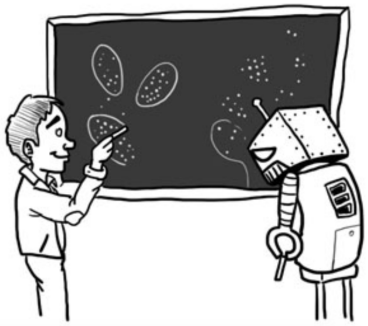
Содержание

1 K-means и EM-алгоритм

Николай Анохин: «Самая сложная лекция курса. Если в ней разобраться, то дальше легче будет!»

Сегодня мы попытаемся понять, что средства статистического моделирования и машинного обучения непрерывно связаны.



1.1 Задача кластеризации

На прошлой лекции мы разбирали такие понятия, как **обучение без учителя** и **обучение с учителем.** В первом случае значение целевой функции для объектов из обучающей выборки неизвестно. Поэтому решение таких задач подразумевает исследование «скрытой структуры» данных.

Сегодня мы с Вами рассмотрим **задачу класстеризации**.

Определение 1.1. *Задача кластеризации — задача без учителя, подразумевающая разбиение множества объектов на непересекающиеся подмножества (кластеры) так, чтобы каждый кластер состоял из схожих объектов, а объекты разных кластеров существенно отличались.*

Существенное отличие задачи кластеризации заключается в том, что нам дано **output space** (конечная обучающая выборка объектов), но нет целевой переменной.

Наша задача — загруппировать выборку на непересекающиеся подмножества, называемые кластерами, так, чтобы каждый кластер состоял из объектов, близких по метрике. Это означает, что всё, что у нас изначально есть — это *признаковое описание* наших объектов, и нам необходимо выбрать существенные признаки и разбить на группы, то есть кластеризовать, наши объекты.

Большинство алгоритмов подразумевают, что мы знаем, какое количество кластеров должны получить. Однако на практике чаще ставится задача определить оптимальное число кластеров, с точки зрения того или иного критерия качества кластеризации.

Зачем вообще так необходимо заниматься задачей кластеризации?

1. Кластеризация позволяет больше узнать о данных (knowledge discovery!);
2. Работать с кластерами удобнее, чем с отдельно взятыми объектами;
3. Кластеризация позволяет конструировать новые признаки.

1.1.1 Постановка задачи кластеризации

Задача 1.1. *Во-первых нам дана обучающая выборка из N объектов, которые мы обозначаем за $x \in \mathbf{X}$ (trainig data set). Необходимо найти модель из семейства параметрических функций вида*

$$H = \{h(x, \theta) : X \times \Theta \to Y | Y = \{1, \ldots, K\}\},$$

ставящую в соответствие произвольному $x \in \mathbf{X}$ один из K кластеров так, чтобы объекты внутри одного кластера были похожи, а объекты из разных кластеров различались. То есть наша цель выбрать такие параметры θ, которые наилучшим образом отображают действительность.

Выше представленная формулировка достаточно обобщённая, попробуем разобраться на более конкретном примере.

Задача 1.2. *Пусть у нас есть некоторый набор объектов с* **двумя свойствами**, *по которым можно построить гистограмму. Необходимо построить модель, описывающеую данное распределение. Считаем, что знаем, сколько получается "колоколов"на гистограме, но не знаем описываемые параметры наших объектов.*

Пункт 1 Для начала разберём простой случай $K = 1$.

Замечание 1.1. *На самом деле это у нас никакая не задача кластеризация, ибо нет никакой группировки. Но мы рассмотрим принцип максимального правдопадобия и научимся с его помощью находить параметры Гауссовского распределения.*

Задачу можно переформулировать следующим образом:

Задача 1.3. *У нас есть ружье, положение которого мы зафиксировали. Мы делаем какое-то количетво выстрелов и считаем, что разброс определяется Гауссовским распределением. Необходимо определить, куда на самом деле смотрит прицел (т.е. нам даны координаты точек попаданий по мишени из гауссовской пушки, а найти нужно куда смещен прицел).*

Замечание 1.2. *Фишер присвоил имя «Гауссовскому» распределению, но сам Гаусс использовал в своих работах с геодезическими нормальное распределение.*

Выпишем вид нормального (Гауссовского) распределения в матричной форме

$$\mathcal{N}(x|\mu,\mathbf{\Sigma}) = \frac{1}{(2\pi)^{\frac{D}{2}}\cdot|\mathbf{\Sigma}|^{\frac{1}{2}}}\cdot e^{-\frac{1}{2}(x-\mu)^T\mathbf{\Sigma}^{-1}(x-\mu)} \tag{1}$$

$$\mu = \int xp(x)dx, \qquad \mathbf{\Sigma} = E\left[(x-\mu)(x-\mu)^T\right],$$

где x — «столбик» размерности D, μ — ввектор средних (размерности D), $\mathbf{\Sigma}$ — матрица ковариаций ($D\times D$), а D — плотность распределения.

Замечание 1.3. $\mathbf{\Sigma}$ — *симметрическая положительно определенная матрица*

Формализуем нашу задачу

$$\mathbf{X} = \{x\in R^2\}, \qquad p(x)\backsim N\left(x|\mu,\mathbf{\Sigma}\right), \qquad \mu\in R^2, \qquad \mathbf{\Sigma}\in R^{2\times 2}.$$

Необходимо найти вектор средних μ и матрицу ковариации $\mathbf{\Sigma}$.

Замечание 1.4. *Ранее у нас была следующая формула*

Learning = Representation + Evaluation + Optimuzation

- *Representation – модель (то, как распределены наши данные). В нашем случае — Гауссовское расперделение (иногда бывает линейное)*
- *Evaluation — то, как мы можем проверить наши данные;*
- *Optimuzation — ?*

Пример: *Рассмотрим закон Ома $U = I\cdot R$. Изначально мы можем замерить силу тока и напряжение, то есть имеем U и I, а необходимо найти R. Таким образом*

- *Representation — много точек, построили линии;*
- *Evaluation — наобходимо понять, какая линия самая лучшая на графике (например методом наименьших квадратов);*
- *Optimuzation —?*

1.2 Принцип максимального правдоподобия

Пусть у нас есть некий набор данных и гипотиза, как понять — хорошая ли наша гипотиза или нет?

Берем гипотизу, вычисляем вероятность, с которой можно пронаблюдать те данные, которые мы видим. Так делаем для всех имеющихся у нас гипотиз (значений параметров). После этого выбраем ту, для которой эта вероятность будет максимальная. То есть та гипотиза, при которой мы имеем больше всего шансов пронаблюдать наши данные и есть самая лучшая.

Пример 1.1. *Допустим, мы сидим дома, смотрим в окно и видим, что все люди на улице идут с зонтами. В таком случае мы можем сделать собственно предположение, в чём же причина этого.*

- *На небе образовалась озоновая дыра, и люди прячуются от вредного излучения под своим зонтом.*
- *На улице идет дождь.*

Конечно, вероятность дождя гораздо выше. По этому по **принципу максимального правдоподобия** *на улице идёт дождь.*

Теперь давайте формализуем наши рассуждения.

