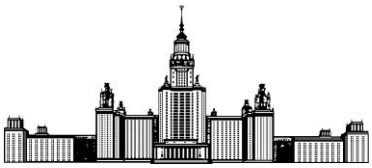
Московский государственный университет имени М. В. Ломоносова



Факультет Вычислительной Математики и Кибернетики

Методы кластеризации для больших данных

Реферат

Выполнил
студент 208 группы
Григорьев Илья Андреевич

Москва, 2020

Содержание

Введение **2**

1 Формальная постановка задачи кластеризации **4**

2 Статистические алгоритмы кластеризации **7**

2.1 Алгоритм K-Means 8

2.2 Алгоритм Mini Batch K-Means 13

3 Алгоритмы кластеризации, основанные на плотности **14**

3.1 Алгоритм кластеризации DBSCAN 14

Вывод **21**

Список литературы **22**

Введение

В современном мире информационное пространство оперирует колоссальным количеством данных и информации, которые настолько велики и сложны, что ни один из традиционных инструментов управления данными не может их хранить и эффективно обрабатывать. Для работы с таким количеством информации вводится понятие больших данных.

Большие данные – обозначение структурированных и неструктурированных данных огромных объёмов и значительного многообразия, эффективно обрабатываемых горизонтально масштабируемыми программными инструментами. В широком смысле о «больших данных» говорят как о социально-экономическом феномене, связанном с появлением технологических возможностей анализировать огромные массивы данных, где количество объектов может быть порядка нескольких миллионов, десятков миллионов и более. В качестве определяющих характеристик для больших данных традиционно выделяют «три V»: объём (величины физического объёма), скорость (скорость прироста, так и необходимости высокоскоростной обработки и получения результатов), многообразие (возможность одновременной обработки различных типов структурированных и неструктурированных данных).

Во многих прикладных задачах измерять степень сходства объектов существенно проще, чем формировать признаковые описания. [1] Задача классификации объектов на основе их сходства друг с другом, когда принадлежность обучающих объектов каким-либо классам не задаётся, называется задачей кластеризации. Задача кластеризации относится к статистической обработке, а также к широкому классу задач обучения без учителя. Спектр применений кластерного анализа очень широк. Вот его основные задачи:

- Понять структуру множества объектов, разбив его на группы схожих объектов. Упростить дальнейшую обработку данных и принятия решений, работая с каждым кластером по отдельности.

- Сократить объём хранимых данных в случае сверхбольшой выборки, оставив по одному наиболее типичному представителю от каждого кластера.

• Выделить нетипичные объекты, которые не подходят ни к одному из кластеров. Эту задачу называют одноклассовой классификацией, обнаружением нетипичности или новизны.

В первом случае число кластеров стараются сделать как можно меньше. Во втором случае важнее обеспечить высокую степень сходства объектов внутри каждого кластера, а кластеров может быть любое количество. В третьем случае наибольший интерес представляют отдельные объекты, не вписывающиеся ни в один из кластеров.

Кластеризация применяется в различных прикладных задачах, например при решении задачи сегментации целевой аудитории компании, для того чтобы в разных группах использовать разные маркетинговые стратегии или предлагать разные услуги. Или если требуется найти скрытые неочевидные закономерности и зависимости в данных. Так же кластеризация очень активно применяется в социологии.

Существует много различных типов алгоритмов кластеризации, например статистические, иерархические, графовые. Они, в свою очередь, делятся на непересекающиеся (каждому объекту выборки ставят в соответствие номер кластера, то есть каждый объект принадлежит только одному кластеру) и пересекающиеся (каждому объекту ставят в соответствие набор вещественных значений, показывающих степень отношения объекта к кластерам, то есть каждый объект относится к каждому кластеру с некоторой вероятностью). Но далеко не все алгоритмы могут работать с большими данными из-за быстрого роста вычислительной сложности с ростом количества объектов. В настоящем реферате будут рассмотрены процедуры кластеризации, основанные на статистическом и плотностном подходе, способные эффективно работать с большими данными.

1 Формальная постановка задачи кластеризации

Задача кластеризации [1] (или обучения без учителя) заключается в следующем. Имеется обучающая выборка $X^l = \{x_1, \ldots, x_l\} \subset X$ и функция расстояния между объектами ρ(x, x′). Требуется разбить выборку на непересекающиеся подмножества, называемые кластерами так, чтобы каждый кластер состоял из объектов, близких по метрике ρ, а объекты из разных кластеров существенно отличались. При этом, каждому объекту $x_i \in X^l$ приписывается метка (номер) кластера y_i.

Алгоритм кластеризации – это функция α: X → Y, которая каждому объекту $x \in X$ ставит в соответствие определённый кластер $y \in Y$. Множество меток Y в некоторых случаях известно заранее, однако как правило ставится задача определить оптимальное число кластеров, с точки зрения того или иного критерия качества кластеризации.

Решение задачи кластеризации принципиально неоднозначно, и тому есть несколько причин. Во-первых, не существует однозначно наилучшего критерия качества кластеризации. Существует целый ряд достаточно разумных критериев, а также ряд алгоритмов, не имеющих чётко выраженного критерия, но осуществляющих достаточно хорошую кластеризацию с геометрической точки зрения. Все они могут давать различные результаты. Во-вторых, число кластеров, как правило, неизвестно заранее и выбирается исходя из некоторого субъективного критерия. В-третьих, результат кластеризации существенно зависит от метрики ρ, выбор которой, как правило, определяется экспертом.

Чтобы вычислять расстояния между элементами выборки, нужно составить вектор характеристик для каждого объекта – как правило, это набор числовых значений. Однако существуют также алгоритмы, работающие с качественными (категориальными) признаками. Затем можно находить расстояния между любой парой объектов с помощью метрик. Далее представлены лишь основные из них.

1. Евклидово расстояние. Наиболее распространенная функция расстояния. Представляет собой геометрическое расстояние в многомерном пространстве.

