ТЕХНОСФЕРА

Лекция 2
Алгоритмы кластеризации

Кристина Федоренко

26 сентября 2016 г.

Кристина Федоренко

- March 2012 - July 2015: Data Scientist at Adriver
- August 2015 - March 2016: Data Scientist at DCA
- March 2016 - Present: Data Scientist at Mail.Ru Group

e-mail: k.fedorenko@corp.mail.ru
тел.: +7 (903) 763-26-83

План лекции

- Задача кластеризации
- Иерархическая кластеризация
- Алгоритмы, основанные на плотности: dbscan и optics

Задача классификации

Задача кластеризации

Задача кластеризации

Задачу кластеризации описывает следующий набор утверждений.

- Даны признаки об объектах
- Нет целевой переменной
- **Цель** – поиск структуры в данных
- **Как** – путем разбиения множества объектов на группы (кластеры). Таким образом, чтобы объекты внутри групп были "похожи"друг на друга.

Задача кластеризации

Дано.
Признаковые описания N объектов $\mathbf{x} = (x_1, \ldots, x_m) \in \mathcal{X}$, образующие обучающую выборку X
$\rho : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ – функция расстояния между объектами.

Найти. Модель из семейства параметрических функций

$$H = \{h(\mathbf{x}, \theta) : \mathcal{X} \to \mathcal{Y} \mid \mathcal{Y} = \{1, \ldots, K\}\},$$

ставящую в соответствие произвольному $\mathbf{x} \in \mathcal{X}$ один из K кластеров так, чтобы расстояние между объектами одного кластера было небольшим.

Применение

- Позволяет больше узнать о данных. Анализ социальных сетей: кластеризация позволяет определять сообщества пользователей по определенным признакам

Применение

- Позволяет конструировать новые признаки. Например, кластеризация доменов позволяет бороться с сильной разреженностью данных в задачах классификации

Users	lamoda.ru	bonprix.ru	auto.mail.ru	cars.ru
$user_1$	0	0	0	1
$user_2$	1	0	0	0
$user_3$	0	1	0	0
$user_4$	0	0	1	0

Users	$cluster_1$	$cluster_2$
$user_1$	0	1
$user_2$	1	0
$user_3$	1	0
$user_4$	0	1

Применение

- Работать с кластерами удобнее, чем с отдельными объектами

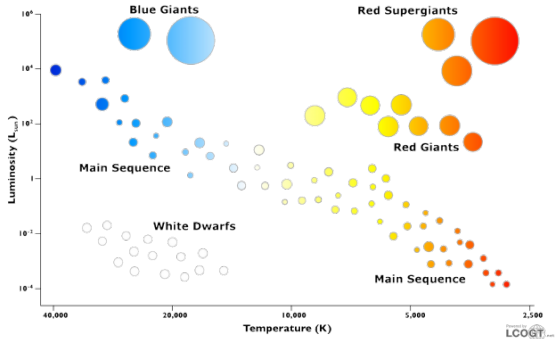
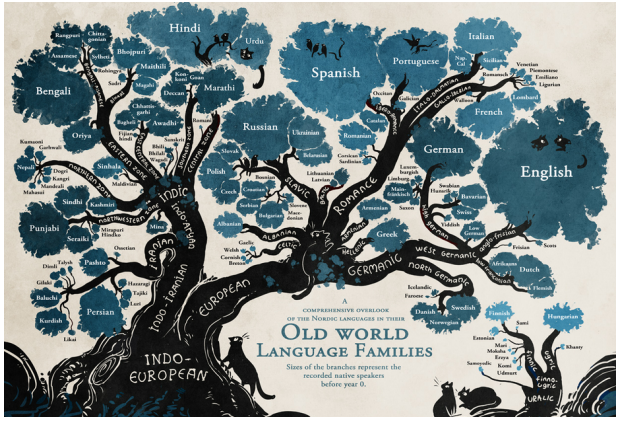


Рис.: Диаграмма Герцшпрунга — Рассела[1]

[1] https://lcogt.net/spacebook/h-r-diagram/

Иерархическая кластеризация[2]



[2]Feast Your Eyes on This Beautiful Linguistic Family Tree

Иерархическая кластеризация: идея метода

Agglomerative

1. Инициализация – когда каждый объект – отдельный кластер
2. на каждом шаге совмещаем два наиболее близких кластера
3. останавливаемся, когда получаем требуемое количество. Например, требуемое количество кластеров может быть единицей.

