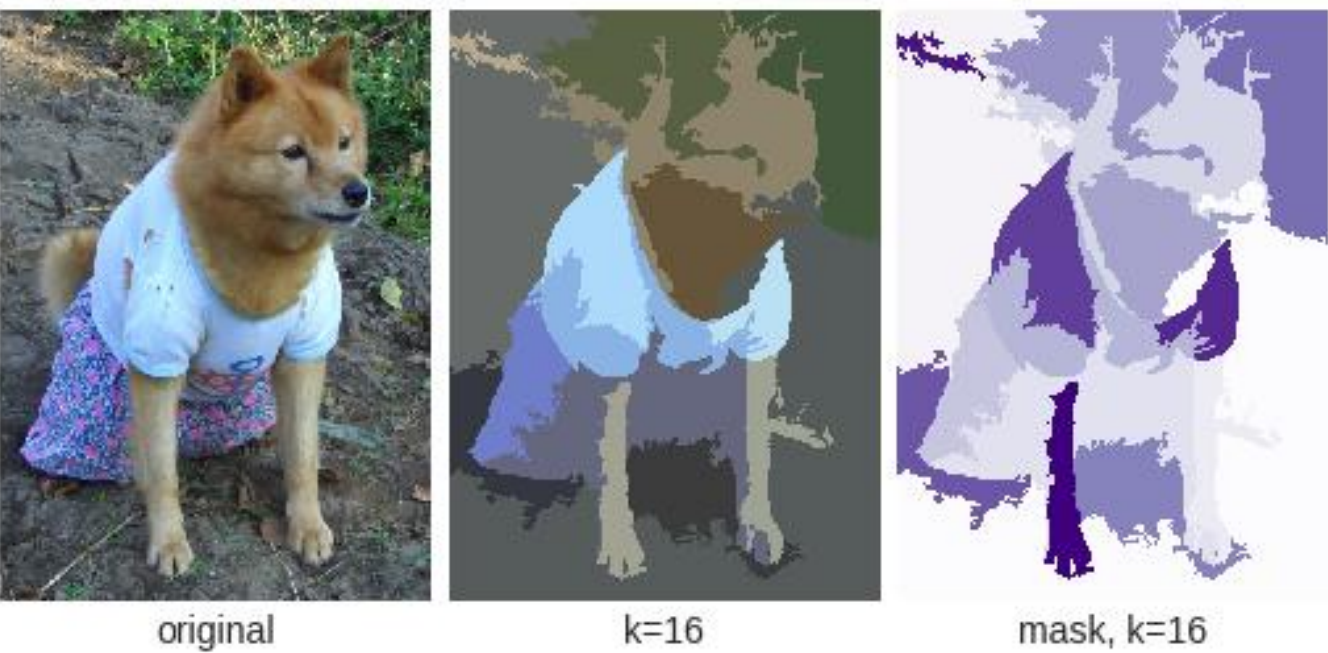
матрица k-соседства

Агломеративная кластеризация для сокращения признакового пространства

```
from sklearn import cluster
agglo = cluster.FeatureAgglomeration(n_clusters=32)
agglo.fit(X)
X_reduced = agglo.transform(X)
X.shape, X_reduced.shape
(1797, 64), (1797, 32)
```

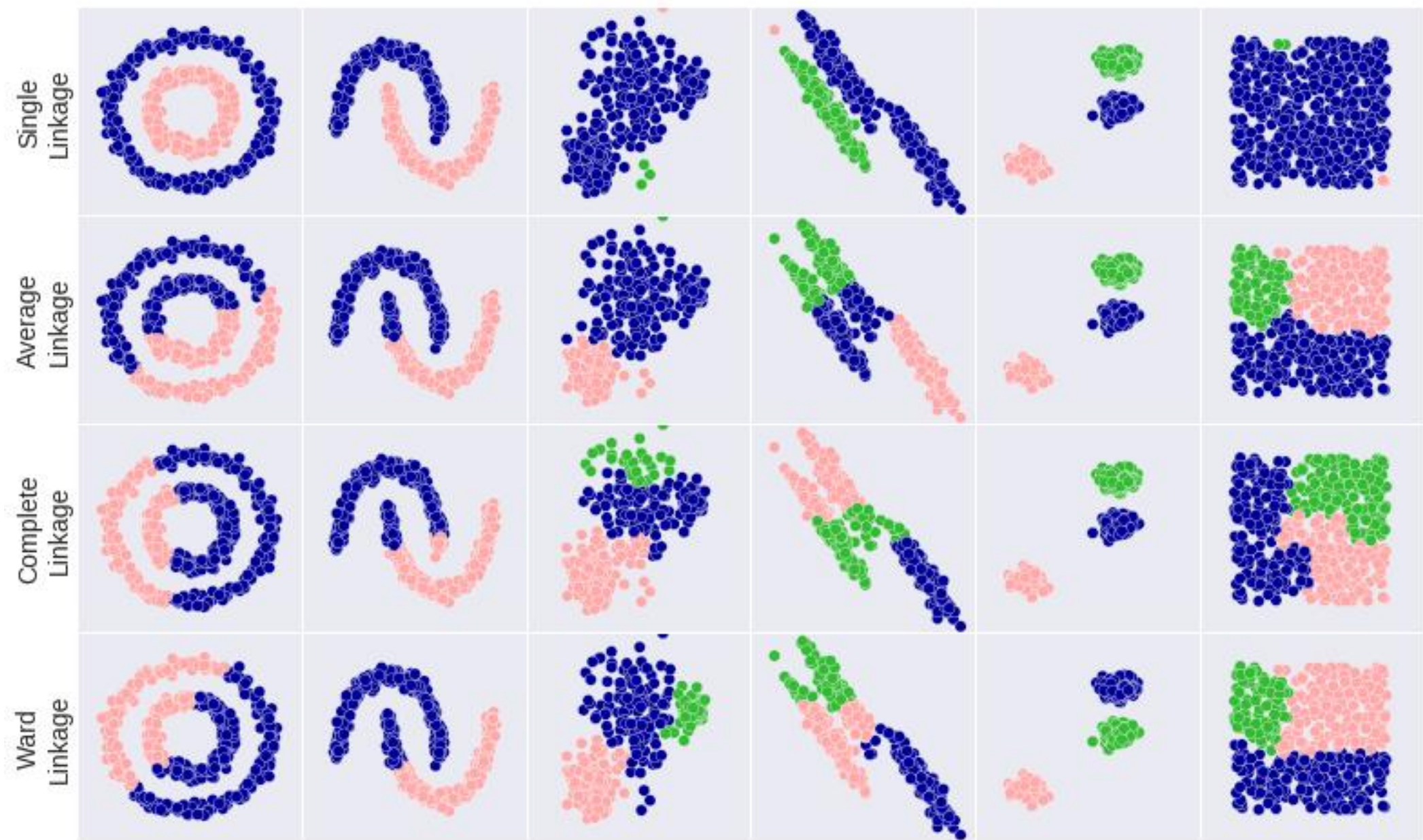


Кластеризация пикселей по признакам (R, G, B)