   $\rho(x, x') = \sqrt{\sum_i^n (x_i - x_i')^2}$

2. Квадрат евклидова расстояния. Применяется для придания большего веса более отдаленным друг от друга объектам.

   $\rho(x, x') = \sum_i^n (x_i - x_i')^2$

3. Расстояние городских кварталов (манхэттенское расстояние). Согласно этой метрике, расстояние между двумя точками равно сумме модулей разностей их координат. Для этой меры влияние отдельных больших разностей (выбросов) уменьшается, так как они не возводятся в квадрат.

   $\rho(x, x') = \sum_i^n |x_i - x_i'|$

4. Расстояние Чебышева. Это расстояние может оказаться полезным, когда нужно определить два объекта как «различные», если они сильно различаются по какой-либо одной координате.

   $\rho(x, x') = \max_{i = 1, \ldots, n} |x_i - x_i'|$

5. Степенное расстояние.

   $\rho(x, x') = \sqrt[r]{\sum_i^n (x_i - x_i')^p}$

Кластеризация отличается от классификации (обучения с учителем) тем, что метки исходных объектов y_i изначально не заданы, а в прикладных задачах, как правило, неизвестно даже само множество Y.

Задачу кластеризации можно ставить как задачу дискретной оптимизации: необходимо так приписать номера кластеров y_i объектам x_i, чтобы значение выбранного функционала качества приняло наилучшее значение. Существует много разновидностей функционалов качества кластеризации, но нет «самого правильного» функционала. По сути, каждый метод кластеризации можно рассматривать как точный или приближённый алгоритм поиска оптимума некоторого функционала.

Среднее внутрикластерное расстояние должно быть как можно меньше:

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

Среднее межкластерное расстояние должно быть как можно больше:

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

Если алгоритм кластеризации вычисляет центры кластеров μ_y, y ∈ Y, то можно определить функционалы, вычислительно более эффективные. Сумма средних внутрикластерных расстояний должна быть как можно меньше:

$$\Phi_0 = \sum_{y \in Y} \frac{1}{|K_y|} \sum_{i:\, y_i = y} \rho^2(x_i, \mu_y) \to min,$$

где $K_y = \{x_i \in X^l | y_i = y\}$ – кластер с номером y. В этой формуле можно было бы взять не квадраты расстояний, а сами расстояния. Однако, если ρ – евклидова метрика, то внутренняя сумма в Φ_0 приобретает физический смысл момента инерции кластера K_y относительно его центра масс, если рассматривать кластер как материальное тело, состоящее из $|K_y|$ точек одинаковой массы.

Сумма межкластерных расстояний должна быть как можно больше:

$$\Phi_1 = \sum_{y \in Y} \rho^2(\mu_y, \mu) \to max,$$

где μ – центр масс всей выборки. На практике вычисляют отношение пары функционалов, чтобы учесть как межкластерные, так и внутрикластерные расстояния:

$\frac{F_0}{F_1} \to min$, либо $\frac{\Phi_0}{\Phi_1} \to min$

2 Статистические алгоритмы кластеризации

Как уже было сказано выше, не все алгоритмы кластеризации подходят для больших данных. Например, в условиях огромной выборки нет возможности вычислять расстояния между всеми парами объектов. Поэтому алгоритмы, основанные на графах, или агломеративные процедуры кластеризации не эффективны в такой ситуации. Для работы с большими данными можно использовать статистические методы кластерного анализа.

Статистические алгоритмы [1] основаны на предположении, что кластеры хорошо описываются некоторым семейством вероятностных распределений. Тогда задача кластеризации сводится к разделению смеси распределений по конечной выборке. Предположим, что объекты выборки X^l появляются случайно и независимо согласно вероятностному распределению, представляющему собой смесь распределений

$$p(x) = \sum_{y \in Y} w_y p_y(x), \qquad \sum_{y \in Y} w_y = 1,$$

где $p_y(x)$ – функция плотности распределения кластера y, w_y – неизвестная априорная вероятность появления объектов из кластера y. Пусть объекты описываются n числовыми признаками $f_1(x), \ldots, f_n(x)$, $X = R^n$. Каждый кластер y ∈ Y описывается n-мерной гауссовской плотностью $p_y(x) = N(x; \mu_y, \Sigma_y)$ с центром $\mu_y = (\mu_{y1}, \ldots, \mu_{yn})$ и диагональной ковариационной матрицей $\Sigma_y = diag(\sigma_{y1}^2, \ldots, \sigma_{yn}^2)$. При этих предположениях задача кластеризации совпадает с задачей разделения смеси вероятностных распределений, и для её решения можно применить EM-алгоритм.

EM-алгоритм заключается в итерационном повторении двух шагов. На E-шаге (expectation) по формуле Байеса вычисляются скрытые переменные g_{iy}. Значение g_{iy} равно вероятности того, что объект $x_i \in X^l$ принадлежит кластеру $y \in Y$. На M-шаге (maximization) уточняются параметры каждого кластера (μ_y, Σ_y), при этом существенно используются скрытые переменные g_{iy}.

2.1 Алгоритм K-Means

Метод k-средних [1] является упрощенной версией EM-алгоритма. Основное отличие заключается в том, что в EM-алгоритме каждый объект x_i распределяется по всем кластерам с вероятностями $g_{iy} = \mathrm{P}\{y_i = y\}$. В методе k-means каждый объект жёстко приписывается только к одному кластеру, то есть $g_{iy} = [y_i = y],\ i = 1,\ldots,n,$ $y \in Y$. Подробный алгоритм работы k-means:

1. Надо выбрать количество кластеров k, которое нам кажется оптимальным для этих данных.

2. Сформировать начальное приближение центров всех кластеров $y \in Y$. Так как k-means крайне чувствителен к выбору начальных приближений центров, лучше следовать некоторым эвристикам выбора начальных точек. Например, брать в качестве центров наиболее удалённые точки выборки: первые две точки выделяются по максимуму всех попарных расстояний; каждая следующая точка выбирается так, чтобы расстояние от неё до ближайшей уже выделенной было максимально. Случайная инициализация центров может приводить к плохому качеству кластеризации.