Определение 1.2. *Функция правдоподобия Пусть дано семейство параметрических моделей $h(\mathbf{x},\theta)$, где θ — неизвестные параметры. Предположим, что $p(\mathbf{X},\theta)$ — функция распределения случайной велечены (вероятность), а все наблюдения исходноый выборке независимы и одинаково распределены, тогда* **функцией**

правдопдобия *(likelihood) будет называться их совместная вероятность (плотность), а именно*

$$l(\mathbf{X}|\theta) = \prod_{n=1}^{N} p(x, \theta),$$

но в нашем курсе функцией правдоподобия мы будем называть её аналог в виде логарифмической функции правдоподобия

$$L(\mathbf{X}|\theta) = \log l = \log\left(\prod_{n=1}^{N} p(x,\theta)\right) = \sum_{n=1}^{N} \log\left(f(x,\theta)\right).$$

Определение 1.3. *Принцип максимального правдоподобия (ML) — это метод оценивания неизвестного параметра путём максимизации функции правдоподобия, который основан на предположении о том, что вся информация о статистической выборке содержится в функции правдоподобия.*

1.2.1 Вычисления параметров гауссовской модели

Вернёмся к нашей задаче (1.3).

Решение.

Случайные велечены в нашем, в нашем случае — координаты точек попаданий по мишени, и все они независимы и одинаково распределены (распределение нормальное). В нашем случае вероятность принимает вид

$$P(x) = \prod_{n=1}^{N} p(\mathbf{x_n}|\mu, \mathbf{\Sigma}) \to \max_{\mu,\mathbf{\Sigma}}.$$

Прологорифмируем нашу полную вероятность, а вместо $p(x_n|\mu, \mathbf{\Sigma})$ подставим нормальное распределение

$$L(\mathbf{X}|\mu, \mathbf{\Sigma}) = \log(P(x)) = \sum_{n=1}^{N} \log(\mathcal{N}(\mathbf{x_n}|\mu, \mathbf{\Sigma}) \to \max_{\mu,\mathbf{\Sigma}}$$

Для начала честно выпишем, чему равно наше $L(\mathbf{X}|\mu, \mathbf{\Sigma})$ (см.ф-лу. 1).

$$L(\mathbf{X}|\mu, \mathbf{\Sigma}) = \sum_{n=1}^{N} \log(\mathcal{N}(\mathbf{x_n}|\mu, \mathbf{\Sigma}) = -\sum_{n=1}^{N}\left(\frac{D}{2}\log 2\pi + \frac{1}{2}\log|\mathbf{\Sigma}| + \frac{1}{2}(\mathbf{x_n}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x_n}-\mu)\right);$$

А теперь найдём максимум нашей функции. Для этого продифференциируем её по μ и приравняем к 0. Но перед этим вспомним несколько свойств дифференциирования матриц.

$$\frac{\partial}{\partial \mathbf{x}}\mathbf{x}^T A\mathbf{x} = (A + A^T)\cdot\mathbf{x}, \qquad \frac{\partial}{\partial A}|A| = |A|\cdot\left(A^{-1}\right)^T, \qquad \frac{\partial}{\partial A}\mathbf{x}^T A\mathbf{y} = \mathbf{x}\cdot\mathbf{y}^T, \qquad \frac{\partial}{\partial A}A\cdot B = B^T.$$

Поэтому,

$$\frac{\partial L}{\partial \mu} = \frac{\partial}{\partial \mu}\left(\frac{1}{2}(\mathbf{x_n}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x_n}-\mu)\right) = \dots$$

Так же вспомним про свойство симметричности матрицы $\mathbf{\Sigma}$

$$\dots = -\frac{1}{2}\sum_{n=1}^{N}\underbrace{\left(\mathbf{\Sigma}^{-1} + \left(\mathbf{\Sigma}^{-1}\right)^T\right)}_{2\cdot\mathbf{\Sigma}^{-1}}(\mathbf{x_n}-\mu) = -\mathbf{\Sigma}^{-1}\cdot\sum_{n=1}^{N}(\mathbf{x_n}-\mu) = \mathbf{\Sigma}^{-1}\left(N\cdot\mu - \sum_{n=1}^{N}\mathbf{x_n}\right) = 0;$$

Соответственно,

$$\sum_{n=1}^{N}\mathbf{x_n} = N\cdot\mu \quad \Longrightarrow \quad \mu_{ML} = \frac{1}{N}\sum_{n=1}^{N}x_n,$$

А теперь аналогично найдём $\mathbf{\Sigma}_{ML}$, только искать максимум L будем через параметр $\mathbf{\Sigma}^{-1}$, то есть

$$\frac{\partial L}{\partial \mathbf{\Sigma}^{-1}} = \frac{1}{2}\left(\sum_{n=1}^{N}\frac{\partial}{\partial \mathbf{\Sigma}^{-1}}\left(\log|\mathbf{\Sigma}|\right) + \sum_{n=1}^{N}\frac{\partial}{\partial \mathbf{\Sigma}^{-1}}\left((\mathbf{x_n}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x_n}-\mu)\right)\right) = \dots$$

Несложно показать, что в нашем случае

$$(\mathbf{x_n}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x_n}-\mu) = tr\left(\mathbf{\Sigma}^{-1}(\mathbf{x_n}-\mu)(\mathbf{x_n}-\mu)^T\right),$$

Тогда

$$\cdots = \frac{1}{2}\left(N \cdot \frac{\partial}{\partial \mathbf{\Sigma^{-1}}}\left(\log|\mathbf{\Sigma^{-1}}|\right) + \sum_{n=1}^{N}\frac{\partial}{\partial \mathbf{\Sigma^{-1}}} tr\left(\mathbf{\Sigma}^{-1}(\mathbf{x_n}-\mu)(\mathbf{x_n}-\mu)^T\right)\right) =$$

$$= \frac{1}{2}\left(N \cdot \frac{|\mathbf{\Sigma^{-1}}| \cdot \left(\left(\mathbf{\Sigma^{-1}}\right)^{-1}\right)^T}{|\mathbf{\Sigma^{-1}}|} + \sum_{n=1}^{N}\left((\mathbf{x_n}-\mu)(\mathbf{x_n}-\mu)^T\right)^T\right) =$$

$$= \frac{1}{2}\left(N \cdot \mathbf{\Sigma} + \sum_{n=1}^{N}\left((\mathbf{x_n}-\mu)^T\right)^T \cdot (\mathbf{x_n}-\mu)^T\right) = \frac{1}{2}\left(N \cdot \mathbf{\Sigma} + \sum_{n=1}^{N}(\mathbf{x_n}-\mu)(\mathbf{x_n}-\mu)^T\right) = 0$$

$$\mathbf{\Sigma_{ML}} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x_n}-\mu)(\mathbf{x_n}-\mu)^T$$

■

1.2.2 Сложности в работе с принципом правдоподобия

1. схлопование компонент;
2. переименование кластеров;
3. невозможно оптимизировать.

1.2.3 Нахождения параметров смеси нормальных распределений

Пункт 2. А теперь давайте рассмотрим случай, когда наши данные исходят не из одного, а из дувух нормальных распределений ($K = 2$).

Небольшое предисловие. В америке есть национальный парк, который называется *Yellow Stone*, в котором «типа красиво». Там происходят извержения гейзеров, но все проблема в том, что они не регулярны, и поэтому пронаблюдать их сложно. Один из самых известных — *Old Faithful*. Давайте построим модель, по которой можно будет предсказать время извережения и его длительность.

Была собрана статисктика по извержению гейзеров по следующим параметрам: время извержения и разница во времени между извержениями. Допустим, мы хотим взять и построить модель на основе наших данных. Наш набор данных двухмерный и очень удобный, мы можем посмтреть, как наша модель работает. Так же у него есть очевидная структура и мы можем сказать, что если мы сделаем всё правильно, то мы что-нибудь да увидим, а именно 2 кластера (как показано на рис. ()).