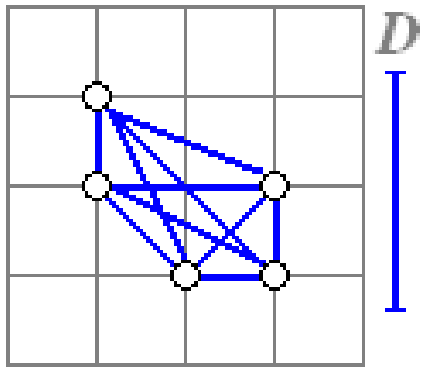
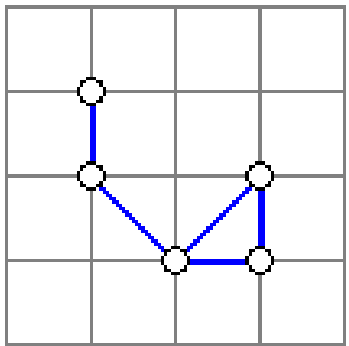
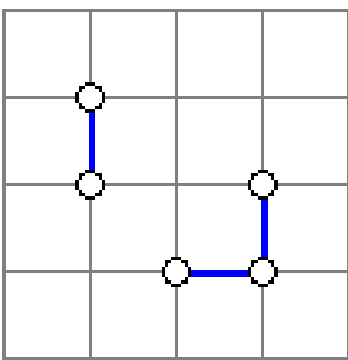
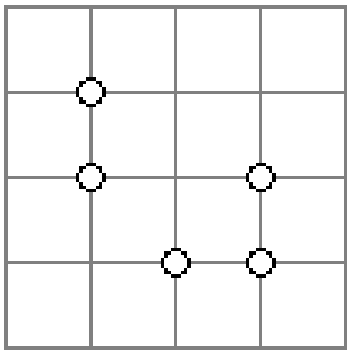
Сегментации в изображениях по признакам (R, G, B) по графу 4-соседства

Разные виды связности



Графовые методы

Эволюция порогового графа



single link
complete link

$\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$
 $\{\{1\},\{2\},\{3\},\{4\},\{5\}\}$

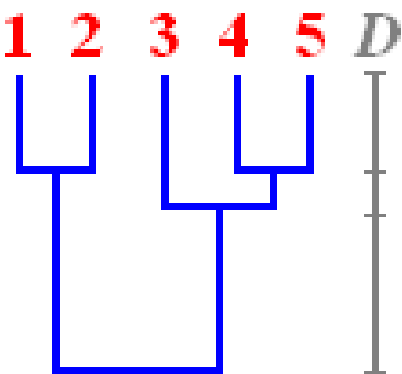
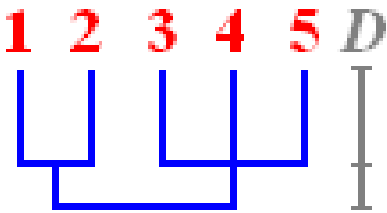
$\{\{1,2\},\{3,4,5\}\}$
 $\{\{1,2\},\{3\},\{4,5\}\}$

$\{\{1,2,3,4,5\}\}$
 $\{\{1,2\},\{3,4,5\}\}$

$\{\{1,2,3,4,5\}\}$

CL – неоднозначные разбиения (возможны ничьи)

Дендрограммы SL / CL

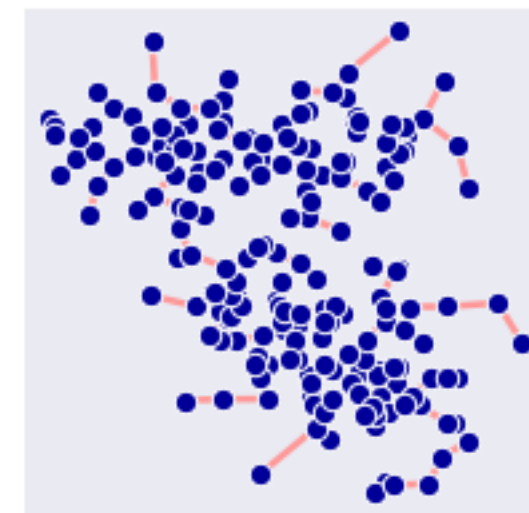
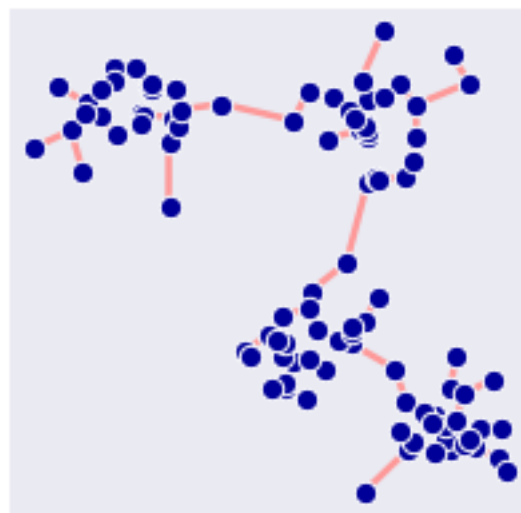
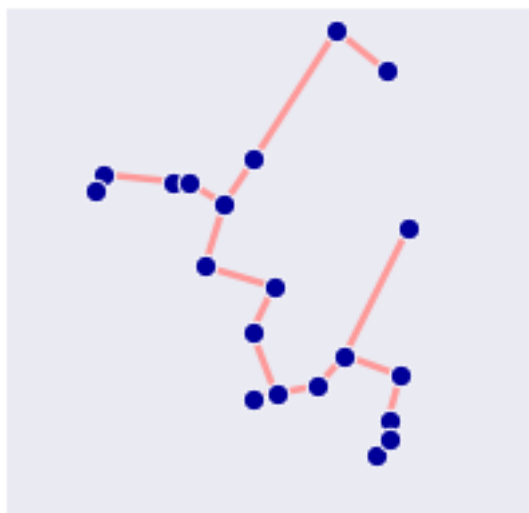


Minimum Spanning Tree (MST) – на основе минимального остовного дерева

1. Построить минимально основное дерево
жадным методом: добавлять в дерево ребро наименьшего веса
(так, чтобы оно оставалось деревом)

2. Удалить из дерева (k-1) ребро наибольшего веса

3. Компоненты связности полученного графа – кластеры



Эквивалентны ли MST и SL?

