Кластеризация. Графовый подход, EM и k means, иерархическая кластеризация

Сергей Лисицын

lisitsyn.s.o@gmail.com

6 апреля 2011 г.

Задача кластеризации
Clustering

- Имеется некоторое множество объектов
- Решением задачи является некоторое разбиение множества на непересекающиеся кластеры
- Кластеры – некоторые характерные группы объектов
- Количество кластеров для разбиения может быть задано изначально
- Эффективность разбиения оценивается функционалами внутриклассового и межклассового расстояния
- Задача ещё более некорректна чем классификация

Общая задача кластеризации

Пусть имеется некоторое множество объектов $\mathcal{X}$ и конечное множество кластеров $\mathcal{C}$. Необходимо найти функциональную зависимость

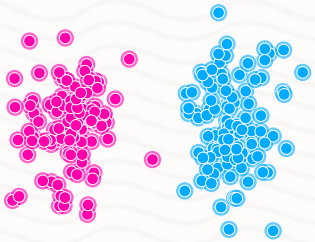
$$f : \mathcal{X} \to \mathcal{C},$$

причём (в отличие от классификации) ни одно значение f на $\mathcal{X}$ априори не известно.

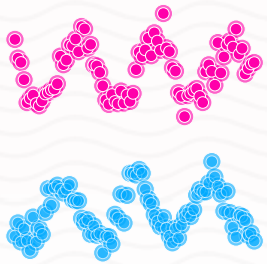
В **задаче кластеризации** функция f должна быть построена таким образом, чтобы выделять наиболее характерные группы объектов.

Свойства кластеров

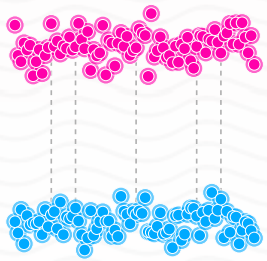
Компактность (compactness): кластеры компактны – объекты лежат внутри некоторого плотного множества



Связность (connectivity): объекты кластера можно выделить в связные структуры



Пространственное разделение (spatial separation): кластеры отстоят друг от друга на некотором расстоянии



Функционалы качества кластеризации

- Межкластерное расстояние

$$\rho_c(c_1, c_2) = \sum_{f(x_1)=c_1, f(x_2)=c_2} \rho(x_1, x_2)$$

Эффективная кластеризация подразумевает максимизацию межкластерных расстояний: $\sum_{\substack{c_1 \neq c_2 \\ c_1, c_2 \in C}} \rho_c(c_1, c_2) \to \max$

- Внутрикластерное расстояние

$$\rho_{\text{inner}}(c) = \sum_{f(x_1)=f(x_2)=c} \rho(x_1, x_2)$$

Эффективная кластеризация подразумевает минимизацию внутрикластерных расстояний: $\sum_{c \in C} \rho_{\text{inner}}(c) \to \min$

- В некоторых случаях минимизируется отношение межкластерного и внутрикластерного расстояний

Цели кластеризации

- Выделение наиболее характерных групп объектов
- Выяснение структуры множества объектов
- Сокращение объема выборки объектов
- Обнаружение «необычных» объектов (novelty detection)

Графовый подход к кластеризации

- Множество объектов в метрическом пространстве представимо в виде неориентированного взвешенного графа, где вес ребра это значение метрики между его объектами-вершинами
- Кластеры в таком графе – непересекающиеся подграфы исходного
- Графы сами по себе достаточно удобны для кластеризации
- Представление объектов в таком случае знать не обязательно, достаточно расстояний между ними, вычисленных даже косвенно
- Кроме того, такие графы-кластеры удобно визуализировать

