Кластеризация

Пусть дана тренировочная выборка X = {x_1, x_2, …, x_n}, где x_i = {f_i1, f_i2, f_i3, …, f_im} --- признаковые описания объектов.

Надо разбить эти объекты на кластера так, чтобы внутри одного кластера объекты были похожи, а для разных --- как можно больше различались.

Оценка качества кластеризации

- Логично минимизировать среднее внутрикластерное расстояние, а максимизировать --- среднее межкластерное.
- В итоге будем стремиться минимизировать отношение среднего внутрикластерного расстояния к среднему межкластерному.

Иерархическая кластеризация Bottom-up

- Пусть есть точки x_1, x_2, …, x_n, их нужно кластеризовать.
- Считаем каждую точку кластером.
- Пока не получим нужное количество кластеров, объединяем ближайшие кластера в один.
- Получаем дерево объединений.

Иерархическая кластеризация Bottom-up

- Def HierarchyClustering(X):

  clusters, tree = X[:, :], X[:, :]

  Выбираем два элемента c_1, c_2 из clusters, расстояние между которыми минимально.

  Добавляем в tree вершину c_12, соединяем ее с вершинами c_1, c_2

  clusters = clusters + c_12 – c_1 – c_2

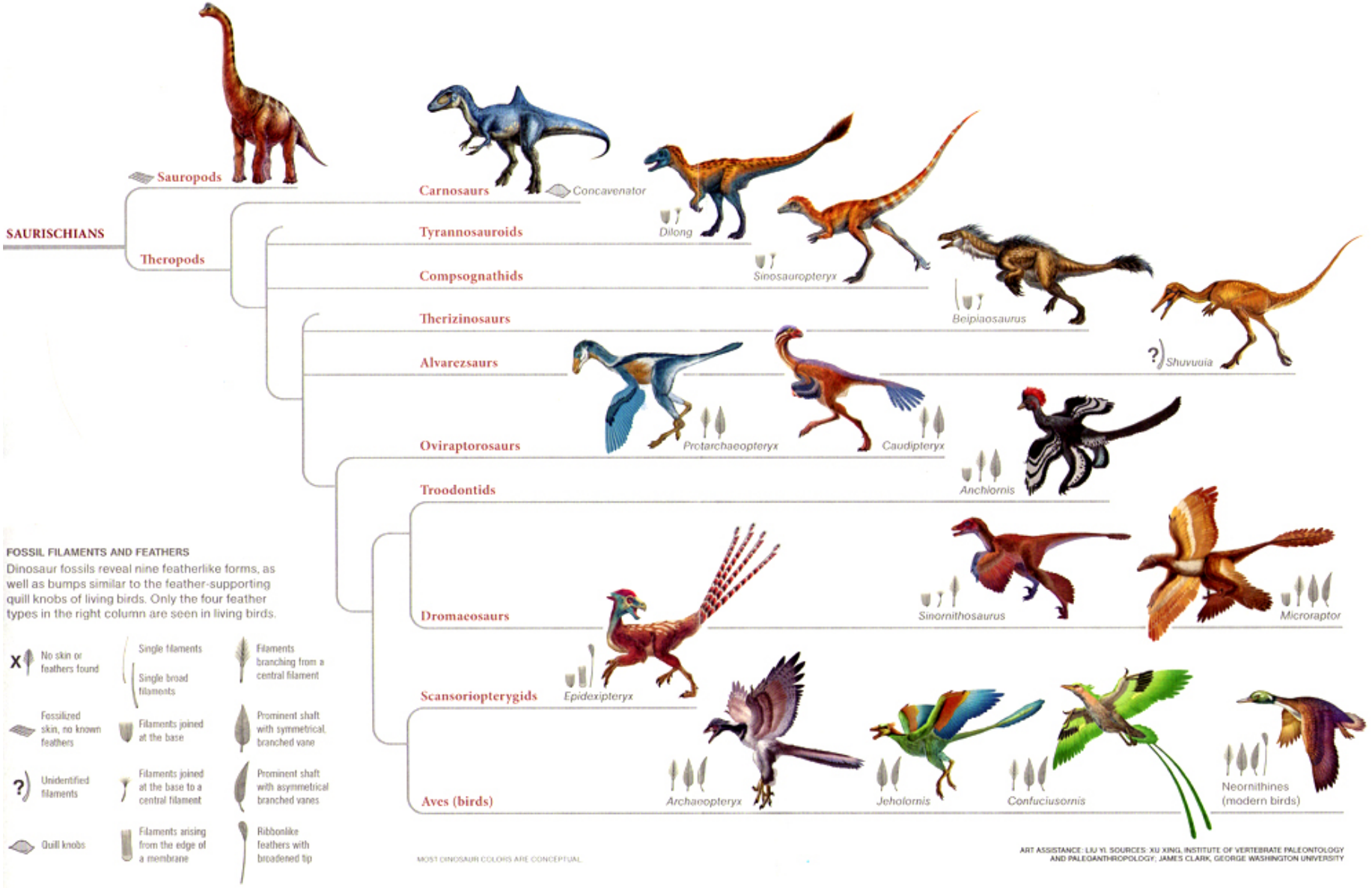
Иерархическая кластеризация Top-down

- Вся выборка -- один кластер (корень дерева)
- На каждом шаге необходимо разбить лист (который тоже выбиратеся по какому-то критерию) на несколько по критерию разбиения
- Выбрать критерии остановки
- Получаем дерево разбиений

Критерии выбора листа

- Разбивать лист, наиболее близкий к корню
- Разбивать лист с наибольшим числом элементов

Иерархическая кластеризация



Расстояния

- Понятно, как вычислять расстояние между объектами.
- Между кластерами существует много способов вычислять расстояния.

Расстояния 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

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

mean

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

Иерархическая кластеризация

+ Не нужно заранее задавать количество кластеров

+ Получаем иерархическую структуру между кластерами

KMeans

- Пусть мы хотим разбить данные на k кластеров.
- Идея: мы хотим минимизировать сумму квадратов отклонений точек в одном кластере от их центра масс.
- То есть фактически мы двигаем центры масс, а точки рапределяем атоматически, смотря к какому ближе центру масс лежит точка.

KMeans

- Def Kmeans(X):