Divisive

1. Инициализация – все объекты составляют один кластер
2. на каждом шаге разделяем один из кластеров пополам
3. останавливаемся, когда получаем требуемое количество. Например, требуемое количество кластеров может быть N – количество элементов

Агломеративный алгоритм

```
function agglomerative(X, K):
    initialize N # number of objects
    initialize C = N # number of clusters
    initialize C_i = x_i # initial clusters
    while C > K:
        C_a = C_b = None # closest clusters
        min_dist = +inf # distance between closest
        for i in 1 .. C:
            for j in i + 1 .. C:
                dist = d(C_i, C_j) # dist. betw. clusters
                if dist < min_dist:
                    min_dist = dist
                    C_a = C_i
                    C_b = C_j
        merge(C_a, C_b)
        C = C − 1
    return C_1, ..., C_K
```

память $O(N)$, сложность $O(N^3)$

Расстояние между кластерами

- single-linkage

$$d_{min}(C_i, C_j) = \min_{\mathbf{x} \in C_i, \mathbf{x}' \in C_j} \|\mathbf{x} - \mathbf{x}'\|$$

- complete-linkage

$$d_{max}(C_i, C_j) = \max_{\mathbf{x} \in C_i, \mathbf{x}' \in C_j} \|\mathbf{x} - \mathbf{x}'\|$$

- average linkage

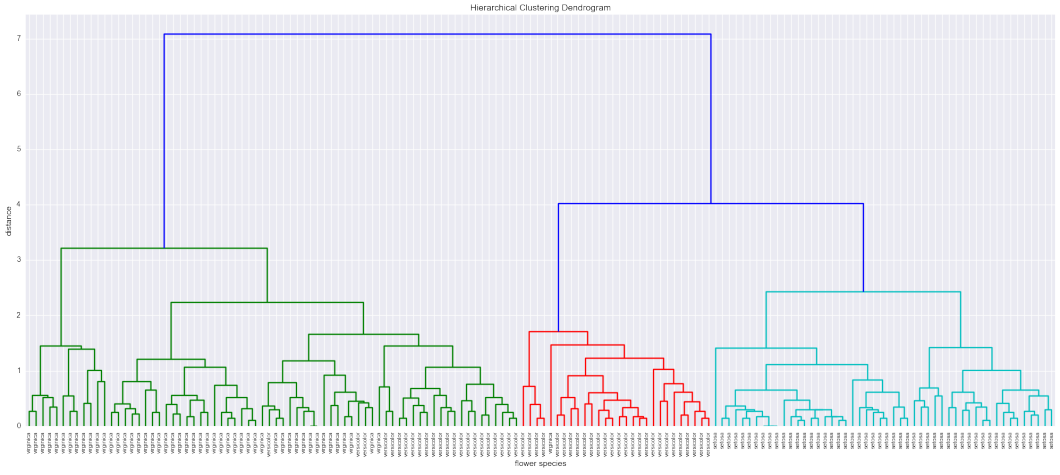
$$d_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{\mathbf{x} \in C_i} \sum_{\mathbf{x}' \in C_j} \|\mathbf{x} - \mathbf{x}'\|$$

