САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ

Алгоритмы кластеризации

Занятие 7

Глазунова Е.В.

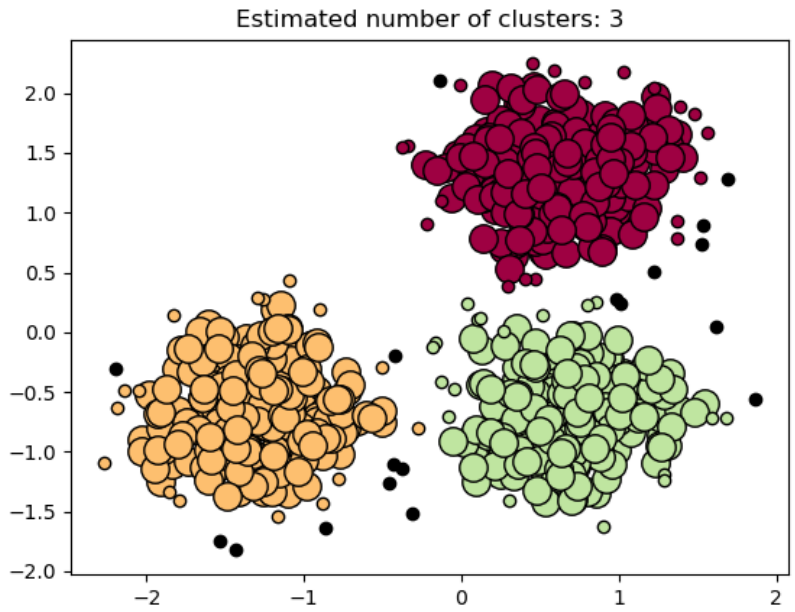
Постановка задачи

Наша цель - сгруппировать объекты между собой.

- Объекты, принадлежащие одному кластеру, похожи
- Объекты, принадлежащие разным кластерам, различны

Таргет:

- может не быть - задача обучения без учителя (unsupervized learning)
- может быть мало размеченных объектов (и много неразмеченных) - частичное обучение (semisupervized learning).



Постановка задачи

Зачем:

- Упростить обработку данных. Работать с каждой группой в отдельности
- Удалить дубликаты
- Выделить нетипичные объекты
- Построить иерархию вложенности объектов (котиков объединить по породам, все породы объединить в больший кластер)

Сложности

- Нет точной постановки задачи
- Много критериев качества кластеризации (все не очень)
- Число кластеров может быть неизвестно
- Много эвристик
- Должны ли точки одного кластера быть близко друг к другу? А что если нет? Какая вообще пространственная структура у данных?
- Разные методы кластеризации накладывают разные предположения на пространственную структуру, и выделяет кластеры лишь своих подтипов
- “Сложные” гиперпараметры типа метрики близости между точками могут давать кардинально разные результаты

Сложности

Среднее внутрикластерное расстояние:

$$F_0 = \frac{\sum_{i<j} [a_i = a_j] \rho(x_i, x_j)}{\sum_{i<j} [a_i = a_j]} \to min$$

Среднее межкластерное расстояние:

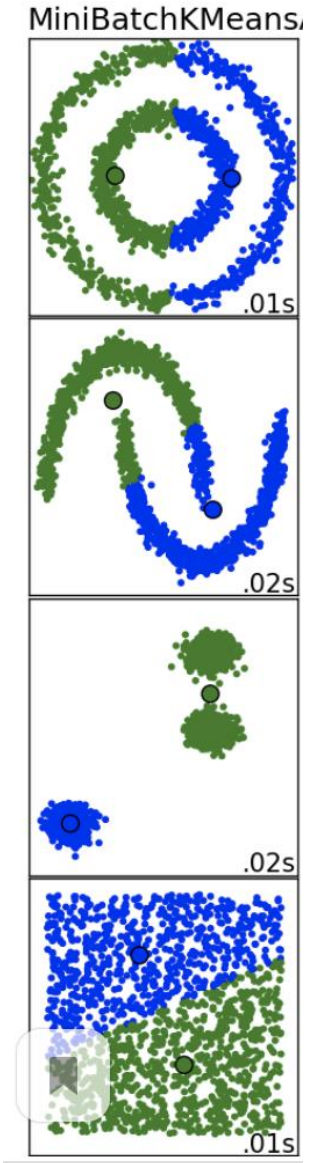
$$F_1 = \frac{\sum_{i<j} [a_i \neq a_j] \rho(x_i, x_j)}{\sum_{i<j} [a_i \neq a_j]} \to max$$