3. **Повторять.**

4. Отнести каждый объект к ближайшему центру (аналог E-шага):
$$y_i = \underset{y \in Y}{argmin}\, \rho(x_i, \mu_y),\ i = 1,\ldots,l.$$

5. Вычислить новое положение центров (аналог M-шага):
$$\mu_{yj} = \frac{\sum_{i=1}^{l}[y_i{=}y] f_j(x_i)}{\sum_{i=1}^{l}[y_i{=}y]},\ y \in Y,\ j = 1,\ldots,n,$$

   в этой формуле вычисляется центр масс кластера отдельно для каждой из n координат. Массы всех точек равны единице, $f_j(x_i)$ возвращает j-ю координату точки x_i, а квадратные скобки означают равенство 0, если выражение, в них заключённое, ложно, и 1 – если истинно.

6. **Пока** положения центров кластеров не перестанут изменяться на величину, меньшую eps. Eps – параметр алгоритма.

Кластеризация может оказаться плохой, если изначально будет неправильно угадано число кластеров. Для этого можно воспользоваться следующим функционалом:

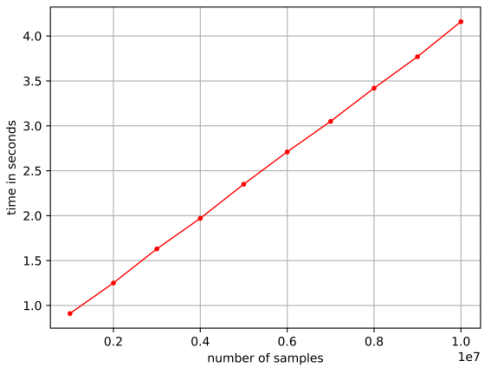
$$J(C) = \sum_{k=1}^{K} \sum_{x_i \in C_k} \rho(x_i, \mu_k)^2 \rightarrow \min_{C},$$

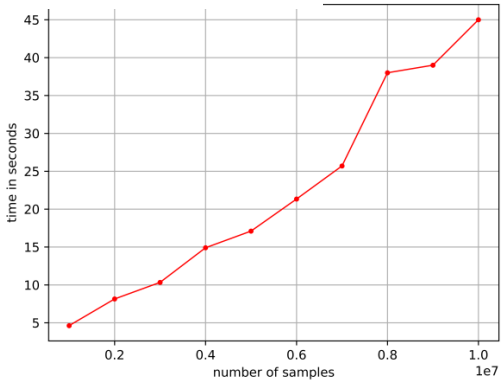
где C – множество кластеров мощности K, μ_k – центр кластера C_k. Очевиден здравый смысл: мы хотим, чтобы точки располагались кучно возле центров своих кластеров, но минимум такого функционала будет достигаться тогда, когда кластеров столько же, сколько и точек, то есть каждая точка – это кластер из одного элемента. Для решения этого вопроса можно минимизировать такой функционал:

$$D(k) = \frac{|J(C_k) - J(C_{k+1})|}{|J(C_{k-1}) - J(C_k)|} \rightarrow \min_{k},$$

здесь C_k означает не k-ый кластер, а множество кластеров мощности k. То есть фактически выбираем то число кластеров, начиная с которого описанный функционал $J(C)$ падает «уже не так быстро».

Одним из главных достоинств алгоритма k-means является его применимость к большим данным. Дадим асимптотическую оценку этого метода. Пусть n – количество объектов для кластеризации, k – количество кластеров, d – размерность пространства (количество признаков у каждого объекта), i – количество итераций алгоритма до сходимости. Тогда последовательная сложность будет $O(ikdn)$. Заметим, что k и d – константы, а количество итераций i не имеет тенденции к росту с ростом числа элементов в выборке. Например, алгоритмы, которые вычисляют расстояния между всеми парами объектов, будут иметь последовательную сложность $O(n^2)$ в лучшем случае. Ниже представлены графики зависимости времени работы алгоритма от количества объектов в выборке при фиксированных d и k. На левом графике d = 2, k = 3, n изменяется от одного миллиона до десяти миллионов с шагом в один миллион. Зависимость линейная. На правом графике n изменяется также, но d = 30, k = 5. График получился не настолько ровный, как в предыдущем случае, потому что сложно контролировать качество сгенерированного датасета в 30-мерном пространстве. Тем не менее, зависимость времени работы от объема данных напоминает линейную функцию. Измерения проводились по четырем запускам с последующим усреднением. Таким образом, теоретическая оценка сложности алгоритма k-means подтверждена.





Рассмотрим небольшую модификацию алгоритма k-средних, которая называется k-means++. Отличие заключается в нахождении более удачных начальных значений центров кластеров. Алгоритм работы k-means++:

1. Выбрать первый центр среди всех точек случайным образом.

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

3. Выбрать из этих точек следующий центр так, чтобы вероятность выбора точки была пропорциональна вычисленному для неё квадрату расстояния.

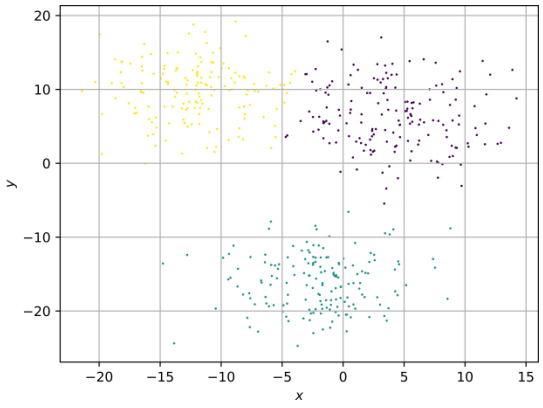
4. Повторять шаги 2 и 3 до тех пор, пока не будет найдено нужное количество центров.