Spectral Clustering – Спектральная кластеризация

**В случае сложных кластеров типа «вложенные окружности»
С помощью нормированного Лапласиана
пришла из теории графов**

**Ulrike von Luxburg «A Tutorial on Spectral Clustering», 2007
<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.165.9323>**

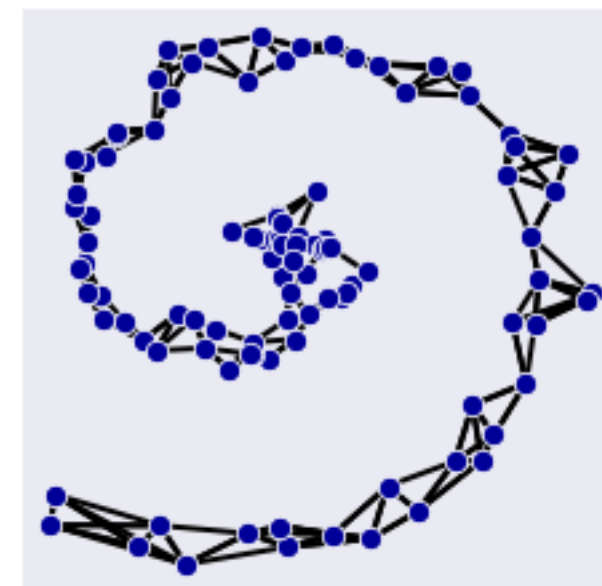
Spectral Clustering – Спектральная кластеризация

Построить по данным $\{x_1, \dots, x_m\}$ граф с весами

Множество вершин: $\{1, \dots, m\}$

Вершина i соединена с k вершинами,
которые являются номерами ближайших k объектов к x_i

Ребро (i, j) если существует, имеет вес $\exp(-\|x_i - x_j\|^2 / \sigma)$



Spectral Clustering – Спектральная кластеризация

Дальше используется спектральная кластеризация графа:

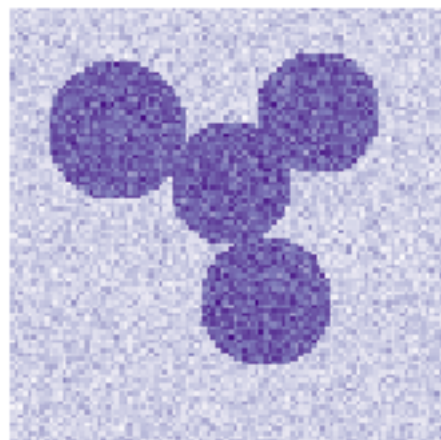
1. Строим матрицу Лапласа $L = D - W$
(или $L' = I - D^{-1}W$)

2. Находим k с.в., соответствующих минимальным с.з.: $v_1 = \tilde{1}, v_2, \dots, v_k$

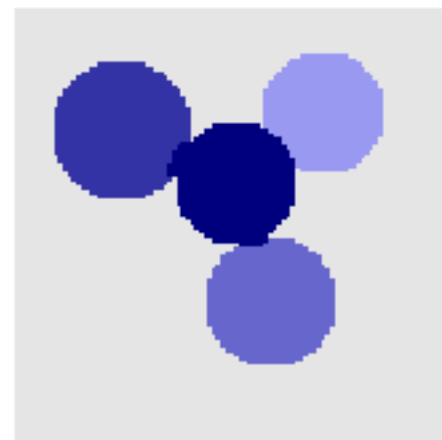
3. Конструируем признаковую матрицу $V = [v_2, \dots, v_k]$

4. Применяем к ней классический алгоритм кластеризации (например, k-средних)

Spectral Clustering – Спектральная кластеризация



изображение



спектральная кластеризация

**Рекомендуется, когда матрица сходства разрежена
надо выбрать `eigen_solver="amg"`
предварительно установив `ruamg`
(library of Algebraic Multigrid)**

```
from sklearn.feature_extraction import image
from sklearn.cluster import spectral_clustering

graph = image.img_to_graph(img, mask=mask)
graph.data = np.exp(-graph.data / graph.data.std())
labels = spectral_clustering(graph, n_clusters=4, eigen_solver='arpack')
```

Spectral Clustering – реализация

`sklearn.cluster.SpectralClustering`

`n_clusters` – **число кластеров**

`eigen_solver` – **метод вычисления собственных векторов: `None`, «arpack», «lobpcg», «amg»**

`n_components` – **размерность пространства с.в., в котором кластеризуем**
`random_state`

`n_init` – **число запусков k-means**

`gamma` – **коэффициент для ядер: `rbf`, `poly`, `sigmoid`, `laplacian`, `chi2`**

`affinity` – **(сходство) «nearest_neighbors» / «precomputed» / «rbf» или любое из `sklearn.metrics.pairwise_kernels`**

`n_neighbors` – **число соседей, нужно для 1-nn, если он используется**

`eigen_tol` – **порог сходимости при декомпозиции матрицы Лапласа**

`assign_labels` – **стратегия для простановки меток после лапласового вложения: «kmeans» / «discretize»**

`degree` – **степень полинома**

`coef0` – **коэффициент полиномиального или сигмоидального ядра**

`kernel_params` – **параметры для самописных ядер**

DBSCAN = Density-Based Spatial Clustering of Applications with Noise

Алгоритм с маркировкой точек как шум и т.п.