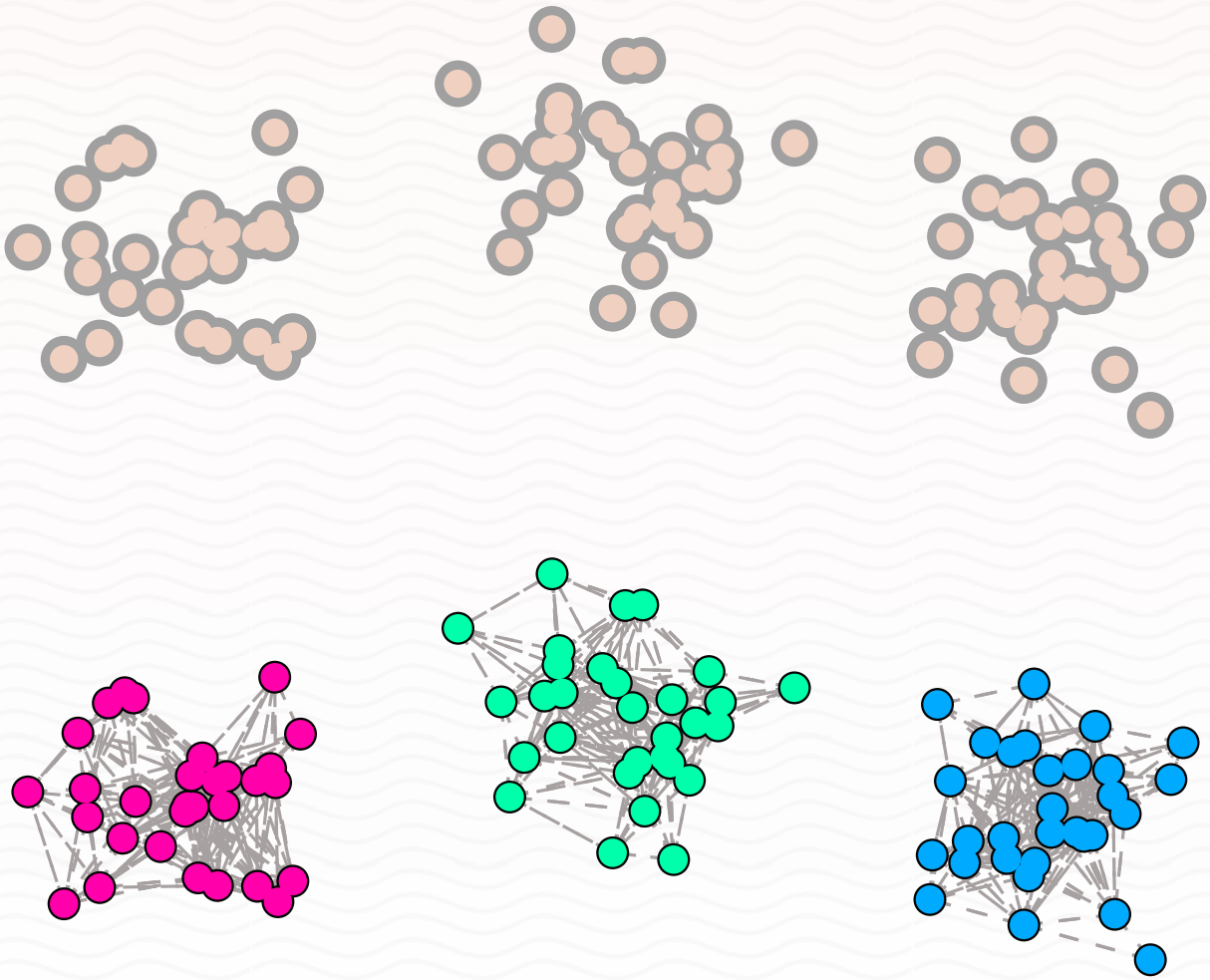
Алгоритм выделения связных компонент

- Кластеры в пространстве можно представлять как группы связных объектов некоторого графа – связные компоненты
- Связная компонента – подграф, в котором все вершины взаимно достижимы
- Удаляя (или не создавая) ребра графа с весом, большим, чем некоторый параметр R_{max}, можно разбить граф на некоторое количество связных компонент
- В распавшемся графе будут выделены некоторые характерные структуры с минимальным внутрикластерным расстоянием

Описание алгоритма выделения связных компонент

Вход: множество объектов $\mathcal{X}$, параметр R_{max}
Выход: множество пар (объект-кластер)

1. Создать граф без ребёр с узлами, соответствующими элементам множества $\mathcal{X}$
2. Соединить все узлы графа, соответствующие элементам $x_i, x_j \in \mathcal{X}, \forall_{i \neq j}$, для которых $\rho(x_i, x_j) < R_{max}$
3. Возвратить соответствующие вершинам графа пары (элемент множества $\mathcal{X}$ – номер связной компоненты)



```python
class ConnectedComponentClusterer():
    def clusterize(self,objs,R_max):
        cluster_graph = graph()
        N = len(objs)
        cluster_graph.add_nodes(range(N))
        for i in range(N):
            for j in range(N):
                if not i==j and metric(objs[i],objs[j])<R_max:
                    try: cluster_graph.add_edge((i,j))
                    except: pass
        component_idxs = connected_components(cluster_graph)
        clusters = defaultdict(list)
        for i in component_idxs.keys():
            clusters[component_idxs[i]].append(objs[i])
```