Как нам к этой задаче приступить?

Рассмотрим суперпозицию 2-ух гауссовских распредений (только мы заведомо не знаем, к которому из них какие точки относятся, т.е. они смогут попасть, как в один так и в другой). То есть у нас имеется двумерная случайная велечина $\mathbf{z} = (z_1, z_2) \in \mathbb{R}^2$, которая обозначает принадлежность объекта к одному из кластеров, то есть может принимать только два вида: $\mathbf{z} = (0, 1)$ или $\mathbf{z} = (1, 0)$. Принадлежность точки к первому или второму кластеру обозначение следующим образом

$$p(z_1 = 1) = \pi_1, \qquad p(z_2 = 1) = \pi_2,$$

где $p(z)$ — априорнок распределение на всём векторе $\mathbf{z}$ (распределение 0 и 1). Несложно заметить, что

$$\sum_{k=1}^{K} z_k = 1, \qquad \sum_{k=1}^{K}\pi_k = 1, \qquad p(z) = \prod_{k=1}^{K}\pi_k^{z_k} \tag{2}$$

(так же предполагаем, что все π_k ненулевые).

Распределение $\mathbf{X}$ для каждого из K кластеров будем считать нормальным (1) (каждому кластеру соответствует свой вектор средних и своя матрица ковариации), тогда для $\forall x \in \mathbf{X}$

$$p(x|z_k) = N(x|\mu_k, \mathbf{\Sigma}_k) \quad \Longrightarrow \quad p(x|\mathbf{z}) = \prod_{k=1}^{K}\left(\mathcal{N}(x|\mu_k, \mathbf{\Sigma}_k)\right)^{z_k}$$

Будем опускать потихоньку параметры модели и найдём распределение $\mathbf{X}$

$$p(x) = p(x|\pi, \mu, \mathbf{\Sigma}) = \underbrace{p(x|\mathbf{z}) \cdot p(\mathbf{z})}_{\text{по формуле полной вероятности}} = \sum_{k=1}^{K}\pi_k\mathcal{N}(x|\mu_k, \mathbf{\Sigma}_k).$$

На самом деле это то, что мы и ожидали увидеть, а именно смесь Гауссовских распределений, которые соответствуют априорной вероятности появления x в неком кластере.

Чтобы нам было проще жить, введём следующую велечину

$$\gamma(z_k) = p(z_k = 1|x) = \underbrace{\frac{p(z_k = 1) \cdot p(x|z_k = 1)}{p(x)}}_{\text{по формуле Байеса}} = \frac{\pi_k \mathcal{N}(x|\mu_k, \mathbf{\Sigma}_k)}{\sum\limits_{j=1}^{K} \pi_j \mathcal{N}(x|\mu_j, \mathbf{\Sigma}_j)}$$

Раньше, до того, как мы начали иметь дело с нашими данными, у нас было единственное предположение (2) — априорное распределение. А γ — апостериорное распределение, то есть вероятность того, что мы попали в кластер под номером k, если пронаблюдали x.

Теперь мы уже готовы применять принцип максимального правдоподобия. Поэтому выпишем функцию максимального правдоподобия и сразу же возьмём логорифм.

$$p(\mathbf{X}|\pi, \mu, \mathbf{\Sigma}) = \underbrace{\prod_{n=1}^{N} p(x_n|\pi, \mu, \mathbf{\Sigma})}_{\text{произведение превратится в сумму}} \qquad \Longrightarrow \qquad L(\mathbf{X}|\pi, \mu, \mathbf{\Sigma}) = \log\left(p(\mathbf{X}|\pi, \mu, \mathbf{\Sigma})\right) =$$

$$= \sum_{n=1}^{K} \log \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \mathbf{\Sigma}_k) \quad \rightarrow \quad max(\pi, \mu, \mathbf{\Sigma}).$$

Далее продифференциируем функцию правдоподобия по каждому из параметров, возьмём точки, где функция обращается в ноль, исследуем минимум это или максимум и всё здорово. Есть только одна проблема: эту функцию вполне можно продифференциировать, но найти решение полученных уравнений аналитически будет невозможно, в отличии от одномерного случая.

Для начала продифференциируем по μ

$$\frac{\partial L}{\partial \mu_k} = \sum_{n=1}^{N} \underbrace{\frac{\pi_k \mathcal{N}(x_n|\mu_k, \mathbf{\Sigma}_k)}{\sum\limits_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \mathbf{\Sigma}_j)}}_{\gamma_{z_{nk}}} \cdot \mathbf{\Sigma}^{-1}(x_n - \mu_k) = \sum_{n=1}^{N} \gamma_{z_{nk}}(x_n - \mu_k) = 0;$$

$$\mu_k = \frac{\sum\limits_{n=1}^{N} \gamma(z_{nk}) x_n}{\sum\limits_{n=1}^{N} \gamma(z_{nk})} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) x_n,$$

где N_k — эффективное количество элементов в кластере. Аналагично получим

$$\mathbf{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(x_n - \mu_k)^T (x_n - \mu_k), \qquad \pi_k = \frac{N_k}{N}.$$

Наконец, кажется всё здорово и задачу мы вроде бы решили, но есть одна проблема: γ зависит от всего, то есть от μ, $\mathbf{\Sigma}$ и K. Нам удалось получить какие-то выражения, но они не представляют из себя решения, ибо вычислить мы ничего не можем. Однако вычислить мы всё-таки что-то хотим, для этого воспользуемся EM-алгоритмом.

1.3 EM-алгоритм

EM-алгоритм (Expectation Maximization) — это, то что сегодня является центром нашей лекции. Идейно алгоритм очень простой. На каждой итерации последовательно выполняется два шага.

На первом шаге (**E**) мы высчитываем наши γ, то есть считаем, что мы как бы знаем $\mu, \mathbf{\Sigma}, \pi$, и расскидываем наши объекты по кластерам.

На втором шаге(**M**) мы используем вычисленные γ и на их основании мы высчитываем новые значения параметров.

Хорошее тут то, что процедура гарантированно сходится. Можно доказать, что на каждом из шагов likelihood никогда не убывает: он может оставаться таким же или возрастать. Рано или поздно мы упрёмся в экстремум, то есть локальный максимум.

Замечание 1.5. *В нашем случае очень важно «локальный», ибо EM не гарантирует глобальной максимизации. То есть, когда мы что-то нашли мы не можем гарантировать, что наше решение оптимально глобально (только локально). Однако с этим можно бороться.*

Иллюстрацию работы алгоритма можно посмотреть на рис. (1)

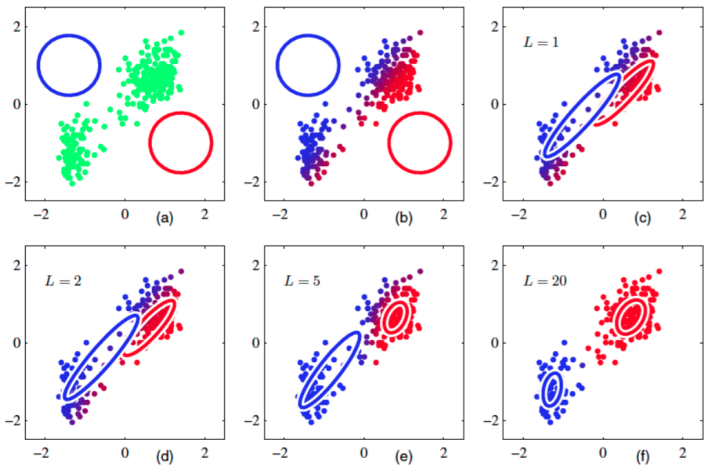


Рис. 1. Пример работы EM алгоритма.