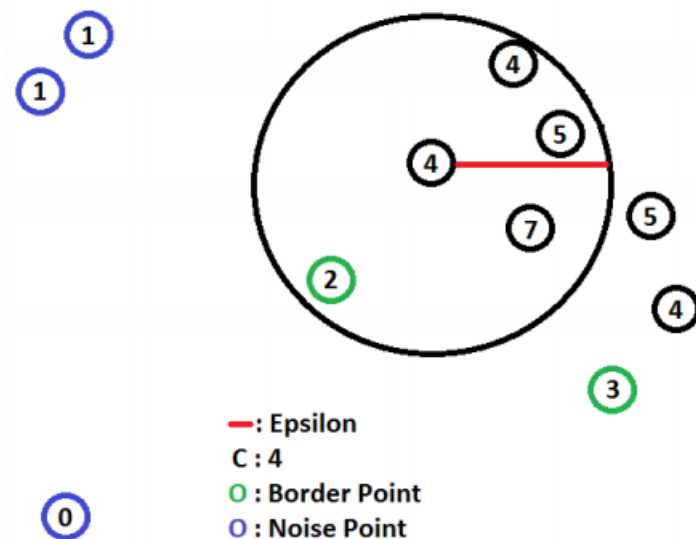
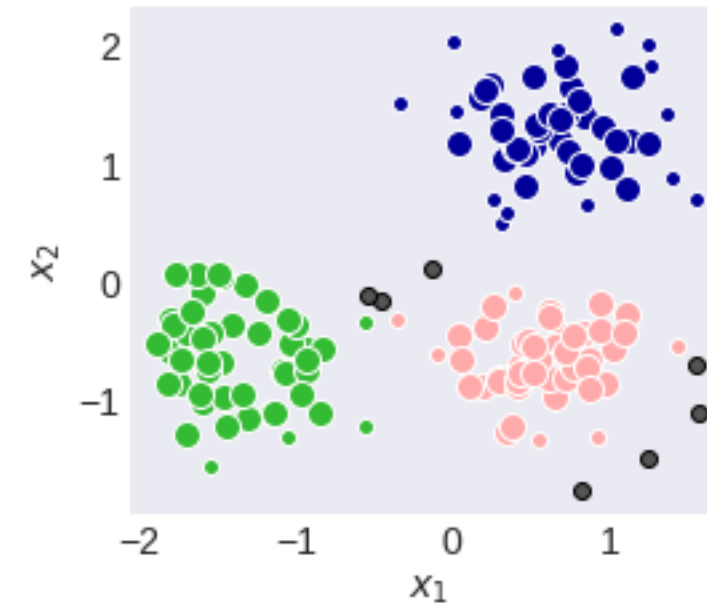
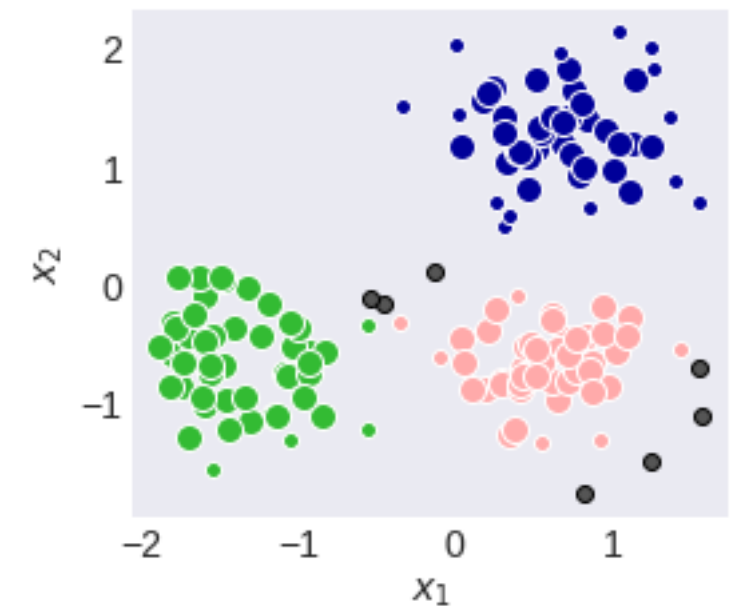
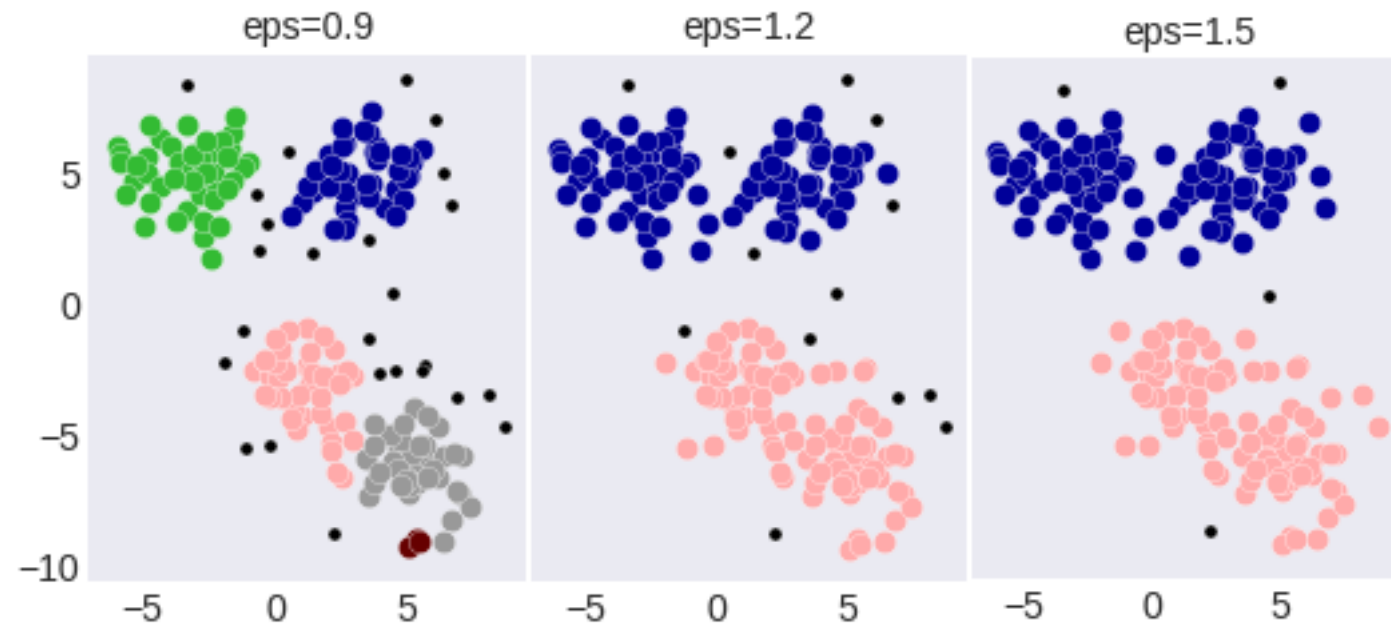