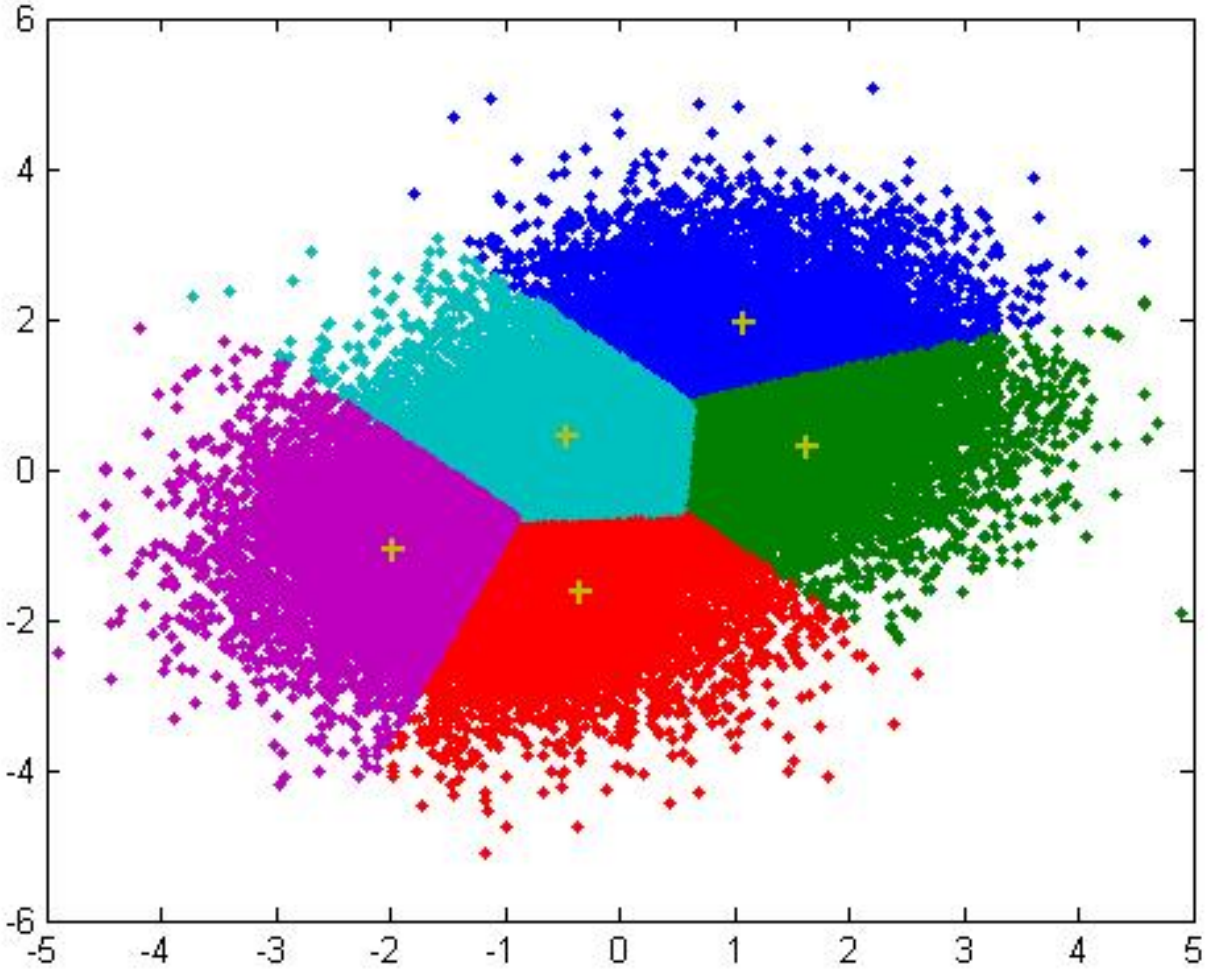
  Инициализировать k центров масс (можно рандомно)

  Распределить точки по кластерам, относительно этих центров масс

  Взять центры масс новых кластеров

  Проделывать до тех пор, пока центры масс изменяются/остановка по количеству итераций

KMeans



KMEANS

- + Для новых точек можно определять, к какому кластеру она принадлежит
- – k заранее неизвестно
- – Положения центров кластеров зависит от начального приближения. Как решить эту проблему?

Mini-batch KMEANS

- Если выборка очень большая, то стандартный алгоритм будет работать долго
- Идея: выбираем рандомную подвыборку из нашей выборки на каждом шаге алгоритма и обучаться на ней