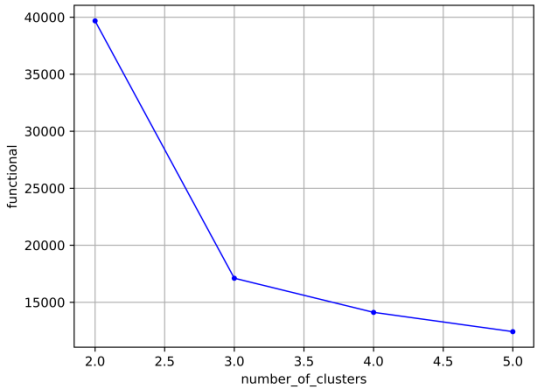
Затем работает основной алгоритм k-means.

Несмотря на то, что k-means++ использует неслучайную начальную инициализацию кластеров, лучше запускать алгоритм несколько раз с разными начальными центрами, а потом выбирать наилучшую кластеризацию. Для этого, например, можно использовать внутренние метрики кластеризации [2]: silhouette_score, calinski_harabasz_score, davies_bouldin_score. Так же можно воспользоваться функционалами качества кластеризации Φ_0 и Φ_1, определенными выше. Надо отметить, что применение silhouette_score при работе с большими данными невозможно из-за слишком большого количества вычислений, а calinski_harabasz_score и davies_bouldin_score отлично работают с большими данными. Из-за невозможности распараллеливания k-means++, запускалось

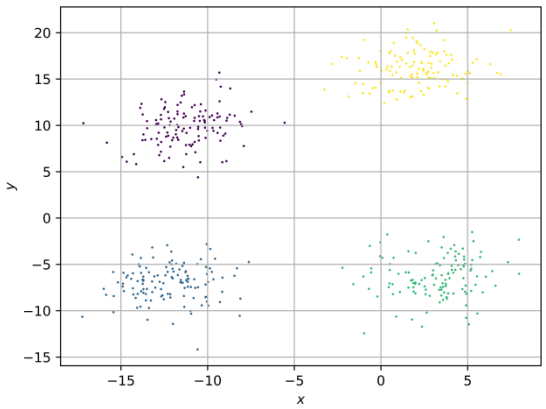
параллельно 8 процедур с разными начальными положениями центров на 8-ядерном процессоре, затем выбирался наилучший результат. Для вычислительных экспериментов с алгоритмом k-means++ использовалась собственная реализация данного метода.

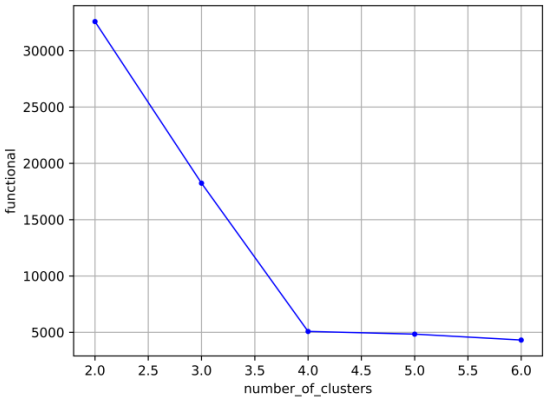
Результаты вычислительных экспериментов с алгоритмом k-means++





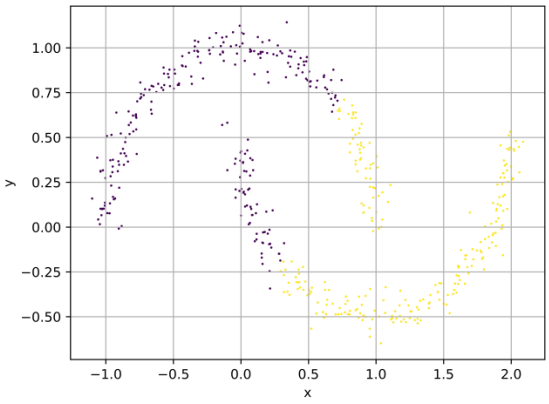
Здесь был использован описанный способ выбора числа кластеров. Функционал J(C) резко уменьшает скорость падения после 3 кластеров, значит это оптимальное число кластеров в данной ситуации. Мы можем в этом убедиться на левой картинке. Ниже приведены результаты такого же эксперимента, но с другими данными.

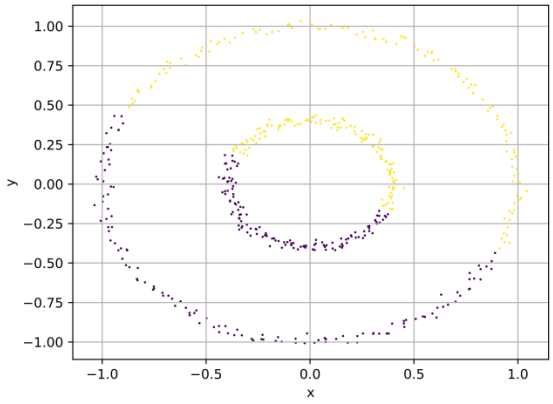




Когда объекты, которые относятся к одной группе, расположены не кучно, а образуют кластеры произвольной формы, алгоритм k-means++ работает явно неправильно. Это обусловливается невыполнением изначального предположения о том, что каждый кластер описывается n-мерной гауссовской плотностью.

Ниже приведены результаты таких экспериментов.





Была проверена работа алгоритма k-means++ с большими данными. Визуализировать результаты таких экспериментов не представляется возможным, поэтому выводы будут приведены в письменной форме. Так как в данном алгоритме присутствует случайность, каждая процедура была запущена по четыре раза, а затем усреднялось время работы программы. Выборку из миллиона элементов в 5-мерном пространстве алгоритм k-means++ кластеризовал в среднем за 3.3 секунды. Затем были сгенерированы выборки из десяти миллионов объектов в 20-мерном пространстве и ста миллионов объектов в 2-мерном пространстве. На таких выборках k-means++ работал в среднем 25 секунд и 64 секунды соответственно, что вполне приемлемо. Была решена реальная прикладная задача кластеризации с помощью алгоритма k-means++. Использовался датасет [3], содержащий информацию обо всех внутренних рейсах в США за 2018 год. Выборка состояла приблизительно из семи миллионов объектов. Каждый полет характеризовался 28 признаками: дата рейса, аэропорты вылета и прилета, время вылета, задержка рейса и т.д. Была проведена обработка данных: удаление ненужных признаков, нормализация значений, добавление нормально распределенного шума. Все категориальные признаки были переведены в числовые. Алгоритм k-means++ разбил данную выборку на 3 кластера за 9 секунд. Для выбора оптимального числа кластеров использовался описанный выше метод. Далее был проведен анализ результатов работы алгоритма. Оказалось, что в одном из кластеров четыре самые часто встречающиеся аэропорты вылета расположены географически рядом, а именно на побережье Тихого океана. В другом, самом немногочисленном кластере (230000 объектов) были собраны рейсы, вылетевшие с огромным опозданием, в среднем – 3 часа.