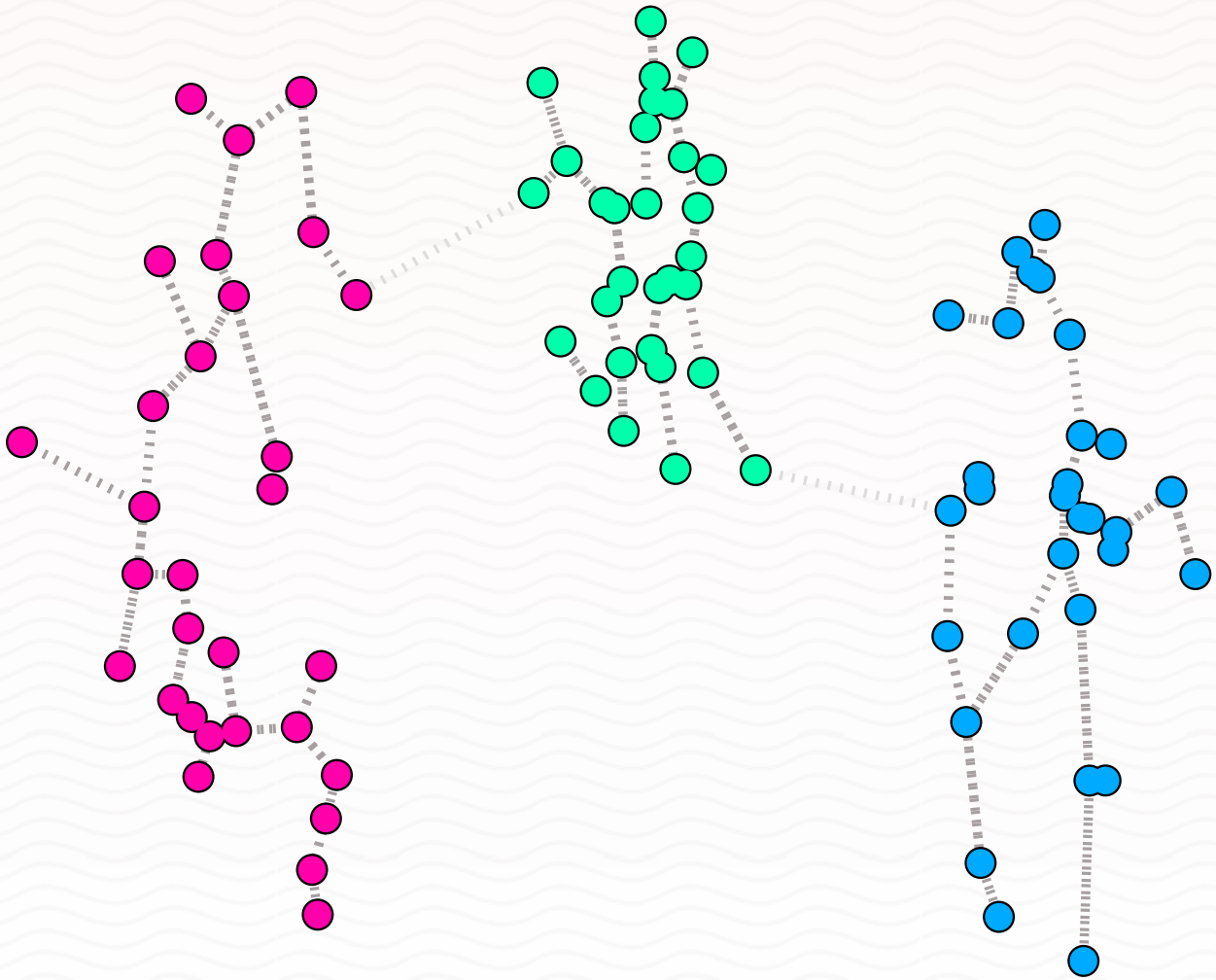
Алгоритм построения покрывающего дерева
Spanning tree clustering

- Минимальное покрывающее дерево – древовидная структура с минимальным суммарным расстоянием между соседними объектами
- При удалении из построенного дерева $K - 1$ самых «длинных» ребёр максимизируется межкластерное расстояние
- Минимум суммарного расстояния между объектами минимизирует внутрикластерное расстояние
- Количество кластеров в таком случае задаётся заранее, алгоритм ведёт себя предсказуемее

Описание алгоритма кластеризации с помощью минимального покрывающего дерева

Вход: множество объектов $\mathcal{X}$, параметр K – количество кластеров
Выход: множество пар (объект-кластер)

1. Построить минимальное покрывающее дерево с узлами $0, 1, \ldots,$ для которого
$$\sum_{i,j} \rho(x_i, x_j) \to \min$$
2. Удалить $K - 1$ ребёр (i, j) максимальной метрики $\rho(x_i, x_j)$ построенного покрывающего дерева
3. Возвратить соответствующие вершинам графа пары (элемент множества $\mathcal{X}$ – номер связной компоненты)