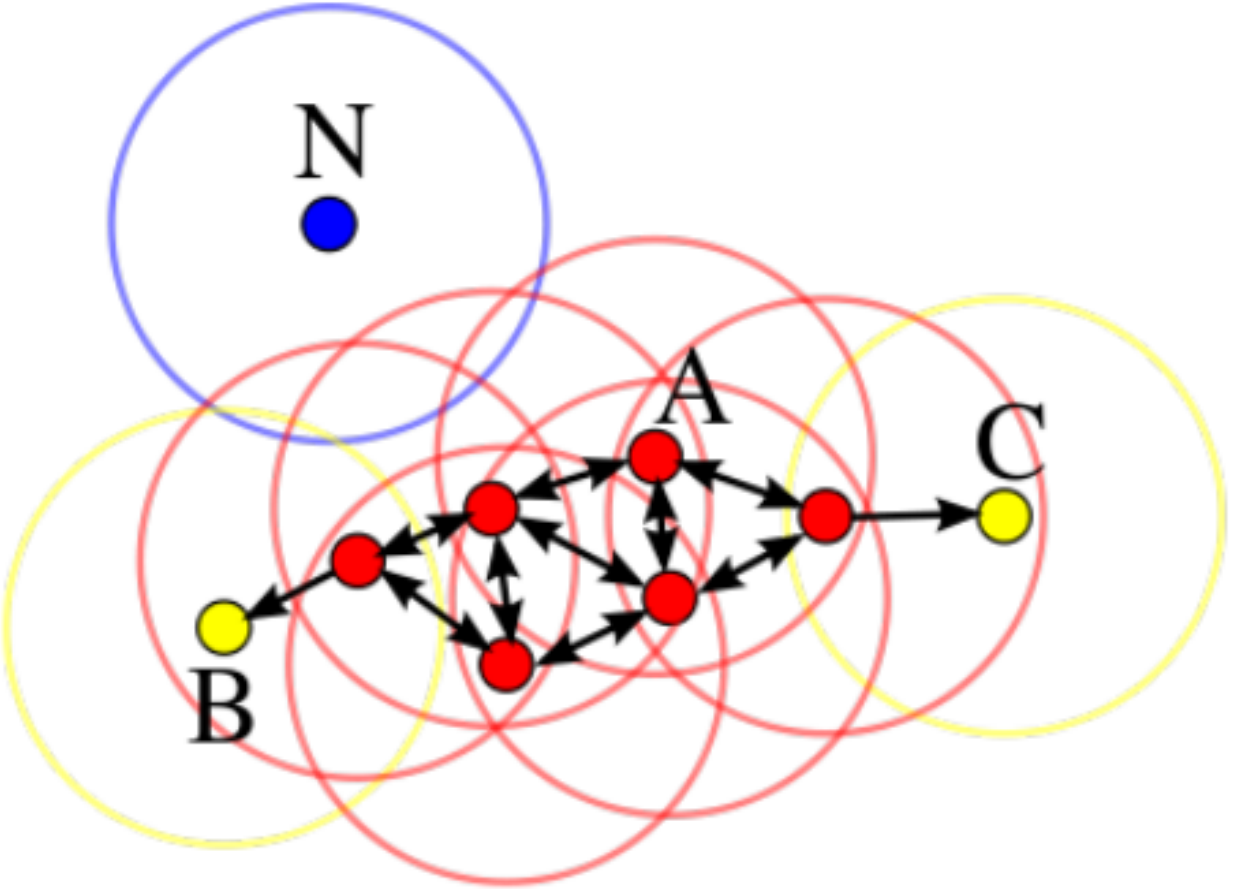
DBSCAN

- Идея: будем кластеризовать объекты, учитывая плотность объектов.
- Кластеры --- участки высокой плотности, разделенные участками низкой плоности.

DBSCAN

- **Плотность** --- количество объектов внутри сферы (в двумерной случае --- окружности) заданного радиуса ε
- **Core-объект** --- такой объект, что плотность вокруг него больше min_pts
- **Граничный объект** --- плотность вокруг него меньше min_pts, но он находися рядом с core-объектом
- **Шум** --- не является ни core-объектом, ни граничным объектом

DBSCAN



DBSCAN

```
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)
            yield C
```