2.2 Алгоритм Mini Batch K-Means

Алгоритм Mini Batch K-Means [4] является существенной модернизацией k-means++. Вначале происходит выбор центров кластеров, согласно процедуре k-means++. Затем на каждой итерации работы алгоритма произвольным образом выбирается фиксированное число (оно является параметром алгоритма) объектов из выборки. Для них вычисляются расстояния до центров кластеров, и эти объекты относятся к нужным кластерам, остальные элементы выборки не меняют своей принадлежности к кластерам. Далее происходит перевычисление центров, если состав кластеров изменился. И так далее до сходимости алгоритма. Таким образом, на каждой итерации обновляется только какое-то число элементов, вместо всей выборки. Это существенно упрощает вычисления в условиях сверхбольшого количества объектов для кластеризации. Главное назначение Mini Batch K-Means – выборки гигантского размера, с которыми не справляется обычный k-means или k-means++. Был проведен вычислительный эксперимент по кластеризации выборки из десяти миллионов объектов с двадцатью признаками и ста миллионов объектов с двумя признаками с помощью Mini Batch K-Means. Реализация была взята из библиотеки Scikit-learn для Python. Время работы процедуры составило в среднем 9.2 секунды и 7.8 секунд соответственно, на этих же данных алгоритм k-means++ работал 25 секунд и 64 секунды. Видно, что Mini Batch K-Means показывает более чем восьмикратный прирост производительности на выборке из ста миллионов объектов, потребляя при этом не более 4.7 гигабайт оперативной памяти, в то время как классический k-means++ использовал 12.8 гигабайт оперативной памяти. При этом качество кластеризации для выборки из ста миллионов элементов, определяемое внешней метрикой Adjusted Rand Index (ARI) [5], для k-means++ составило 0.987442, а для Mini Batch K-Means – 0.987435. Ухудшения качества практически не наблюдается. Mini Batch K-Means был проверен на той же самой прикладной задаче с внутренними рейсами в США. Данный алгоритм произвел аналогичную кластеризацию, что и k-means++, но в среднем за 1.79 секунды. Скорость работы Mini Batch K-Means очень сильно зависит от выбора того количества объектов, которые будут обновляться на каждой итерации. В ходе экспериментов было обнаружено, что наилучшее значение для batch_size примерно в 20-50 раз меньше, чем количество элементов в самой выборке.

3 Алгоритмы кластеризации, основанные на плотности

В этом разделе будут рассмотрены алгоритмы кластеризации для больших данных, основанные на анализе плотности расположения объектов. Эти процедуры обладают рядом преимуществ и недостатков по сравнению с EM-алгоритмами. Одно из самых важных преимуществ заключается в том, что плотностные алгоритмы не требуют числа кластеров в качестве входного параметра, они сами разбивают выборку на подходящее число кластеров. Это важно, когда эксперт не может предположить ожидаемого количества кластеров. В случае использования, например, статистического алгоритма пришлось бы запускать его отдельно для каждого числа кластеров.

3.1 Алгоритм кластеризации DBSCAN

DBSCAN (Density-based spatial clustering of applications with noise) [6] – это алгоритм кластеризации, основанный на плотности. Если дан набор точек в некотором пространстве, алгоритм группирует вместе точки, которые тесно расположены, помечая как выбросы точки, которые находятся одиноко в областях с малой плотностью. На вход алгоритм требует два параметра: радиус eps-окрестности и minPts – количество соседей.

Рассмотрим набор точек в некотором пространстве, требующий кластеризации. Для выполнения кластеризации DBSCAN точки делятся на основные точки, достижимые по плотности точки и выпадающие следующим образом:

- Точка $\boldsymbol{p}$ является основной точкой, если по меньшей мере minPts точек находятся в её eps-окрестности, включая саму точку $\boldsymbol{p}$.

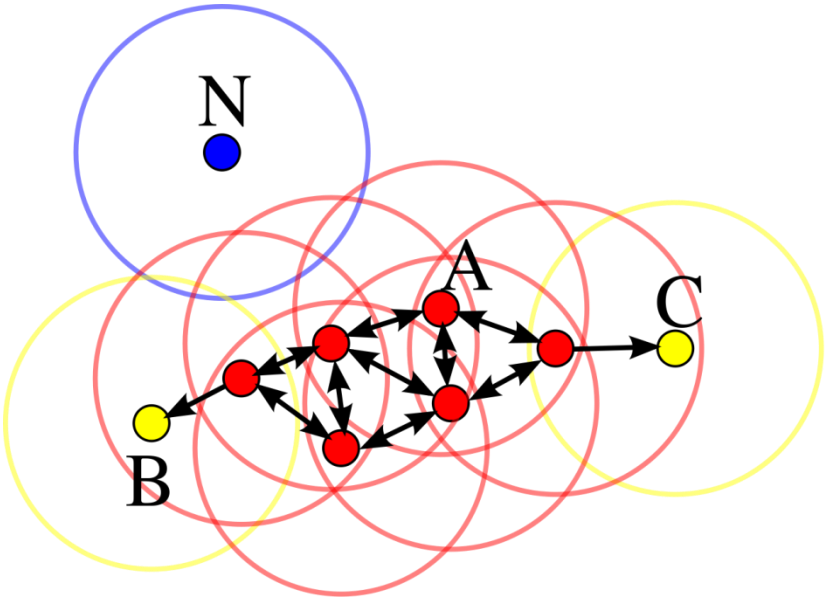
- Точка $\boldsymbol{q}$ прямо достижима из $\boldsymbol{p}$, если точка $\boldsymbol{q}$ находится в eps-окрестности точки $\boldsymbol{p}$ и $\boldsymbol{p}$ является основной точкой.