- centroid linkage

$$d_{mean}(C_i, C_j) = \|\mathbf{m}_i - \mathbf{m}_j\|$$

Визуализация

Результат иерархической кластеризации представляется в виде дендограммы



Быстрый алгоритм

```
function fast_agglomerative(X, K):
    initialize N # number of objects
    initialize C = N # number of clusters
    initialize C_i = x_i # initial clusters
    initialize delta_set = get_delta_set(C_i)
    while C > K:
        C_a = C_b = None # closest clusters
        min_dist = +inf # distance between closest
        for C_i in delta_set:
            for C_j in delta_set:
                dist = d(C_i, C_j) # dist. betw. clusters
                if dist < min_dist:
                    min_dist = dist
                    C_a = C_i; C_b = C_j
        new_cluster = merge(C_a, C_b)
        update_delta_set(C_i, new_cluster)
        C = C − 1
        if delta_set is empty:
            delta_set = get_delta_set(C_i)
    return C_1, ..., C_K
```

Быстрый алгоритм

delta-set – набор кластеров расстояние между которыми меньше δ Как реализовать функцию get-delta-set?

- Если $C <= K_1$, то delta-set – это все C_i
- Иначе выбрать K_2 случайных расстояний между кластерами, $\delta =$ минимальному из них
- K_1, K_2 влияют только на скорость, но не на результат кластеризации; рекомендованные значения $K_1 = K_2 = 20$.

Иерархическая кластеризация: итог

+ Несферические кластеры
+ Разнообразие критериев
+ Любые K из коробки
— Требует много ресурсов

Алгоритмы, основанные на плотности

Идея метода[3]

- Кластеризация, основанная на плотности объектов
- Кластеры – участки высокой плотности, разделенные участками низкой плотности

[3] http://biarri.com/spatial-clustering-in-c-post-2-of-5-running-dbscan/

Определения

Плотность

Количество объектов внутри сферы заданного радиуса ε

Core-объект

Объект **x** является core-объектом, если плотность вокруг него больше *min_pts*

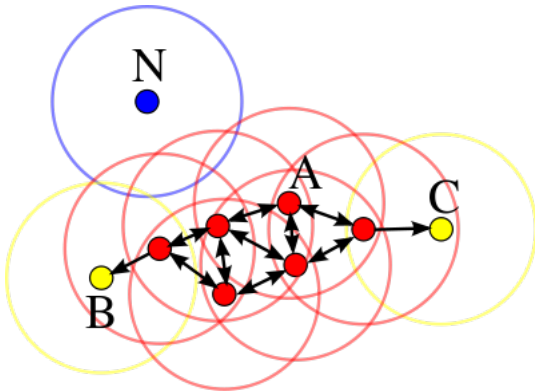
Граничный-объект

Объект **x** является граничным-объектом, если плотность вокруг него меньше *min_pts*, но он находится внутри сферы заданного радиуса вместе с хотя бы одним core-объектом

Шум

Объект **x** является шумом, если он не является ни core-объектом, ни граничным объектом

Виды объектов



DBSCAN 1

```
function dbscan(X, eps, min_pts):
    initialize NV = X # not visited objects
    for x in NV:
        remove(NV, x) # mark as visited
        nbr = neighbours(x, eps) # set of neighbours
        if nbr.size < min_pts:
            mark_as_noise(x)
        else:
            C = new_cluster()
            expand_cluster(x, nbr, C, eps, min_pts, NV)
            yield C
```

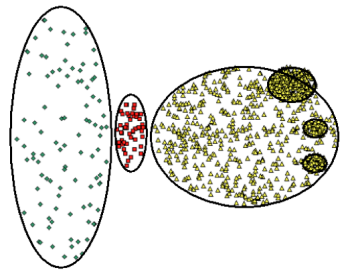
DBSCAN 2

```
function expand_cluster(x, nbr, C, eps, min_pts, NV):
    add(x, C)
    for x1 in nbr:
        if x1 in NV: # object not visited
            remove(NV, x1) # mark as visited
            nbr1 = neighbours(x1, eps)
            if nbr1.size >= min_pts:
                # join sets of neighbours
                merge(nbr, nbr_1)
        if x1 not in any cluster:
            add(x1, C)
```

Сложность: $O(n^2)$ или $O(n \log n)$ (R^*Tree)
Память: $O(n)$ или $O(n^2)$

DBSCAN: итог и демо[4]

+ не требует K
+ кластеры произвольной формы
+ учитывает выбросы
— Не вполне детерминированный
— Не работает при разных плотностях кластеров



[4] http://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

OPTICS

Ordering points to identify the clustering structure

Идея состоит в том чтобы обрабатывать регионы с высокой плотностью первыми.