DBSCAN

```
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)
```

DBSCAN

+ не требует K

+ кластеры произвольной формы

+ учитывает выбросы

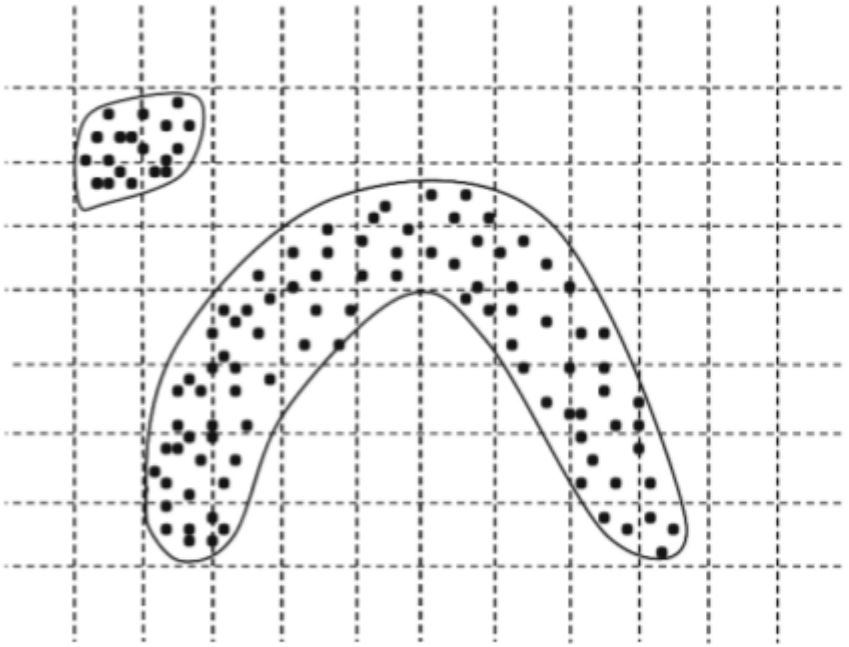
— Не вполне детерминированный

— Не работает при разных плотностях кластеров

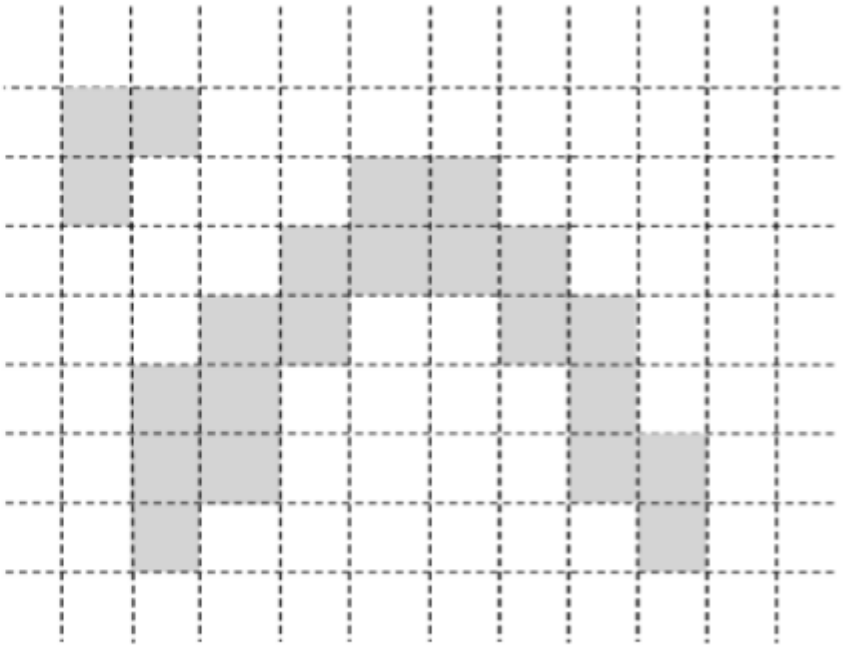
Grid-based

- Разобьем пространство на n-мерные кубики
- Если в кубик попало больше, чем k точек, “закрашиваем его”, объявляем вершиной графа
- Между вершинами проводится ребро, если они смежны по r измерениям (из d)
- Определяем связные компоненты получившегося графа