- Точка $\boldsymbol{q}$ достижима из $\boldsymbol{p}$, если имеется путь $\boldsymbol{p_1, \ldots, p_n}$, такой что $\boldsymbol{p_1 = p}$ и $\boldsymbol{p_n = q}$, где каждая точка $\boldsymbol{p_{i+1}}$ достижима прямо из $\boldsymbol{p_i}$, а все точки на пути являются основными, за исключением $\boldsymbol{q}$.

- Все точки, не достижимые из основных точек, считаются выбросами.

Теперь, если $\boldsymbol{p}$ является основной точкой, то она формирует кластер вместе со

всеми точками (основными или неосновными), достижимыми из этой точки $\boldsymbol{p}$. Каждый кластер содержит, по меньшей мере, одну основную точку. Неосновные точки могут быть частью кластера, но они формируют его «край», поскольку не могут быть использованы для достижения других точек.



На этой диаграмме minPts = 4. Точка A и другие красные точки являются основными. Поскольку все они достижимы друг из друга, точки образуют один кластер. Точки B и C основными не являются, но они достижимы из основных точек, и также принадлежат кластеру. Точка N является точкой шума, она не является ни основной, ни достижимой.

Алгоритм работы DBSCAN:

1. Берем случайный элемент выборки.

2. Если в его eps-окрестности меньше minPts точек, то переносим его в список возможных выбросов и выбираем другой объект.

3. **Иначе:**

    a. Исключаем его из списка элементов, которые надо обойти.

    b. Отмечаем эту точку основной.

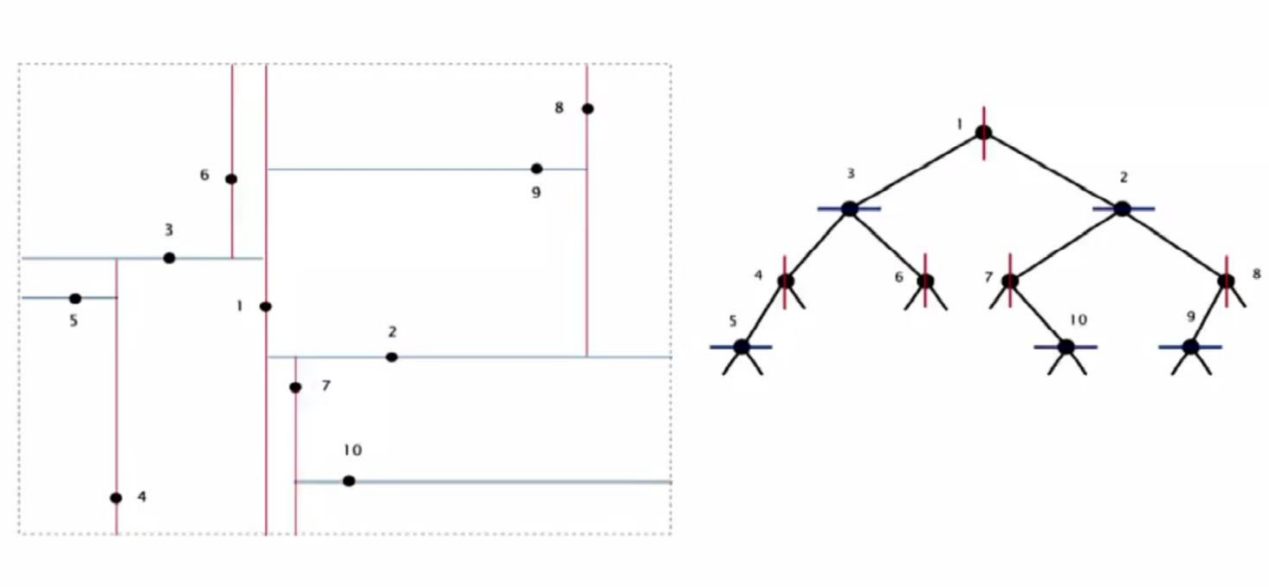
c. Обходим все точки, попавшие в eps-окрестность этой точки. Если соседняя точка уже в списке потенциальных выбросов или в её eps-окрестности меньше minPts точек, то эта точка – край кластера, отмечаем её жёлтым флагом, присоединяем к группе и продолжаем обход. Если соседняя точка тоже оказывается основной точкой, то она не образует новый кластер, а присоединяется к уже созданному, и добавляем в список обхода её соседние точки. **Повторяем** этот пункт, пока список обхода не окажется пуст.

4. **Повторяем** шаги 1-3, пока не обойдём все объекты.

5. Отмечаем оставшиеся в списке потенциальных выбросов объекты шумом.