Figure 2: Noise Points in DBSCAN



- для каждой точки строим круг радиуса ε
- если в круге есть как минимум k точек – она плотная / основная (core point)
- если нет, но есть плотные точки – она граничная / достижимая (border point)
иногда – если есть путь точек на расстоянии $< \varepsilon$ до основной точки
- иначе – шум (noise point)
- множество основных точек – кластеризовать SingleLink
с пороговым расстоянием $\geq \varepsilon$
- множество граничных точек – к соответствующим кластерам

DBSCAN = Density-Based Spatial Clustering of Applications with Noise



```
from sklearn.cluster import DBSCAN
clustering = DBSCAN(eps=0.9, min_samples=3).fit(X)
labels = clustering.labels_
```

DBSCAN = Density-Based Spatial Clustering of Applications with Noise

- + кластеры произвольной формы**
 - + можно работать с большими данными**
 - + детерминированный (нет случайности, ответ однозначный)**
 - но может зависеть от порядка данных**
 - (как приписываемые метки, так и приписывание неосновные точек)**
 - неудобные гиперпараметры**
 - когда плотности сильно неоднородны**
 - не применяют для высокоразмерных данных**
- у scikit-learn-реализации могут быть проблемы с памятью**

Ester, M., H. P. Kriegel, J. Sander, X. Xu «A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise» // In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, Portland, OR, AAAI Press, pp. 226-231. 1996

DBSCAN = Density-Based Spatial Clustering of Applications with Noise

`sklearn.cluster.DBSCAN`

eps – порог близости

min_samples – число объектов в окрестности, чтобы считать точку основной

metric – пользовательская или ...

metric_params – параметры метрики

algorithm – алгоритм для нахождения ближайших соседей: «auto», «ball_tree», «kd_tree», «brute»

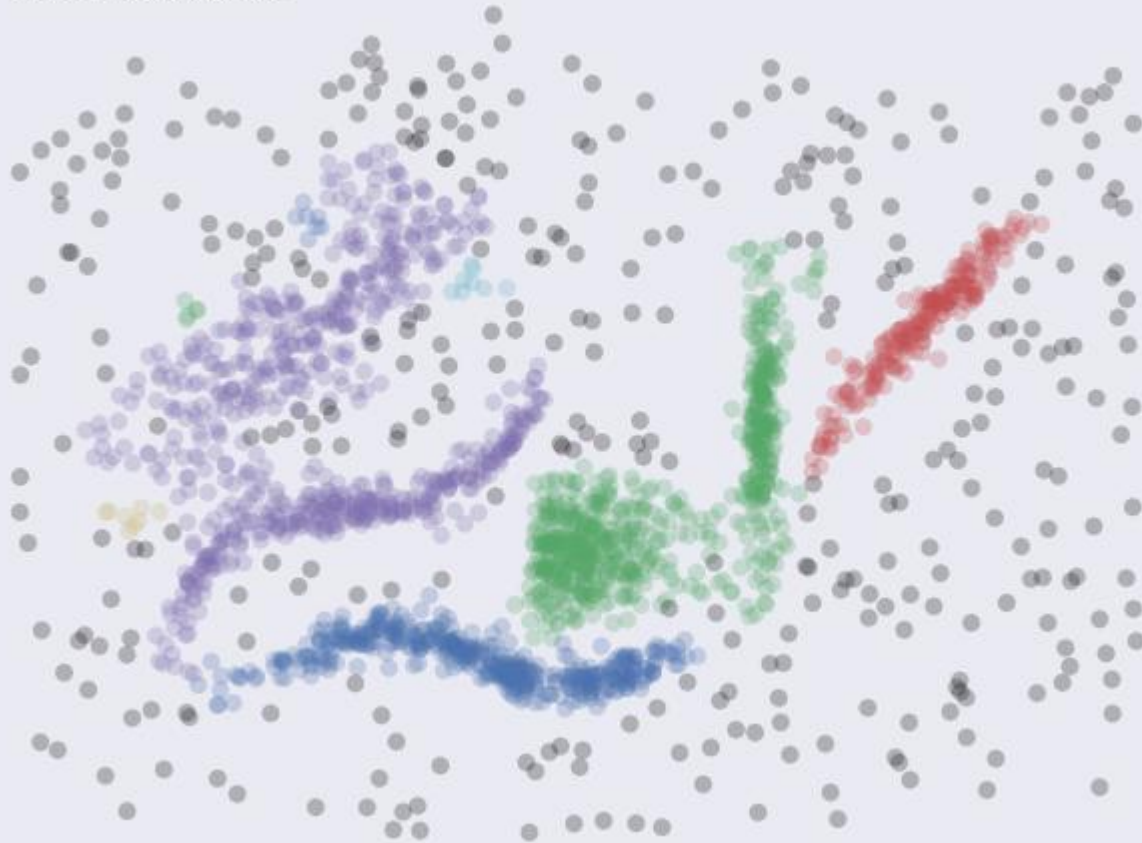
leaf_size – размер листьев в BallTree / KDTree