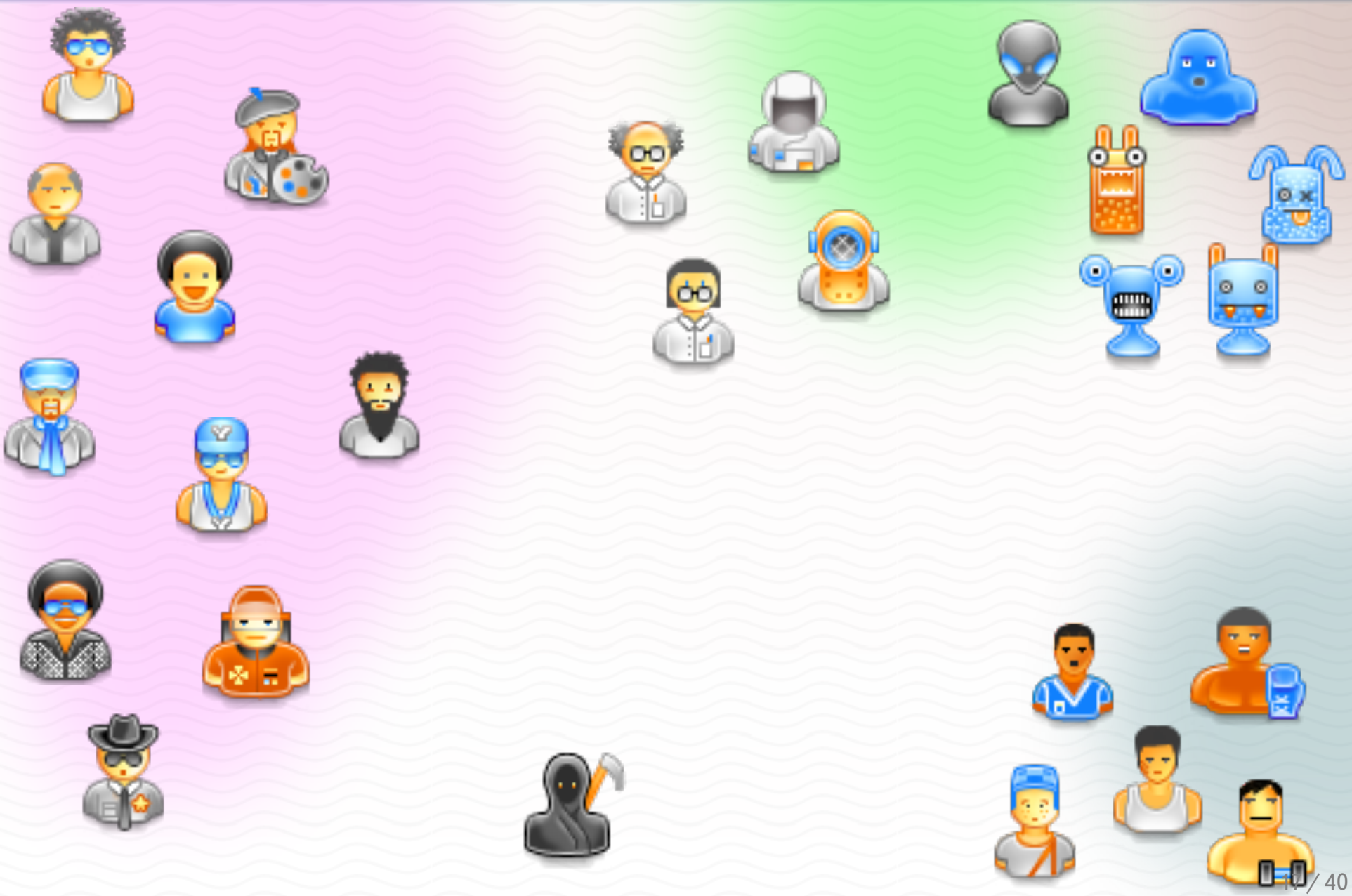


```python
class SpanningTreeClusterer():
    def clusterize(self,objs,metric,K):
        N = len(objs)
        c_graph = graph()
        c_graph.add_nodes(range(N))
        for i in range(N):
          for j in range(i,N):
            c_graph.add_edge((i,j),wt=metric(objs[i],objs[j]))
        stree, c_graph = minimal_spanning_tree(c_graph), graph()
        c_graph.add_nodes(range(N))
        del stree[stree.keys()[0]]
        for i in stree.keys():
          c_graph.add_edge((i,stree[i]),
                           wt=metric(objs[i],objs[stree[i]]))
        for _ in range(K-1):
          max_edge = max(c_graph.edges(),
                         key=lambda x: metric(objs[x[0]],objs[x[1]]))
          c_graph.del_edge((max_edge))
        component_idxs = connected_components(c_graph)
        clusters = defaultdict(list)
        for i in component_idxs.keys():
          clusters[component_idxs[i]].append(objs[i])
        return clusters
```