Оказывается, что алгоритм EM можно обобщить на произвольных случай неизвестных переменных, На шаге E мы вычисляем распределение латентных (скрытых) переменных $p\left(\mathbf{z}|\mathbf{X},\theta^{old}\right)$, при учёте, что нам даны «старые» параметры модели (начинаем со случайных). Этому распределению соответствуют наши γ, которые мы уже ранее обсуждали.

На шаге M мы высчитываем $\theta^{new} = \arg\max_{\theta} Q(\theta,\theta^{old})$, через математическое ожидание по $\mathbf{z}$ логарифма совместного распредления $\mathbf{x}$ и $\mathbf{z}$.

$$Q(\theta,\theta^{old}) = E_{\mathbf{z}}\left[\ln p(\mathbf{X},\mathbf{z}|\mu,\mathbf{\Sigma},\pi)\right] = \sum_{\mathbf{z}} p(\mathbf{z}|\mathbf{X},\mu,\mathbf{\Sigma},\pi)\ln p(\mathbf{X},\mathbf{z}|\mu,\mathbf{\Sigma},\pi)$$

Как раньше это всё происходит до сходимости.

Так жу в данном случае есть способ улучшить наш алгоритм, введя априорное распределение $p(\theta)$ с помощи формулы Байеса.

1.4 K-means

Алгоритм представляет собой версию EM-алгоритма, применяемого также для разделения смеси гауссиан. Он разбивает множество элементов векторного пространства на заранее известное число кластеров k.

Основная **идея** алгоритма заключается в том, что на каждой итерации перевычисляется центр масс для каждого кластера, полученного на предыдущем шаге, затем векторы разбиваются на кластеры вновь в соответствии с тем, какой из новых центров оказался ближе по выбранной метрике.

Алгоритм завершается, когда на какой-то итерации не происходит изменения центра масс кластеров. Это происходит за конечное число итераций, так как количество возможных разбиений конечного множества конечно, а на каждом шаге суммарное квадратичное отклонение не увеличивается, поэтому зацикливание невозможно.

А теперь давайте немного формализуем наши рассуждения.

Пусть матрицы ковариации все одинаковые, а ещё и круглые, т.е. $\mathbf{\Sigma}_k = \varepsilon \cdot Id$. $Det(\mathbf{\Sigma}_k) = \varepsilon$, тогда

$$p(x|\mu_k,\mathbf{\Sigma_k}) = \mathcal{N}(x|\mu_k,\mathbf{\Sigma}_k) = \frac{1}{\sqrt{2\pi\varepsilon}} e^{-\frac{1}{2\varepsilon}||x-\mu_k||^2}$$

Рассмотрим предельный переход $\varepsilon \to 0$

$$\gamma_{nk} = \frac{\pi_k \mathcal{N}(x|\mu_k,\mathbf{\Sigma}_k)}{\sum\limits_{j=1}^{K}\pi_j\mathcal{N}(x|\mu_j,\mathbf{\Sigma}_j)} = \frac{\pi_k e^{-\frac{1}{2\varepsilon}||x_n-\mu_k||^2}}{\sum\limits_{j=1}^{K}\pi_j e^{-\frac{1}{2\varepsilon}||x_n-\mu_j||^2}} \quad \to \quad r_{nk} = \begin{cases} 1, & \text{для } k = \arg\min\limits_j ||x_n-\mu_k||; \\ 0, & \text{иначе.} \end{cases}$$

Таким образом γ — это просто функции вхождения n-го элемента в k-й кластер. Она будет стремиться к 1 для ближайшего кластера, для остальных к 0. Функция правдоподобия принимает вид

$$E_{\mathbf{z}}\left[\ln p(\mathbf{X},\mathbf{z}|\mu,\mathbf{\Sigma},\pi)\right] \to -\sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}||x_n-\mu_k||^2 + \text{const},$$

$$\mu_k = \frac{\sum\limits_{n=1}^{N} r_{nk} x_n}{\sum\limits_{n=1}^{N} r_{nk}}.$$

Замечание 1.6. *Не сложно заметить, что в знаменателе последней формулы стоит просто количество элементов в кластере.*

Таким образом, используя идею EM-алгоритма, мы перешли к одному из замых известных алгоритмов кластеризации k-means. Получается, между вероятностной моделью и моделью машинного обучения есть связь: алгоритм k-means по сути базируется на модели смеси гауссовских распределений.

1.4.1 Алгоритм работы программы

Рассмотрим алгоритм в виде псевдокода на языке *Python*:

```
function kmeans(X, K):
    initialize N # number of objects
    initialize Mu = (mu_1 ... mu_K) # random centroids
    do:
    # E step
        for k in 1..K:
            for x in 1..N:
                compute r_nk # Cluster assignment
    # M step
        for k in 1..K:
            recompute mu_k # Update centroids
    until Mu converged
    J = loss(X, Mu)
    return Mu, J
```

Несложно заметить, что сложность данного алгоритма равна $O(NK)$. Так же на рисунке (2) представлен пример реализции данного метода.

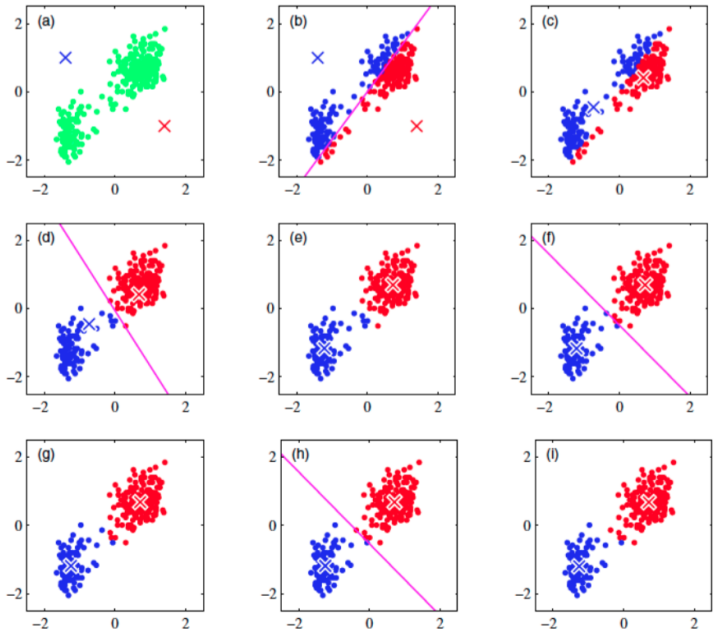


Рис. 2. Пример работы алгоритма k-means.

1.4.2 Проблемы

1. Оптимизация, как и в случае с EM, является локальной.
   Для решения этой проблемы есть такая традиция запускать k-means много раз, случайным образом выбирая начальные центры кластеров. В качестве лучшего выбираем тот, где значение функции потерь J минимальна. Практика показывает, что количество перезапуска этого алгоритма должна начинаться примерно от 1000 раз.