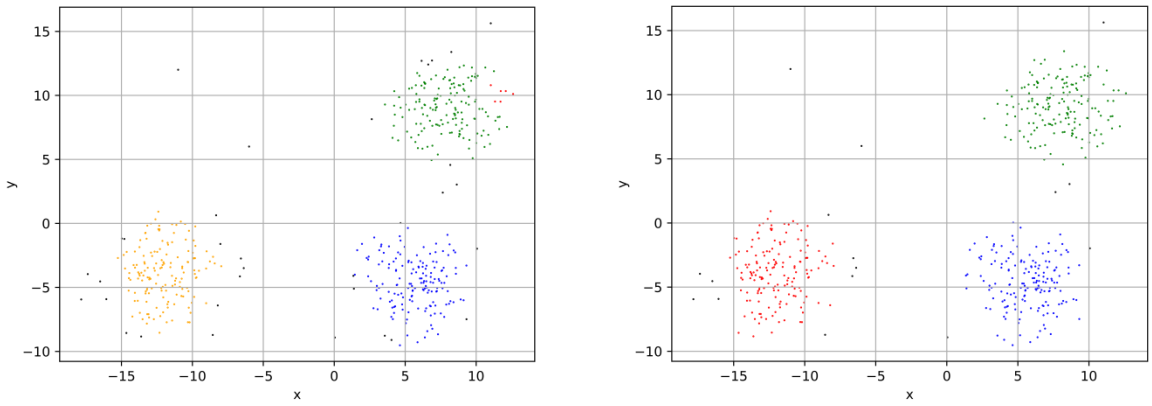
Можно заметить, что алгоритм обхода точек в DBSCAN очень напоминает алгоритм обхода графа в ширину (BFS). DBSCAN с не случайным правилом обработки краевых точек детерминирован. Однако большинство реализаций для ускорения работы и уменьшения количества параметров отдают краевые точки ближайшим кластерам. Как правило, это несильно влияет на качество работы алгоритма, ведь через граничные точки кластер всё равно не распространяется дальше – ситуация, когда точка перескакивает из кластера в кластер и «открывает дорогу» к другим точкам, невозможна. Алгоритм DBSCAN находит выбросы, это бывает очень важно, особенно, когда надо найти объекты, поведение которых отличается от общего поведения толпы.

Проблема заключается в том, что в ходе работы алгоритма для каждой точки надо найти все точки, находящиеся в её eps-окрестности. А для этого надо вычислять расстояния между всеми парами объектов, что нельзя делать при работе с большими данными. Для решения этой проблемы в алгоритме DBSCAN используют k-мерные деревья (k-d деревья) [6]. Ниже приведен пример 2-мерного дерева. Работа с k-d деревьями не отличается от работы с обычными деревьями поиска, за исключением того, что оси, вдоль которых происходит сравнение в каждом узле, циклически чередуются. Поэтому добавление и поиск элементов в k-d дереве имеет в среднем логарифмическую сложность.

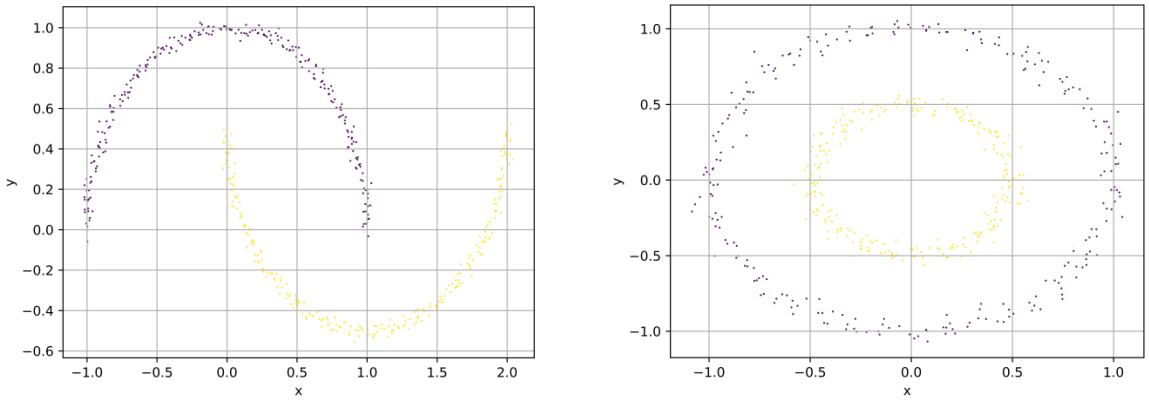


В алгоритме DBSCAN поиск соседей с помощью k-d деревьев можно осуществлять следующим образом: ищем все точки, попадающие в k-мерный куб с ребром длины 2ε, в центре которого находится точка, соседей которой надо найти. Для найденных точек проверяем расстояние до рассматриваемой точки, так как k-мерный шар вписан в k-мерный куб. Для вычислительных экспериментов реализация DBSCAN была взята из библиотеки Scikit-learn для Python.

Результаты вычислительных экспериментов с алгоритмом DBSCAN



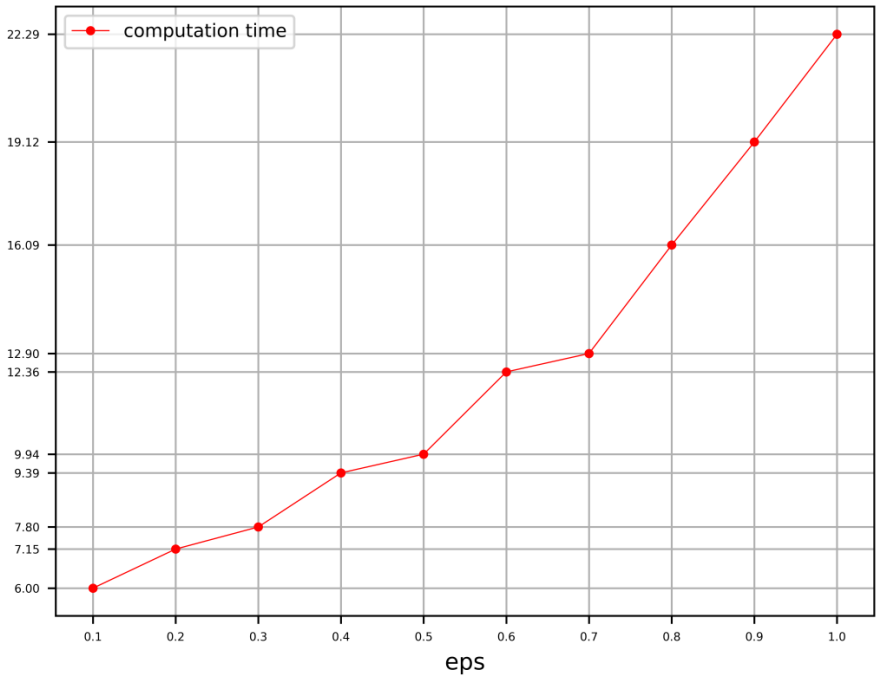
Недостатком DBSCAN является сложность подбора параметров ε и minPts.
На левом изображении $\varepsilon = 1$, minPts = 4 (4 кластера и выбросы). На правом изображении $\varepsilon = 1.5$, minPts = 4 (3 кластера и выбросы). Качество кластеризации на правом изображении существенно лучше.