p – степень в метрике Минковского

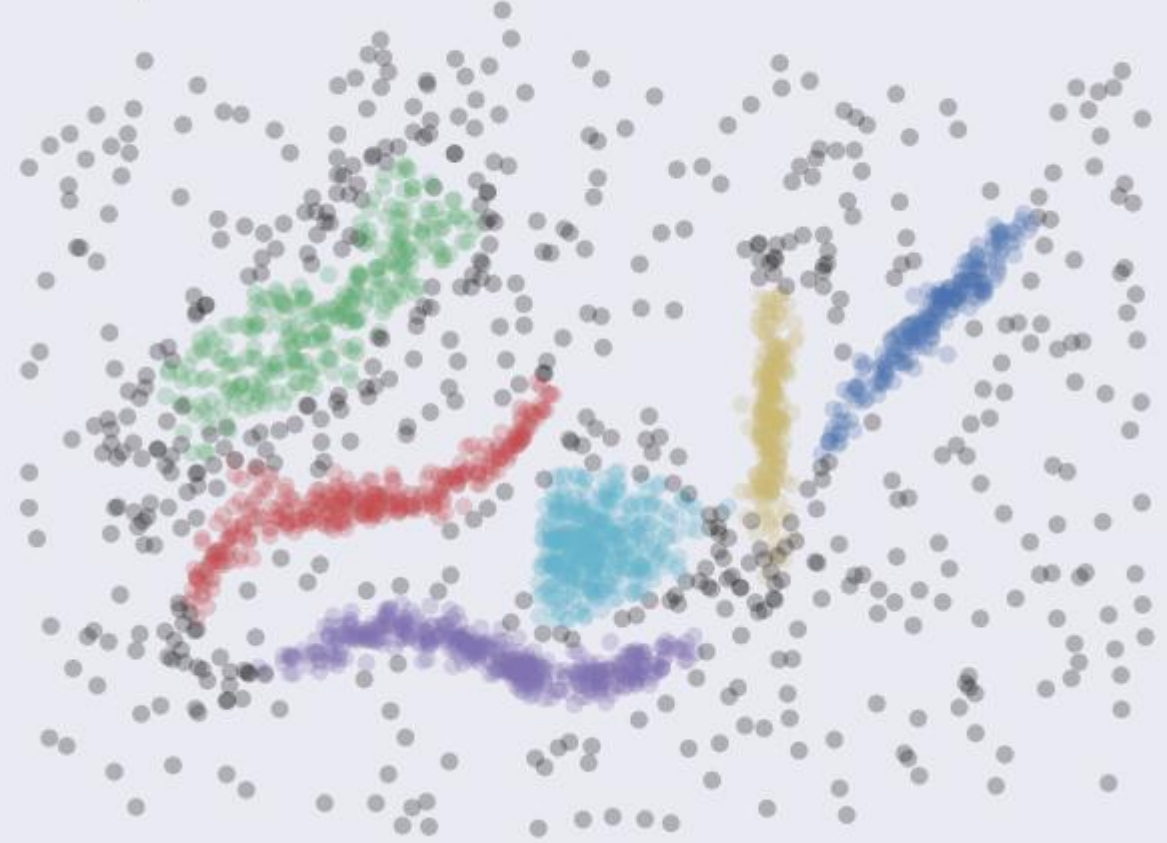
n_jobs

HDBSCAN

Clustering took 0.02 s



Clustering took 0.06 s



есть реализация здесь...

<https://github.com/scikit-learn-contrib>

BIRCH = Balanced Iterative Reducing Clustering using Hierarchies

Tian Zhang, Raghu Ramakrishnan, Maron Livny BIRCH: An efficient data clustering method for large databases. <http://www.cs.sfu.ca/CourseCentral/459/han/papers/zhang96.pdf>

строит дерево с центроидами в листьях

больше структура данных и подход к кластеризации, чем метод

+ эффективный по памяти

спроектирован для больших наборов

в некотором варианте применения сканирует все данные лишь раз

+ онлайн (при динамическом поступлении данных)

локальный – не надо сканировать все данные при отнесении к кластеру

+ есть механизм обнаружения выбросов

– зависит от порядка обработки объектов

BIRCH = Balanced Iterative Reducing Clustering using Hierarchies

Clustering Feature (CF)

$$CF = \left(\sum_{i=1}^m x_i^0, \sum_{i=1}^m x_i, \sum_{i=1}^m x_i^2 \right)$$

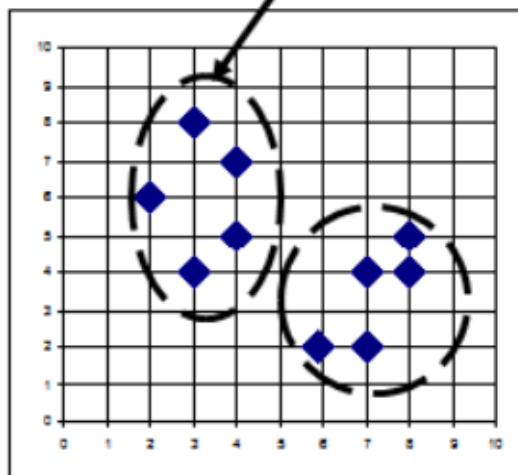
$=m$

на основе CF понятно как вычислить
центроид, радиус, среднее расстояние
между кластерами

$$CF_{A \cup B} = CF_A + CF_B \text{ при } A \cap B = \emptyset$$

Хранить не кластер, а CF!

$$CF = (5, (16,30),(54,190))$$



(3,4)

(2,6)

(4,5)

(4,7)

(3,8)

<https://ru.wikipedia.org/wiki/BIRCH>

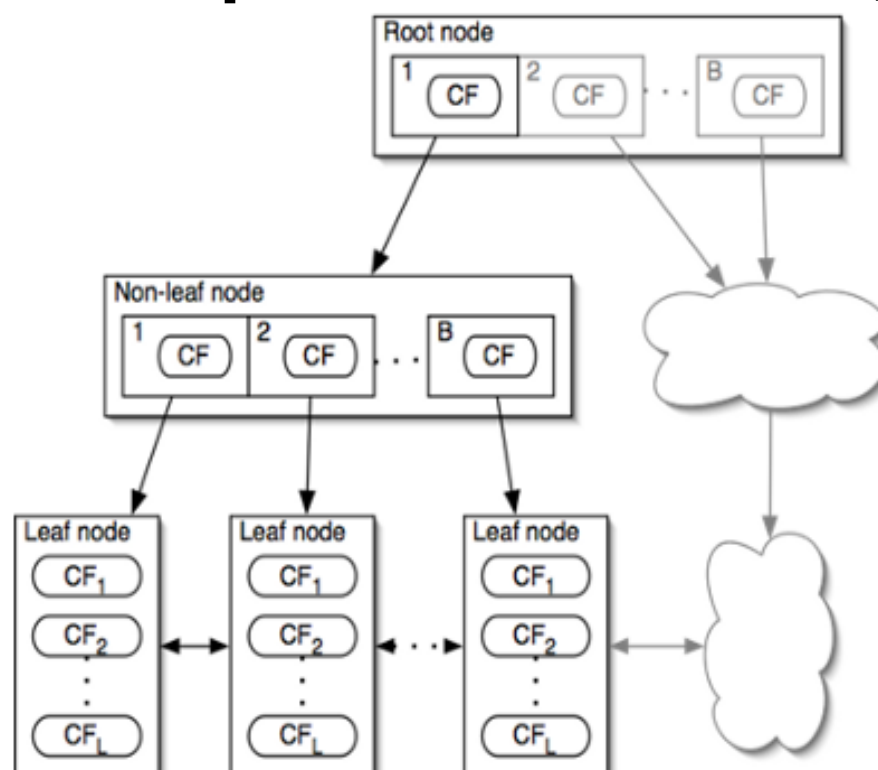
BIRCH = Balanced Iterative Reducing Clustering using Hierarchies

CF-tree – сбалансированное по высоте дерево, в котором хранят CF для иерархической кластеризации, все листья соединены в список (см. рис.)