2. Данных слишком много.
   Решение: Если не можем загрузить все данные, то можно выбирать какое-то подмножество и на его

основе построить центроиды. После этого разбить всё множество на соответствующие кластеры с уже имеющимися центрами. (данная модификация алгоритма называется *mini-batch k-means*)

3. Признаки не числовые
   Решение: x = {набор различных признаков объекта}
   В качестве центройдов – не целое значение, а какого-то типичного представителя и с ним сравнивать. Функция сравнения — $d(x_i, x_j)$ — расстояние до центра (или ещё какого-то параметра).

1.4.3 Критерий качества

$$\overline{J} = \sum_{n=1}^{N} \sum_{k=1}^{k} r_{nk} d(x_n, \mu_k),$$

где $d(x_n, \mu_k)$ — функция расстояния, μ_k — один из объектов кластера.

Таким образом, качество работы алгоритма зависит от выбранной функции растояния между объектами.

Замечание 1.7. *Если с помощью нашей метрики мы не можем найти центроид кластера, то выбирается один из обучающих объектов, который лежит далеко от границы кластера, в виде типичного представитель. Такой представитель называется medoid, а алгоритм соответственно будет иметь новое название $k - medoids$.*

1.4.4 Оптимизация (k-means++)

(Из Википедии)
Данный алгоритм оптимизирует способ нахождения начальных центроид наших кластеров.

1. Выбрать первый центроид случайным образом (среди всех точек)

2. Для каждой точки найти значение квадрата расстояния до ближайшего центроида (из тех, которые уже выбраны) dx^2

3. Выбрать из этих точек следующий центроид так, чтобы вероятность выбора точки была пропорциональна вычисленному для неё квадрату расстояния
   Это можно сделать следующим образом. На шаге 2 нужно параллельно с расчётом dx^2 подсчитывать сумму $Sum(dx^2)$. После накопления суммы найти значение $Rnd = random(0.0, 1.0) \cdot Sum$. Rnd случайным образом укажет на число из интервала $[0; Sum)$, и нам остаётся только определить, какой точке это соответствует. Для этого нужно снова начать подсчитывать сумму $S(dx^2)$ до тех пор, пока сумма не превысит Rnd. Как только это случится, суммирование останавливается, и мы можем взять текущую точку в качестве центроида.
   При выборе каждого следующего центроида специально следить за тем, чтобы он не совпал с одной из уже выбранных в качестве центроидов точек, не нужно, так как вероятность повторного выбора некоторой точки равна 0.

4. Повторять шаги 2 и 3 до тех пор, пока не будут найдены все необходимые центроиды. Далее выполняется основной алгоритм k-means.

2 Алгоритмы кластеризации

2.1 Виды расстояний

2.1.1 Между объектами

Для начала давайте разберёмся, как лучше всего выбирать расстояние между объектами?

Вариантов выбора способа задания расстояния огромное множество, но мы будем рассматривать только некоторые, наиболее популярные, из них.

1. Минковского
$$d_r(\mathbf{x}, \mathbf{y}) = \left(\sum_{j=1}^{N} |x_j - y_j|^r \right)^{\frac{1}{r}}$$
2. Евклидово ($r = 2$)
$$d_E(\mathbf{x}, \mathbf{y}) = d_2(\mathbf{x}, \mathbf{y})$$
3. Манхеттен ($r = 1$) — расстояние, которое надо пройти пешеходу.
$$d_M(\mathbf{x}, \mathbf{y}) = d_1(\mathbf{x}, \mathbf{y})$$
4. $r = \infty$
$$d_\infty(\mathbf{x}, \mathbf{y}) = \max_j |x_j - y_j|$$
5. Жаккар
$$d_J(\mathbf{x}, \mathbf{y}) = 1 - \frac{|x_j \cap y_j|}{|x_j \cup y_j|}$$
6. Косинус
$$d_C(\mathbf{x}, \mathbf{y}) = \arccos \frac{\mathbf{xy}}{||\mathbf{x}|| \cdot ||\mathbf{y}||}$$
7. Правки
d_e — наименьшее количество удалений и вставок, приводящее $\mathbf{x}$ к $\mathbf{y}$;
8. Хэмминг
d_H — количество различных компонент в $\mathbf{x}$ и $\mathbf{y}$.

2.1.2 Между кластерами

А теперь предположим, что мы уже разбили наши объекты, на какое-то количество кластеров и хотим определить, на каком расстоянии кластеры расположены друг относительно друга. Для этого используются следующие способы задания расстояния:

1. Single-linkage
$$d_{\min}(C_i, C_j) = \min_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} ||\mathbf{x} - \mathbf{y}||$$
2. Complete-linkage
$$d_{\max}(C_i, C_j) = \max_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} ||\mathbf{x} - \mathbf{y}||$$
3. Average
$$d_{avg}(C_i, C_j) = \frac{1}{n_j n_i} \sum_{\mathbf{x} \in C_i} \sum_{\mathbf{y} \in C_j} ||\mathbf{x} - \mathbf{y}||$$
4. Mean
$$d_{mean}(C_i, C_j) = ||\mathbf{m_i} - \mathbf{m_j}||$$

2.2 Иерархическая кластеризация

2.2.1 Идея метода

Мы все из биологии помним, что всё живое на земле делится на следующую иерархию: царство, порядок, семейство, род, вид и так далее. Каждая из ступенек этой иерархии является непересекающейся группой, которую мы и будем называть кластером.

Суть иерархической кластеризации заключается в том, чтобы взять и разделить весь наш *input space* (входное пространство) на части, но только не плоско, а, грубо говоря, вложено. То есть снача мы делим на несколько частей, каждая из которых делится ещё и ещё до тех пор, пока результат нас не устроит.

Есть 2 достаточно очевидных, но принципиально разных способа реализации данного алгоритма — *Agglomerative* (восходящие) и *Divisive* (нисходящие).

Визуализация данных, как правило, осуществляется с помощью **дендрограммы** (дерево, то есть граф без циклов).

2.2.2 Agglomerative

Наиболее подходит для случая, когда нам нужно получить большое количество кластеров. Так же данный вариант обычно немножко проще реализуется.

1. начинаем с ситуации, когда каждый объект – отдельный кластер;

2. на каждом шаге совмещаем два наиболее близких кластера, получая из них один новый;

3. останавливаемся, когда получаем требуемое количество или единственный кластер.