Пример социальной кластеризации

- Объекты пространства – пользователи социальной сети (электронной почты, ICQ, etc)
- Никакой информации о самих объектах не имеется, можно определить только расстояния между объектами
- Метрика между двумя пользователями может определяться как обратное нормированное количество сообщений между ними
- Результатом кластеризации будут группы людей, наиболее тесносвязанные друг с другом (можно на это надеяться)

Пример социальной кластеризации



Эффективность алгоритмов

- Для обоих алгоритмов необходимо найти все взаимные метрики за $O(n^2)$, выделение кластеров выполняется поиском в ширину или глубину $O(|V| + |E|)$ (однако для построения покрывающего дерева существуют особые алгоритмы)
- Оба алгоритма неустойчивы к близкорасположенным кластерам – нет «баланса» между уменьшением функционалов качества, минимизируется только внутрикластерное расстояние
- Алгоритм выделения связных компонент плохо обусловлен по своему параметру R_{max}: небольшие изменения R_{max} могут повлечь резкое увеличение или уменьшение количества кластеров
- Алгоритм кластеризации с помощью покрывающего дерева выделяет заданное количество кластеров

Смеси распределений

- Выборка с группами может рассматриваться как группа нескольких многомерных распределений, такое предположение задаёт общую плотность распределения

$$p(x) = \sum_c P(c)p_c(x),$$

где $p_c(x)$ – функция плотности распределения кластера c, а $P(c)$ – априорная вероятность класса

- Кластер объектов – компонента этой смеси распределений с плотностью f_c
- Кластеризация сводится к нахождению компонент смеси распределений по эмпирическим оценкам плотности, а принадлежность конкретных объектов определяется максимумом $P(c)p_c(x)$

EM-алгоритм
Expectation-Maximization

- Описан в 1977 году Демпстером, Лэйрдом и Рабином и предназначен для оценок максимального правдоподобия в статистических моделях
- В случае кластеризации EM-алгоритм позволяет найти компоненты смеси распределений
- По полученным компонентам смеси распределений классификация производятся достаточно легко
- Для кластеризации, как правило используют гауссианы:

$$p_c(x) = \mathcal{N}(x, \mu_c, \mathbf{R}_c)$$

- Алгоритм итеративен и выполняется до тех пор, пока всем объектам не будут назначены «стабильные» кластеры, не изменяющиеся в течение следующих итераций

Вход: множество объектов $\mathcal{X}$, количество кластеров k
Выход: множество пар (объект-кластер)