$$\frac{F_0}{F_1} \to min$$

Требуется только расстояние между точками

K-means

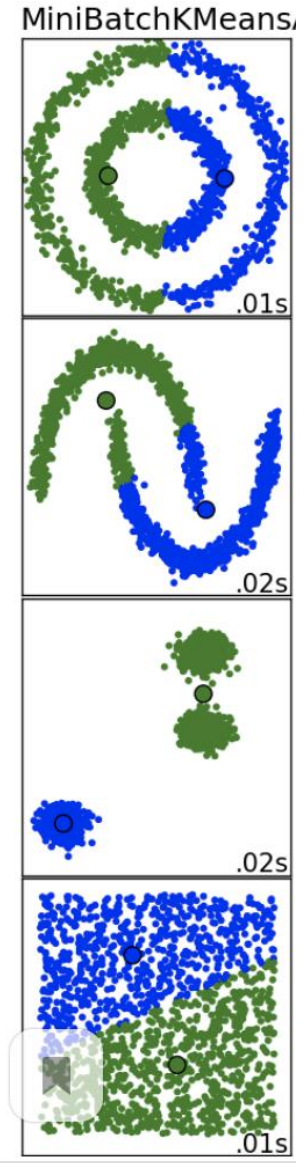
Алгоритм:
Шаг 1: Выбираем число кластеров, k
Шаг 2: Выбираем k случайных значений (точек, центроидов)
Шаг 3: Создаем k кластеров:
Относим каждую точку к тому кластеру, к центру которого она ближе
Шаг 4: Вычисляем новый центроид каждого кластера («средняя координата»)
Шаг 5: Если алгоритм сошелся, то конец, иначе шаг 3.



K-means

Проблемы:

- Не гарантируется достижение глобального минимума суммарного квадратичного отклонения, а только одного из локальных минимумов.
- Результат зависит от выбора исходных центров кластеров, их оптимальный выбор неизвестен.
- Число кластеров надо знать заранее.



DBSCAN

Density-Based Spatial Clustering of Applications with Noize

Объект x_i , его ϵ -окрестность $U_\epsilon = u \in U : \rho(x,u) \leq \epsilon$
Три типа объектов:

- Корневой: плотная окрестность $\|U_\epsilon(x)\| \geq m$
- Граничный: не корневой, но в окрестности корневого
- Шумовой: выброс, не корневой и не граничный

Предполагает, что плотность всех кластеров по всему пространству одинакова
Расширение: Generalized DBSCAN - подбирает ϵ, m автоматически

DBSCAN

```
DBSCAN(DB, distFunc, eps, minPts) {
   C=0                                                  /* Счётчик кластеров */
   for each point P in database DB {
      if label(P) ≠ undefined then continue               /* Точка была просмотрена во внутреннем цикле */
      Neighbors N=RangeQuery(DB, distFunc, P, eps)      /* Находим соседей */
      if|N|< minPts then {                              /* Проверка плотности */
         label(P)=Noise                                 /* Помечаем как шум */
         continue
      }
      C=C + 1                                           /* следующая метка кластера */
      label(P)=C                                        /* Помечаем начальную точку */
      Seed set S=N \ {P}                                /* Соседи для расширения */
      for each point Q in S {                             /* Обрабатываем каждую зачаточную точку */
         if label(Q)=Noise then label(Q)=C            /* Заменяем метку Шум на Край */
         if label(Q) ≠ undefined then continue            /* Была просмотрена */
         label(Q)=C                                     /* Помечаем соседа */
         Neighbors N=RangeQuery(DB, distFunc, Q, eps)   /* Находим соседей */
         if|N|≥ minPts then {                           /* Проверяем плотность */
            S=S U N                                     /* Добавляем соседей в набор зачаточных точек */
         }
      }
   }
}
```

DBSCAN

```
RangeQuery(DB, distFunc, Q, ϵ) {
    Neighbors=empty list
    for each point P in database DB {                    /* Scan all points in the database */
        if distFunc(Q,P) ⩽ ϵ then {                      /* Compute distance and check epsilon */
            Neighbors=Neighbors U {P}                    /* Add to result */
        }
    }
    return Neighbors
}
```