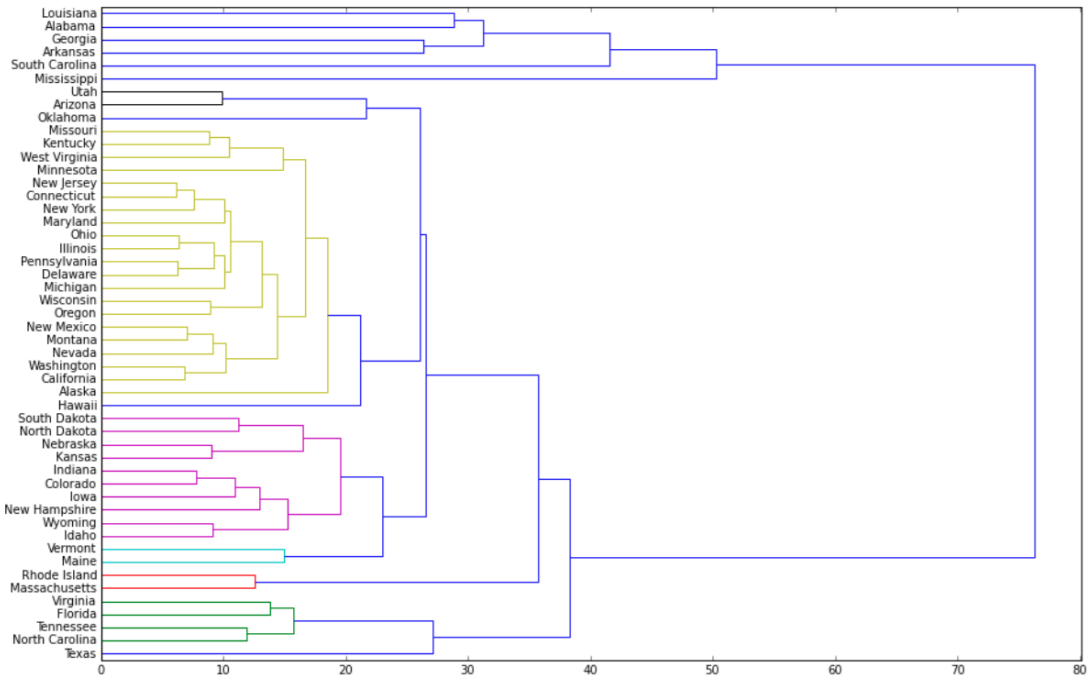


Рис. 3. Пример дендограммы восходящей кластеризации на основе красно-синих штатов.

2.2.3 Divisive

Более удобен, когда нам нужно несколько кластеров.

1. начинаем с ситуации, когда все объекты составляют один большой кластер;

2. на каждом шаге разделяем каждый кластер две или несколько частей;

3. останавливаемся, когда получаем требуемое количество или заранее заданное N кластеров.

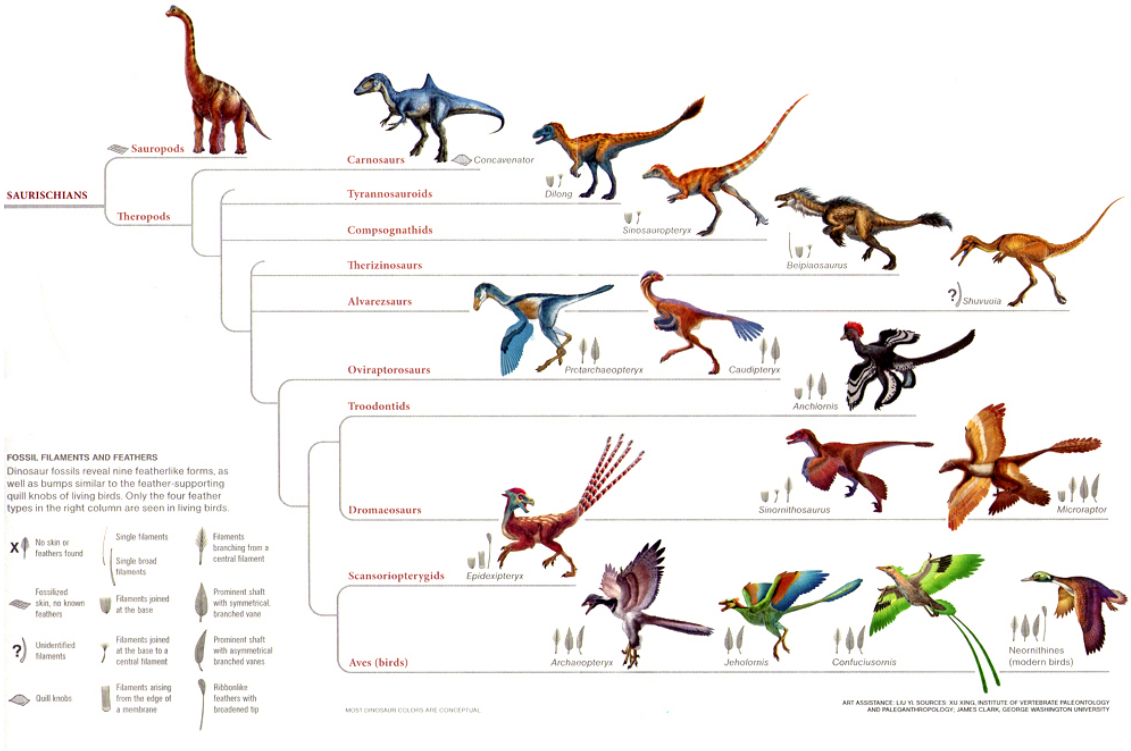


Рис. 4. Пример дендограммы нисходящей кластеризации на основе классификации динозавров.

2.2.4 Реализация алгоритма

Рассмотрим восходящий алгоритм в виде псевдокода на языке *Python*:

```
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
```

Как мы видим, необходимая память $O(N^2)$, а вычислительная сложность данного алгоритма $(N^2 \log N)$ (хорошо, что не куб). Получается, что этот алгоритм вычислительно очень дорогой и применять его на очень больших объемах данных очень непродуктивно, и никакой памяти не хватит. Но у него есть несколько приемуществ: удобная визуализация и не сложно выбрать количество кластеров.

2.2.5 Stepwise-optimal HC

Что плохого в том алгоритме, что мы описали выше?

На каждом шаге мы пытаемся совместить два кластера, но не очень понимаем почему и зачем. Понятно, что минимальное расстояние между класстерами – это хорошо. Придумали сделать некое обобщение этого алгоритма, который позволяет делать всё тоже самое для произвольного критерия качества (J).

Окаывается, что те расстояния между кластерами, которые мы использовали, имеют под собой функционал, который они оптимизируют. В частности d_{max} обеспечивает наименьшее увеличение диаметра кластера. Так же часто используется расстояние d_e (см. 3), которое обеспечивает наименьшее увеличение квадратичного критерия (квадратичный критерий — это сумма расстояний внутри кластера от каждого объекта, до соответствующего центра).

$$d_e(C_i, C_j) = \sqrt{\frac{n_i n_j}{n_i + n_j}} ||m_i - m_j|| \tag{3}$$

Рассмотрим данный алгоритм в виде псевдокода на языке *Python*:

```
function swo(X, K):
    initialize N # number of objects
    initialize C = N # number of clusters initialize
    C_i = x_i # initial clusters
    while C > K:
        # choose the pair that optimizes
        # the given criterion J when merged
        C_a, C_b = find_best_merge(J, C_1, ..., C_C)
        merge(C_a, C_b)
        C=C - 1
    return C_1, ... , C_k
```

2.2.6 Неевклидовы пространства

Как измерить расстояние между кластерами, если невозможно определить центроид?

Оказывается, что иерархический подход так же работает, но только нужно изменить функцию растояния между кластерами.

Идея. В каждом из кластеров выбрать «типичныи представителя» "—- clustroid. Считать расстояние не между кластерами, а между «кластроидами»

Каким образом выбрать «типичного представителя» ?

Есть несколько подходов. Можно выбать такой объект, для которого

- минимальная сумма расстояний до других объектов в кластере (он находится в центре);
- сумма квадратов расстояний до других объектов в кластере минимальна;
- максимальное расстояние до других объектов в кластере минимально.

Таким образом можно кластеризовать что-то другое помимо числовых векторов.

2.2.7 Плюсы и минусы алгоритма

\+ Могут получиться несферические кластеры;

\+ Можно придумать разнообразие критерии, как разнообразные виды расстояния между кластерами, так и между объектами;

\+ Поддерживает любые K из коробки (то есть один раз запустив алгоритм, мы уже кластеризовали наши объекты на любое количество кластеров);

\- Требует много вычислительных ресурсов, невозможно работать при больших объёмов данных.