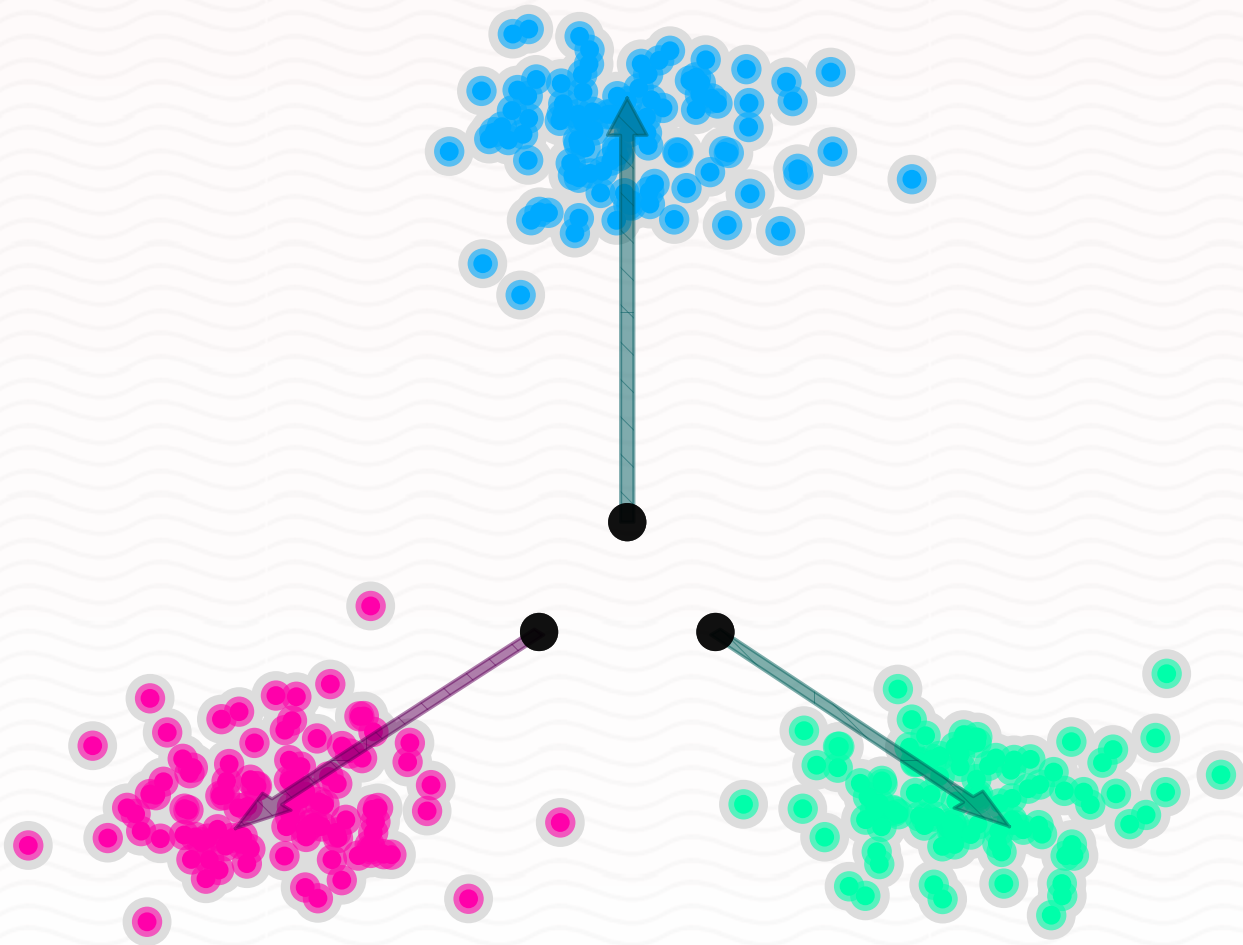
1. Задать начальные приближения $P(c) = 1/k$, за μ_c принять случайные объекты выборки, рассчитать $diag\ \mathbf{R_c} = \frac{\sum_{x\in\mathcal{X}} g_c(x)(x-\mu_c)^2}{|\mathcal{X}|P(c)}$
2. До тех пор, пока кластеры не стабилизируются:
   - 2.1 E-шаг
     Вычисление апостериорных вероятностей
     $$g_c = \frac{P(c)\mathcal{N}(x, \mu_c, \mathbf{R}_c)}{\sum_k P(k)\mathcal{N}(x, \mu_k, \mathbf{R}_k)}$$
   - 2.2 M-шаг
     Пересчёт параметров гауссианов
     $$P(c) = \frac{\sum_x g_c(x)}{|\mathcal{X}|}, \quad \mu_c = \frac{\sum_x g_c(x)x}{|\mathcal{X}|P(c)}, \quad diag\ \mathbf{R_c} = \frac{\sum_{x\in\mathcal{X}} g_c(x)(x-\mu_c)^2}{|\mathcal{X}|P(c)}$$
   - 2.3 Для каждого объекта выбрать кластер $\arg\max_c g_c$

Эффективность

- Алгоритм EM в самом общем виде не очень практичен
- В случае плохого выбора скрытых переменных и неправильной оценки количества кластеров алгоритм плохо сходится
- Алгоритм очень трудоёмок в вычислениях
- Итеративная природа алгоритма требует определять либо количество итераций, либо критерий останова

Алгоритм кластеризации k средних
k means clustering

- Является упрощенным вариантом EM-алгоритма
- Идея алгоритма состоит в «стабилизации» центров кластеров на M-шаге и жёсткой привязке объектов к ближайшему центру на E-шаге
- Алгоритм итеративен: выбирается либо количество итераций, либо условие сходимости
- Выделяет только заданное количество компонент, но существуют модификации для переменного параметра k



Описание алгоритма k средних

Вход: множество объектов $\mathcal{X}$, количество центроидов k, количество итераций или условие останова алгоритма
Выход: множество пар (объект-кластер)