Grid-based



(a) Data points and grid



(b) Agglomerating adjacent grids

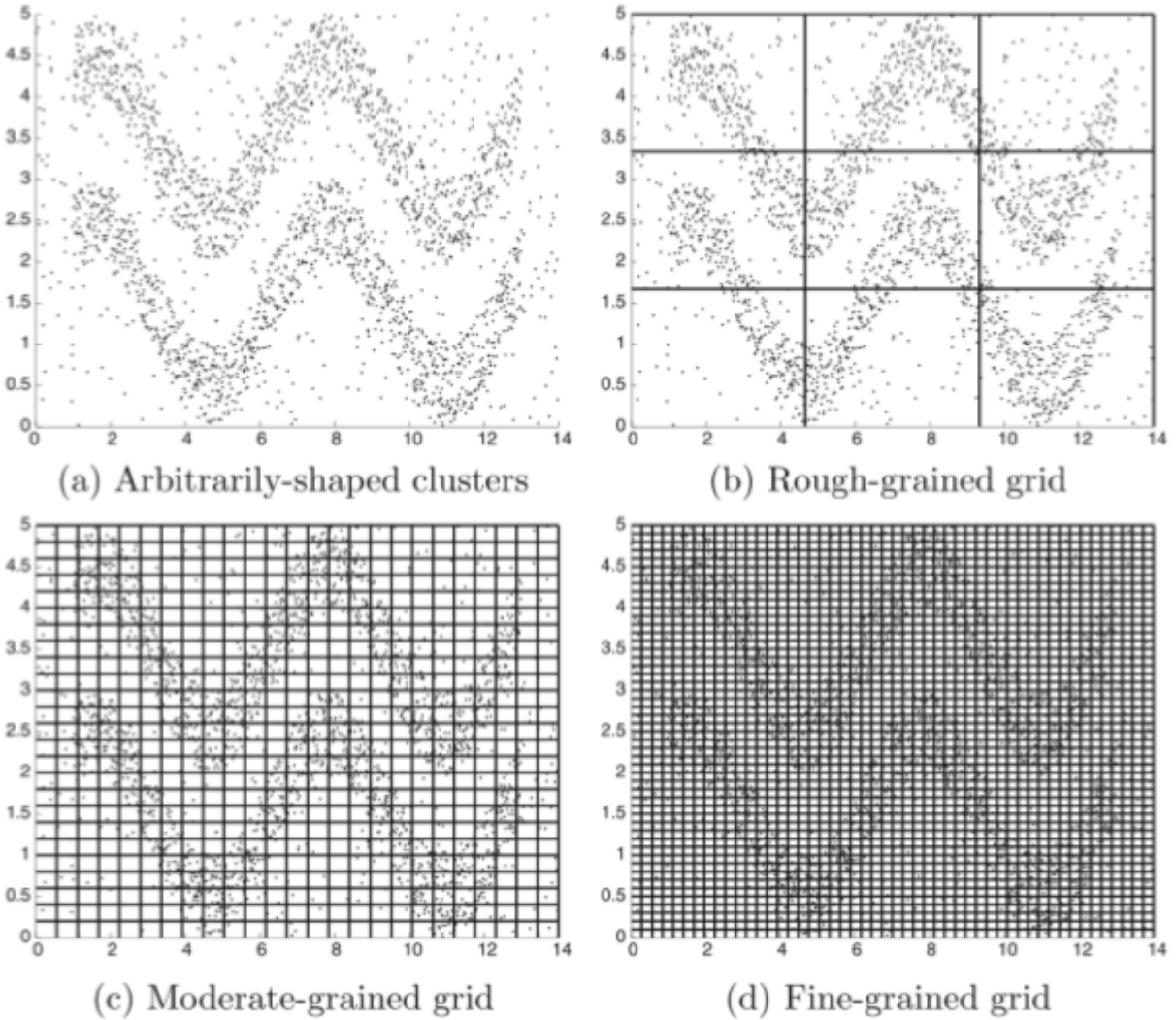
Grid-based

+ Колчество кластеров определяется автоматически

- Необходимо определить сторону куба, минимальное количество точек в кубе, r

- Качество кластеризации зависит от размера сетки

Grid-based


(a) Arbitrarily-shaped clusters
(b) Rough-grained grid
(c) Moderate-grained grid
(d) Fine-grained grid

Оценка качества кластеризации Силуэт (Silhouette)

- Для каждого объекта x_i:
- Пусть s_i --- среднее расстояние от x_i до других объектов в том же кластере, которому принадлежит x_i
- d_i --- среднее расстояние от x_i до других объектов в ближайшем к текущему кластеру
- Sil_i = (d_i – s_i) / max(d_i, s_i)
- Sil = среднее арифметическое по всем S_i

Работа различных алгоритмов