2.3 «Быстрая» модификация алгоритма

```
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
```

Осталось только понять, что же такое этот delta-set и откуда его брать.

Delta-set — набор кластеров расстояние между которыми меньше δ. Чтобы реализовать функцию **get-delta-set** нужно учитывать следующие моменты:

1. Если $C \leqslant K_1$, то δ-set – это все C_i;
2. Иначе выбрать K_2 случайных расстояний между кластерами, $\delta = \min K_1, K_2$;
3. K_1 и K_2 влияют только на скорость, но не на результат кластеризации;
4. рекомендованные значения $K_1 = K_2 = 20$.

2.4 DBSCAN

DBSCAN — *«Density Based Spatial Clustering of Applications with Noise»* (плотностный алгоритм для кластеризации пространственных данных с присутствием шума) впервые был предложен Мартином Эстер, Гансом-Питером Кригель и их коллегами в 1996 году как решение проблемы разбиения (изначально пространственных) данных на кластеры произвольной формы.

Основная **идея** алгоритма заключается в том, что внутри каждого кластера наблюдается *типичная плотность точек* (объектов), которая заметно выше, чем плотность вне этого кластера. Также есть области *шума*, плотность которых згачительно меньше плотности любого из кластеров. Дабы отделить шум от основных класстеров задаётся некоторое значение, соответствующее минимальному пороговому значению точек в кластере, а именно *Min_ Pts*

А теперь для удобства рассуждений введём несколько определений.

Определение 2.1. *Плотность — количество объектов внутри сферы заданного радиуса ε.*

Определение 2.2. *Core-объект — объект, плотность вокруг котрого больше, чем Min_ Pts.*

Определение 2.3. *Граничный объект — объект, плотность вокруг котрого меньше, чем Min_ Pts, однако он находится в непосредственной близости с Core-объектом.*

Определение 2.4. *Шум — объект, который не является ни core-объектом, ни граничным объектом.*

Определение 2.5. *Кластеры — участки высокой плотности, состоящие из core-объектов и граничных-объектов, разделенные участками низкой плотности.*

Таким образом, для реализации алгоритма нам понадобиться два параметра: ε и *Min_ Pts.*

2.4.1 Алгоритм

На вход подаём множество объектов — X, ε-радиус «соседства» и минимальное количество объектов в кластере Min_Pts. Рассмотрим алгоритм в виде псевдокода на языке *Python*:

```
def 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)
    return C

def 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)
```

2.4.2 Плюсы и минусы

+ не требуется заранее знать количество кластеров;
+ кластеры могут быть произвольной формы. К тому же может случиться такое, что один кластер будет полностью лежать в другом, но они не будут соприкасаться;
+ в данном алгоритме присутствует понятие шума, поэтому он устойчив к выбросам;
+ для работы алгоритмы достаточно всего два параметра.
− не является вполне детерминированным, так как если граничные точки равноудалены сразу от двух кластеров, то они могут быть быть интерпретированными по разному в зависимости от порядка обхода множества объектов;
− не работает при большой разности в плотностях кластеров, потому что нельзя подобрать комбинацию ε и *Min_ Pts* соответствующим образом сразу для всех кластеров;

– если нет чёткого понимания природы данных, с которыми предстоит работать, то бывает трудно подобрать параметры ε и Min_Pts.

2.4.3 Вычислительная сложность

В общем случае алгоритм DBSCAN имеет квадратичную вычислительную сложность (n^2) за счёт поиска ε-соседства. Однако, если для этой цели использовать специальную структуру данных — $R*Tree$, то в результате сложность поиска ε-соседей для одной точки – $O(\log n)$. Таким образом общая вычислительная сложность алгоритма DBSCAN составляет $O(n \log n)$.

2.4.4 Пример

Зафиксируем параметры $\varepsilon = \frac{\sqrt{13}}{2} + \delta$ (расстояние между точками B1 и B2 + некое маленькое $\delta > 0$) и $Min\_Pts = 3$. В случае (5) в качестве стартовой точки выберем B1, а в случае (6) — B2. Несложно заметить, что результаты полчатся разные. В первом случае B2 — граничная точка для кластера B, а во втором — шум.

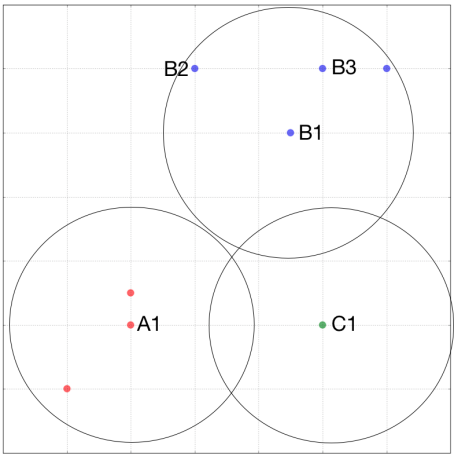


Рис. 5. Пример кластеризации методом DBSCAN с начальной точкой B1.

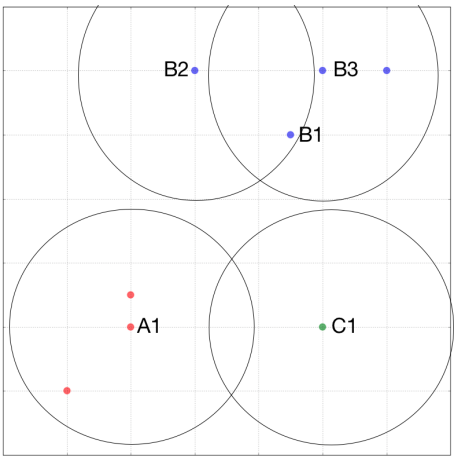


Рис. 6. Пример кластеризации методом DBSCAN с начальной точкой B2.

2.5 Оценка качества алгоритма кластеризации

Вот предположим, что у нас есть какие-то данные, есть готовая кластеризация, но нам этого мало. Мы сели и ручками написали свой алгоритм кластеризации. Как можно оценить качество нашего алгоритма?

Пусть нам заведомо дана обучающая выборка, для которой правильная кластеризация C известна. С помощью выбранного алгоритма получена кластеризация K и проверим, насколько K совпадает с C. Для этого используем такие понятия, как *Adjusted Rand Index* (ARI) и *Mutual Information* (MI)

$$RI = \frac{a+b}{C_2^N}, \qquad ARI = \frac{RI - E_{rdm}[RI]}{\max(RI) - E_{rdm}[RI]},$$

где a — количество пар объектов, попавших в один кластер и в C, и в K,
а b — кол-во пар объектов, попавших в разные кластеры и в C, и в K.

$$MI = \sum_{c \in C} \sum_{c \in K} p(c,k) \log \frac{p(c,k)}{p(k)p(c)}.$$

А что делать, если изначально «правильной» кластеризации попросту нет в наличии?! И как выбирать параметры нашей для нашей кластеризации?!

2.5.1 Критерий Silhouette («силуэт»)

Пусть дана кластеризация N объектов в K кластеров, и в каждом кластере C_k находтся N_k объектов. Предположим, что наш объект i попал в C_k, тогда

- a(i) — среднее расстояние от i объекта до объектов из его же кластера C_k:

$$a(i) = \frac{1}{N_k - 1} \sum_{j \in C_k} d_k(i,j),$$

  где $d_k(i,j)$ — расстояние между i и j объектами;

- $b(i) = \min\limits_{j \neq k} b_j(i)$, где $b_j(i)$ — минимальное среднее расстояние от i объекта до объектов из другого кластера C_j (то есть расстояние до соседнего кластера, куда бы он мог попасть, если бы не в этот кластер).