1. Занести в множество центроидов **C** случайные объекты из $\mathcal{X}$, не изменяя $\mathcal{X}$
2. До тех пор, пока центроиды не стабилизировались (или не выполнилось заданное количество итераций):
   - 2.1 Для каждого объекта $x \in \mathcal{X}$: задать объекту x ближайший к нему центроид c: $f(x) = c,\ \rho(c, x) \to \min_c$
   - 2.2 Для каждого центроида $c \in \mathbf{C}$: пересчитать положение

$$c = \frac{\sum_{x, f(x)=c} x}{\sum_{x, f(x)=c} 1}$$

3. Вернуть пары (объект из $\mathcal{X}$ – номер ближайшего к нему центроида)

```
class KMeansClusterer():
    def clusterize(self,objects,k,max_iter=10):
        cs = numpy.array(random.sample(objects,k))
        for iter in range(max_iter):
            new_cs = numpy.zeros([k,2])
            counts = numpy.zeros(len(cs))
            for x in objects:
                nearest_c = min(cs,key=lambda c: metric(c,x))
                for (index, centroid) in enumerate(cs):
                    if numpy.all(centroid== nearest_c):
                        new_cs[index] += x
                        counts[index] += 1
        cs = numpy.array(map(lambda x,y: x/y, new_cs, counts))
        clusters = [[c] for c in cs]
        for x in objects:
            nearest_c = min(cs,key=lambda c: metric(c,x))
            for cluster in clusters:
                if numpy.all(cluster[0] == nearest_c):
                    cluster.append(x)
        for cluster in clusters:
           cluster.pop(0)
        return clusters
```

Алгоритм k средних

k means

- k means – один из самых популярных алгоритмов кластеризации и находит применение во многих задачах
- Сходимость сильно зависит от начального выбора центроидов
- Композиция алгоритма с каким-либо более простым алгоритмом может помочь достичь лучшей сходимости
- Каждая итерация линейна по размеру выборки – $O(n)$
- Существует множество различных модификаций для лучшего выбора центроидов и лучшей сходимости

Иерархический подход

- Из кластеров может быть составлена иерархия: в вершине иерархии множество всех объектов, ниже группы объектов, в самом низу
- На любом уровне иерархии можно получить нужную степень разбиения на кластеры
- Метрика между объектами вполне однозначна, но метрику между кластерами следует определить каким-либо образом

Межкластерные расстояния

- Расстояние между ближайшими элементами (single linkage)

$$\rho(c_1, c_2) = \min_{x_1 \in c_1, x_2 \in c_2} \rho(x_1, x_2)$$

- Расстояние между самыми дальними элементами (complete linkage)

$$\rho(c_1, c_2) = \max_{x_1 \in c_1, x_2 \in c_2} \rho(x_1, x_2)$$

- Среднее групповое расстояние (group linkage)

$$\rho(c_1, c_2) = \frac{\sum_{x_1 \in c_1, x_2 \in c_2} \rho(x_1, x_2)}{|c_1| \cdot |c_2|}$$

Агломеративный подход к кластеризации
Agglomerative clustering

- Наиболее простой метод построения иерархии классов
- Изначально каждый объект содержится в отдельном кластере
- Последовательным объединением двух кластеров с минимальной взаимной метрикой
- С каждой итерацией количество кластеров уменьшается, процесс останавливается при достижении нужного их количества