можем изначально как-то задать, потом можно листья кластеризовать любым алгоритмом и «подстроить»

CF вершины = сумма CF потомков

Два параметра – max число вершин-потомков / max диаметр подкластеров



BIRCH = Balanced Iterative Reducing Clustering using Hierarchies

- 1. Строим начальное CF-дерево (например, просто последовательно добавляя точки)**
точка «падает» по дереву и присоединяется к похожему кластеру
(если расстояние до кластера больше порога – создаёт новый кластер)
можно устранять выбросы, улучшать кластеризацию, запускать кластеризацию в
листьях, объединять листья
 - 2. Кластеризация всех вершин по CF-признакам (агломеративно иерархически),**
точнее, на основе этих признаков задаются специальные расстояния – **не указываем**
дальше можно ещё раз пройтись по выборке, относя точки к ближайшим центроидам,
пересчитать центроиды...
- легко вносить информацию в CF-дерево при пополнении точкой / можно отлавливать
выбросы – $O(n)$**

Если при пополнении слишком большой кластер – расщепляем

<https://towardsdatascience.com/machine-learning-birch-clustering-algorithm-clearly-explained-fb9838cbeed9>

BIRCH = Balanced Iterative Reducing Clustering using Hierarchies

`sklearn.cluster.Birch`

threshold – порог для организации нового подкластера (на радиус кластера)

branching_factor – максимальное число подкластеров в каждом узле (если превышает, то узел расщепляется на два)

n_clusters – число кластеров, если число, то делаем кластеризацию с помощью

AgglomerativeClustering – если `None` не делаем формально окончательную

кластеризацию, если `sklearn.cluster Estimator`, то делаем им кластеризацию подкластеров