Таким образом

$$silhouette(i) = S_i = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

Заметим, что если наша кластеризация хорошая, то $b(i)$ всегда будет больше, чем $a(i)$, и $\max(a(i), b(i)) = b(i)$. А оценкой качества нашего алгоритма будет являться средний *silhouette* для всех точек из множества **X.** То есть

$$SWC = \frac{1}{N} \sum_{j=1}^{N} S_j.$$

То значение K, для которого SWC будет максимально, выбирается в качестве оптимального количества кластеров.

Замечание 2.1. *Если K окажется равным* 1*, то* $SWC = -1$.

2.6 OPTICS

OPTICS — *«ordering points to identify the clustering structure».*

Основная идея — не ограничиваться фиксированной плотностью, как в DBSCAN, а варьировать ее в зависимости от того, как много объектов попадают в ε-окрестность.

Как в алгоритме DBSCAN, OPTICS требует двух параметров: ε, который описывает максимальное расстояние (радиус) для оценки плотности, и $MinPts$ — та самая минимальная плотность, то есть количество объектов, необходимых для формирования кластера. Однакое, в отличии от DBSCAN, OPTICS также учитывает то, что точки могут являться частью более плотного кластера, поэтому вводится понятие $core\_distance$, которое описывает расстояние до ближайшей точки внутри круга радиуса $MinPts$.

$$core\_distance(p) = \begin{cases} \text{UNDEFINED}, & \text{if } |N_\varepsilon(p)| < MinPts; \\ MinPts \text{ is the smallest distance to } N_\varepsilon(p), & \text{otherwise.} \end{cases}$$

Так же введём понятие $reachability\_distance$, описывающее расстояние от объекта O до p или $core\_distance(p)$:

$$reachability\_distance(p) = \begin{cases} \text{UNDEFINED}, & \text{if } |N_\varepsilon(p)| < MinPts; \\ \max(core\_dist(p), dist(p,O)), & \text{otherwise.} \end{cases}$$

Если окажется так, что O и P — ближайшие друг к другу объекты, то необходимо, что бы в конечном итоге они оказались в одном кластере.

Оба «новых» параметра будут неопределены, если в районе данной точки не будет располагаться достаточно плотный кластер, удовлетворяющий основным параметрам $(\varepsilon, MinPts)$.

2.6.1 Алгоритм OPTICS

Рассмотрим данный алгоритм в виде псевдокода на языке *Python*:

```
function optics(X, eps, min_pts)
    for each point x in X:
        x.reachability-distance = UNDEFINED
    for each unprocessed point x in X:
        N = getNeighbors(x, eps)
        mark x as processed
        output x to the ordered list
        if (core-distance(x, eps, min_pts) != UNDEFINED):
            Seeds = empty priority queue
            update(N, x, Seeds, eps, min_pts)
            for each next z in Seeds:
                N1 = getNeighbors(z, eps)
                mark z as processed
                output z to the ordered list
                if (core-distance(z, eps, min_pts) != UNDEFINED):
                    update(N1, z, Seeds, eps, min_pts)

function update(N, x, Seeds, eps, min_pts)
    coredist = core-distance(x, eps, min_pts)
    for each z in N:
        if (z is not processed):
            new-reach-dist = max(coredist, dist(x, z))
            # z is not in Seeds
            if (z.reachability-distance == UNDEFINED):
                z.reachability-distance = new-reach-dist
                Seeds.insert(o, new-reach-dist)
            # z in Seeds, check for improvement
            else:
                if (new-reach-dist < z.reachability-distance):
                    z.reachability-distance = new-reach-dist
                    Seeds.move-up(z, new-reach-dist)
```

2.7 BIRCH

BIRCH — *Balanced Iterative Reducing and Clustering using Hierarchies.*

Идея метода: построить иерархию кластеров, которая позволит хранить ограниченное количество данных в виде агрегатов.

- локальность: каждая точка «кластеризуется» без сканирования всех других точек или имеющихся кластеров;
- выбросы: точки в «густонаселенных» регионах принадлежат кластерам, а в «малонаселенных» — к выбросам;
- экономность: используется вся доступная память, при этом минимизируется I/O;
- масштабируемость: при определенных условиях обучается «онлайн» и требует единственного прохода по данным.

2.7.1 Меры компактности кластера

- Центроид

$$x_0 = \frac{1}{N}\sum_{i=1}^{N} x_i;$$

- Радиус

$$R = \frac{1}{N}\sum_{i=1}^{N} d(x_i, x_0);$$

- Диаметр

$$D = \frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j=1}^{N} d(x_i, x_j).$$

В начале работы алгоритма все объекты принадлежат одному кластеру, который на последующих шагах делится на меньшие кластеры, в результате образуется последовательность расщепляющих групп.

В этом алгоритме предусмотрен двухэтапный процесс кластеризации.

2.7.2 Clustering Feature

Clustering feature — это объект, содержащий сжатую информацию о кластере.

Определение 2.6. *Пусть кластер C содержит N d-мерных объектов x_i. Clustering feature (CF) для C определяется как тройка $CF = (N, LS, SS)$, где*

$$LS = \sum_{i=1}^{N} x_i, \qquad SS = \sum_{i=1}^{N} x_i^2.$$

Утверждение 2.1. *Пусть $CF_1 = (N_1, LS_1, SS_1)$ и $CF_2 = (N_2, LS_2, SS_2)$ — CF для кластеров C_1 и C_2. Тогда CF для кластера, полученного слиянием C_1 и C_2, определяется как*

$$CF = (N_1 + N_2, LS_1 + LS_2, SS_1 + SS_2).$$

2.7.3 CF-Tree

Определение 2.7. *CF-Tree — это взвешенно сбалансированное дерево, состоящее из множества кластерных элементов (clustering feature).*
Балансирующие параметры:

- *B – коэффициент разветвление (максимальное количество детей у внутреннего узла);*
- *L – максимальное количество детей у листа;*
- *T – пороговая велечина, а именно максимальная компактность (R или D) ребенка листа(т.е. лист не может быть больше, чем это T).*

Каждый нелистъевой узел данного дерева имеет не более чем B вхождений узлов следующей формы: $[CF_i, Child_i]$, где $i = 1, 2, \ldots, B$ ($Child_i$ – указатель на i-й дочерний узел).
Каждый листъевой узел имеет ссылку на два соседних узла. Кластер состоящий из элементов листъевого узла должен удовлетворять следующему условию: диаметр или радиус полученного кластера должен быть не более пороговой величины T.

Замечание 2.2. *При разделении узла выбираем две наиболее удаленные CF и лепим к ним ближайшие.*

2.7.4 Итоги

- Назначение: кластеризация очень больших наборов числовых данных.
- Ограничения: работа с только числовыми данными.
- Достоинства:
  - \+ двухступенчатая кластеризация,
  - \+ кластеризация больших объемов данных,
  - \+ работает на ограниченном объеме памяти,
  - \+ является локальным алгоритмом,
  - \+ может работать при одном сканировании входного набора данных,
  - \+ использует тот факт, что данные неодинаково распределены по пространству,
  - \+ обрабатывает области с большой плотностью как единый кластер.
- Недостатки:
  - – работа с только числовыми данными,
  - – хорошо выделяет только кластеры сферической формы,
  - – есть необходимость в задании пороговых значений.