Агломеративная кластеризация

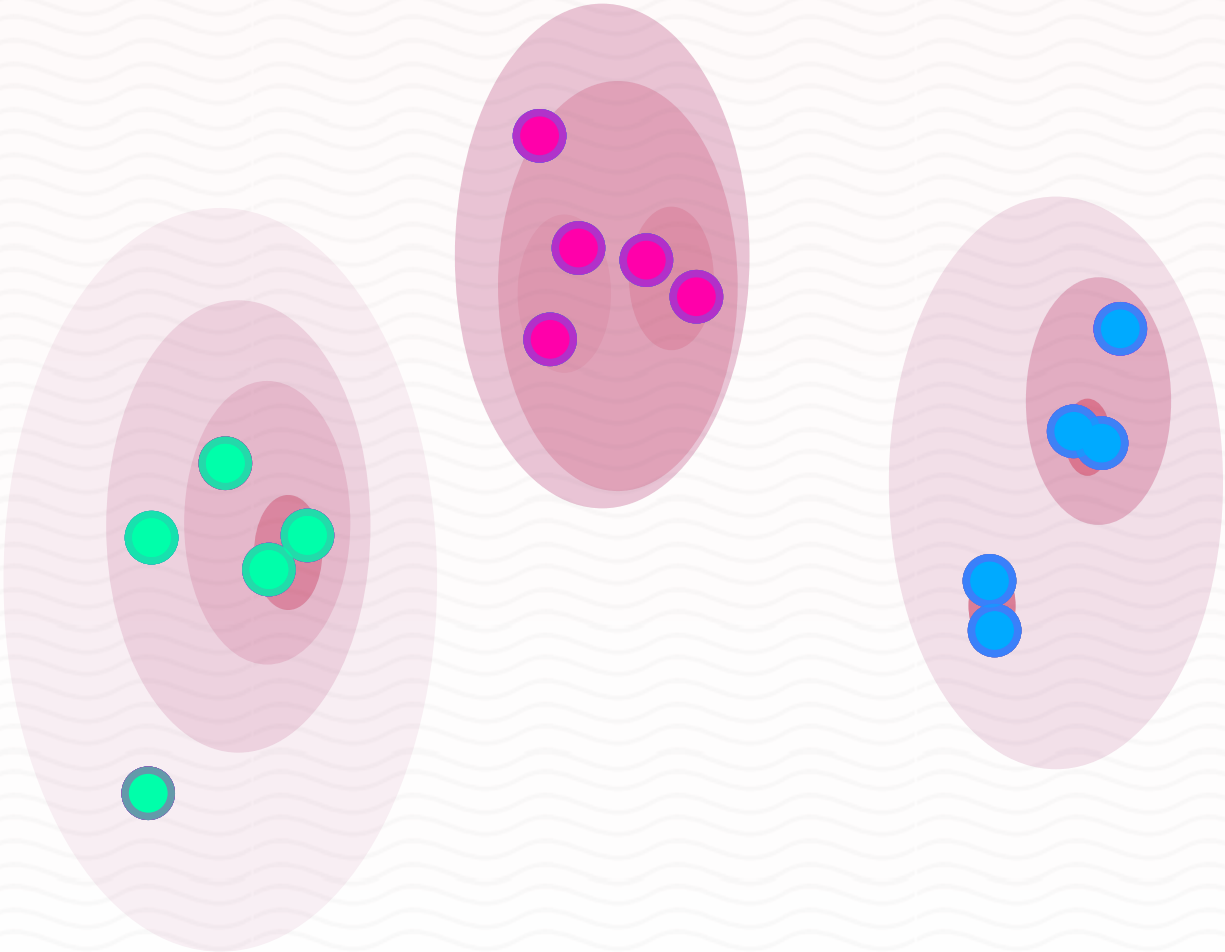
Вход: множество объектов $\mathcal{X}$, параметр k количества кластеров
Выход: множество пар (объект-кластер)

- Создать множество C, состоящее из одноэлементных кластеров, элементов множества $\mathcal{X}$
- Пока количество кластеров больше, чем k
  - Выбрать два наиболее близких по метрике кластера из C:

$$(S_1, S_2) = \underset{S_1, S_2 \in C, S_1 \neq S_2}{\arg\min} \rho(S_1, S_2)$$

  - Заменить в C кластеры S_1, S_2 кластером $S_1 \cup S_2$, то есть объединить их в один, исключая исходные
- Возвратить пары (объект из $\mathcal{X}$ – номер кластера, в который попал этот объект)

```
class AglomerativeClusterer():
    def clusterize(self,objects,till,metric):
        clusters = [[x] for x in objects]
        iteration = 0
        while len(clusters)>till:
            n_idx = (0,1)
            n_dist = metric(clusters[0],clusters[1])
            for i, cluster_lhs in enumerate(clusters):
                for j, cluster_rhs in enumerate(clusters):
                    m = metric(cluster_lhs,cluster_rhs)
                    if n_dist>m and not i==j:
                        n_dist = m
                        n_idx = (i,j)
            aglomerated = clusters[n_idx[0]]+clusters[n_idx[1]]
            clusters = \
            [clusters[i] for i in range(len(clusters)) \
            if i != n_idx[0] and i != n_idx[1]]
            clusters.append(aglomerated)
            iteration += 1
        return clusters
```



Кластеризация дивизимным анализом
Divisive clustering

- Более сложный по сравнению с агломеративной кластеризацией метод
- Изначально все объекты содержатся в одном кластере
- Кластер(ы) некоторым образом разбиваются дихотомически
- С каждой итерацией количество кластеров увеличивается, процесс останавливается при достижении нужного их количества

Дивизимная кластеризация

Вход: множество объектов $\mathcal{X}$, параметр k количества кластеров
Выход: множество пар (объект-кластер)

- Все объекты из $\mathcal{X}$ отнести к одному классу из $\mathcal{C}$
- Пока количество кластеров меньше, чем k
  - Разбить кластер с наибольшим внутрикластерным расстоянием на два с помощью какого-либо алгоритма кластеризации
- Возвратить пары (объект из $\mathcal{X}$ – номер кластера, в который попал этот объект)

Источники

1. Воронцов, К.В. «Лекции по алгоритмам кластеризации и многомерного шкалирования»
2. Moore A. "K-means and Hierarchical Clustering"
3. Singh A. "Spectral Clustering"
4. Николенко С.И. «Алгоритмы кластеризации»
5. Ng A. "The k-means clustering algorithm"