compute_labels – вычислять ли метки

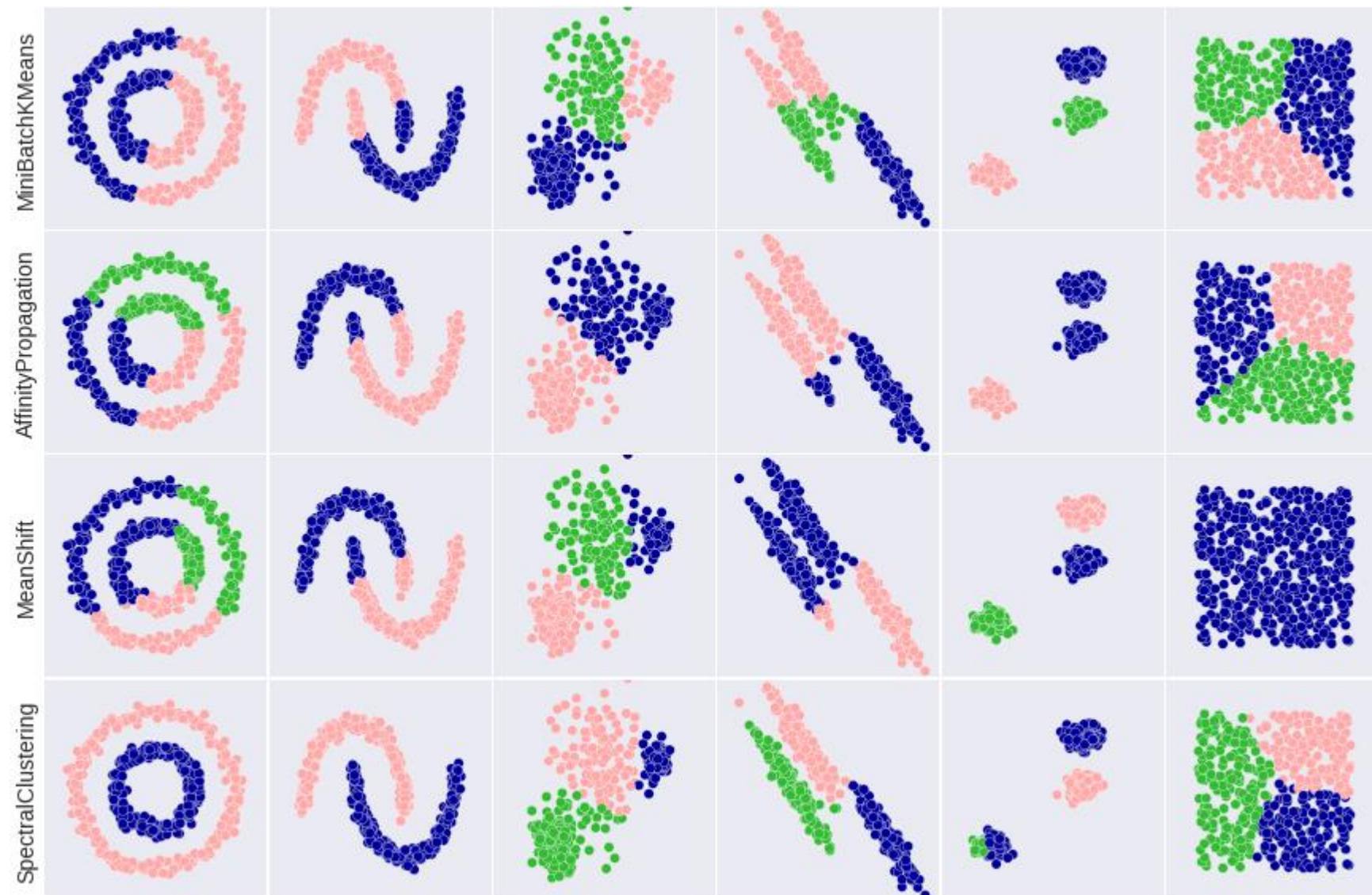
copy – создавать ли копию данных

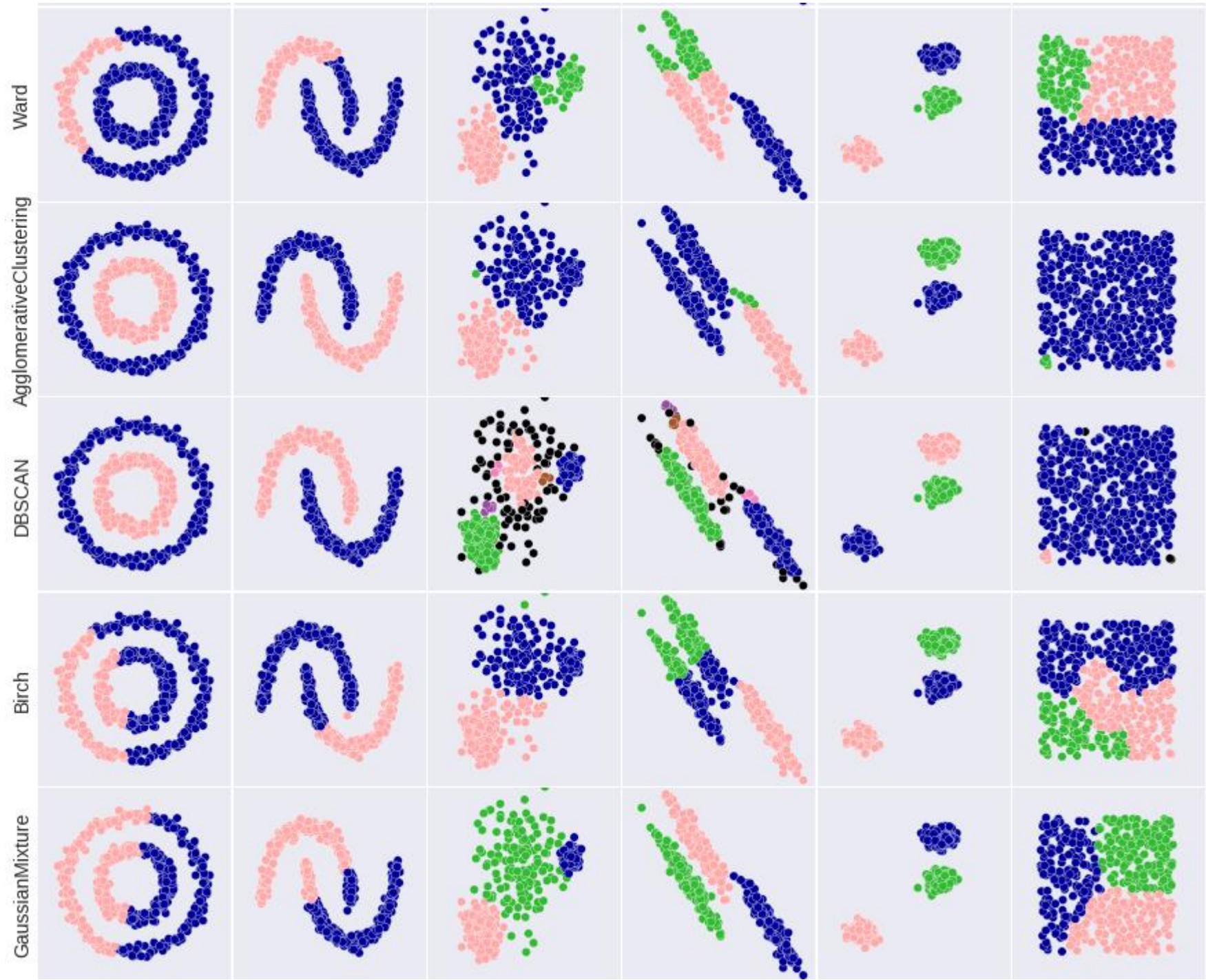
Оригинальная статья

<https://www2.cs.sfu.ca/CourseCentral/459/han/papers/zhang96.pdf>

Метод	Параметры	Использование	Геометрия
k-means	Число кластеров	Простой Популярный Есть обобщения	Кластеры равномошны и шарообразны, их не слишком много
Affinity propagation	Фактор забывания, вероятность стать экземпляром	Много кластеров Не масштабируется	По матрице расстояний, произвольные расстояния
Mean-shift	Параметр ядра	Детектирование выбросов Много кластеров Произвольные распределения Не масштабируется	Нахождение мод плотности
Spectral clustering	Число кластеров	Хороша для большинства модельных задач Мало кластеров Плохо масштабируется	Граф расстояний / матрица сходства
Ward hierarchical / Agglomerative clustering	Число кластеров / тип слияния и метрика / матрица связности	Результат – дендрограмма Много кластеров Можно добавить ограничения на связи Хорошо масштабируется	Расстояния + матрица связности
DBSCAN	Порог близости, число объектов в окрестности	Детектирование выбросов Хорошо масштабируется Неудобные гиперпараметры	Сложная геометрия, разномошные кластеры, но однородная плотность
Gaussian mixtures	много	Специальный случай Не масштабируется	Расстояния Махаланобиса
Birch	Мета-кластеризатор, порог, максимальное число подкластеров	Для больших данных Удаление выбросов Сжатие данных Хорошо масштабируется	Евклидовы расстояния между точками

Сравнение алгоритмов кластеризации





Функции ошибки в кластеризации

Отдельные слайды

Неклассические приложения кластеризации

Как и некоторые другие методы USL:

Признаки для обучения с разметкой

Хорошие признаки для Semi-Supervised Learning

Итоги

Общие подходы при кластеризации:
вычисление центров кластеров – итерационное
оценка распределений (будет потом)
оценка плотности

Иерархические – разбиение / слияние кластеров
Графовые методы

Есть ещё нейростевые... будет потом

Ссылки

Дьяконов, А. Г. «Анализ данных, обучение по прецедентам, логические игры, системы WEKA, RapidMiner и MatLab (практикум на ЭВМ кафедры математических методов прогнозирования)» МАКСПресс 2010 //

<http://www.machinelearning.ru/wiki/images/7/7e/Dj2010up.pdf>

Интересные алгоритмы кластеризации

<https://habr.com/ru/post/322034/>

Ulrike von Luxburg «A Tutorial on Spectral Clustering»

https://people.csail.mit.edu/dsontag/courses/ml14/notes/Luxburg07_tutorial_spectral_clustering.pdf