Optics использует две дополнительные метрики

$$\text{core-dist}(p) = \begin{cases} \text{UNDEFINED} & \text{if } |N_\varepsilon(p)| < \mathit{MinPts} \\ \mathit{MinPts}\text{-th smallest distance to } N_\varepsilon(p) & \text{otherwise} \end{cases}$$

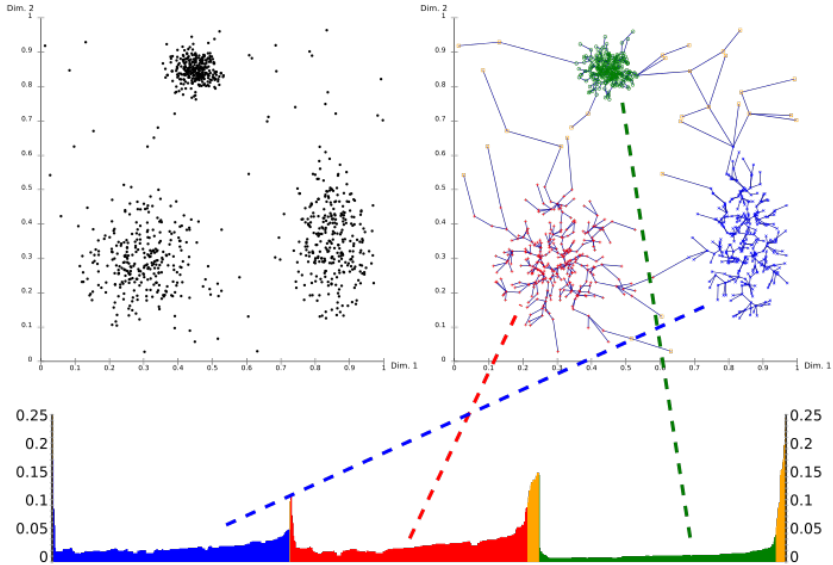
$$\text{reachability-dist}(o, p) = \begin{cases} \text{UNDEFINED} & \text{if } |N_\varepsilon(p)| < \mathit{MinPts} \\ \max(\text{core-dist}(p), \text{dist}(p, o)) & \text{otherwise} \end{cases}$$

OPTICS 1

```
1  function optics(X, eps, min_pts)
2      for each point x in X
3         x.reachability-distance = UNDEFINED
4      for each unprocessed point x in X
5         N = getNeighbors(x, eps)
6         mark x as processed
7         output x to the ordered list
8         if (core-distance(x, eps, min_pts) != UNDEFINED)
9            Seeds = empty priority queue
10           update(N, x, Seeds, eps, min_pts)
11           for each next z in Seeds
12              N' = getNeighbors(z, eps)
13              mark z as processed
14              output z to the ordered list
15              if (core-distance(z, eps, min_pts) != UNDEFINED)
16                 update(N', z, Seeds, eps, min_pts)
```

OPTICS 2

```
1  function update(N, x, Seeds, eps, min_pts)
2      coredist = core-distance(x, eps, min_pts)
3      for each z in N
4         if (z is not processed)
5            new-reach-dist = max(coredist, dist(x, z))
6            # z is not in Seeds
7            if (z.reachability-distance == UNDEFINED)
8                z.reachability-distance = new-reach-dist
9                Seeds.insert(o, new-reach-dist)
10           # z in Seeds, check for improvement
11           else
12               if (new-reach-dist < z.reachability-distance)
13                  z.reachability-distance = new-reach-dist
14                  Seeds.move-up(z, new-reach-dist)
```



Сложность $O(n \log(n))$

Вопросы