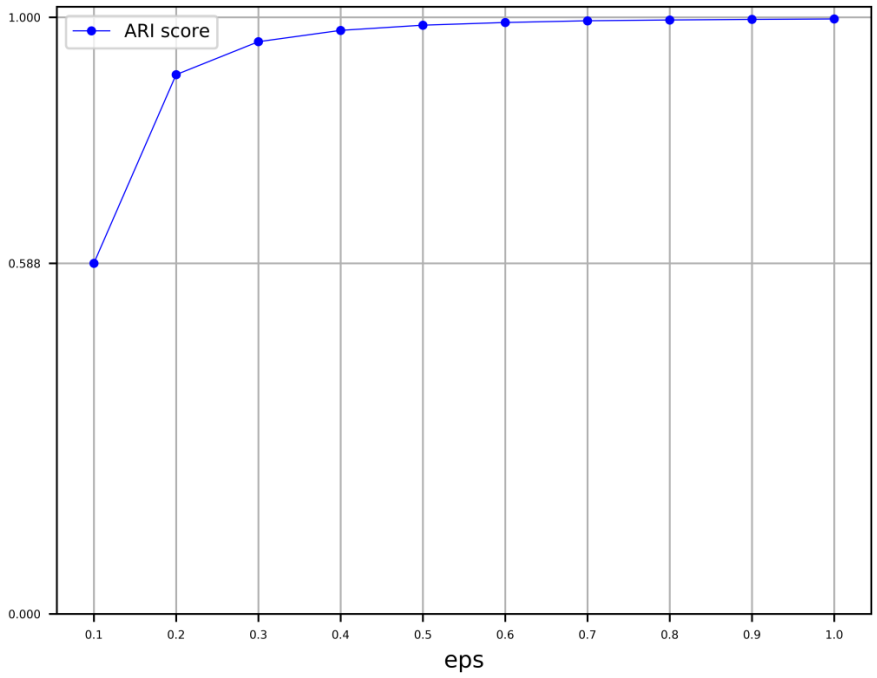


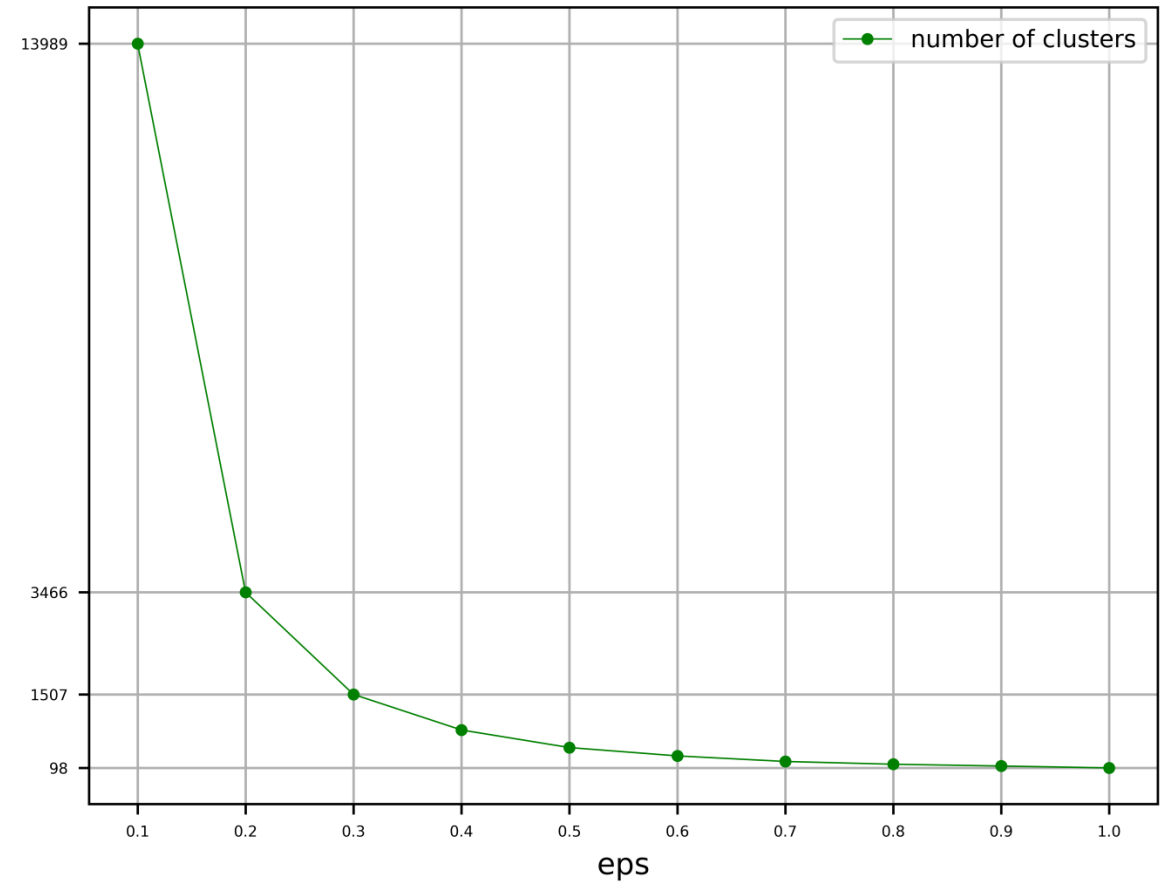
С данными необычной формы алгоритм DBSCAN в отличие от k-means справляется прекрасно. Это объясняется тем, что DBSCAN основывается на плотности расположения объектов.

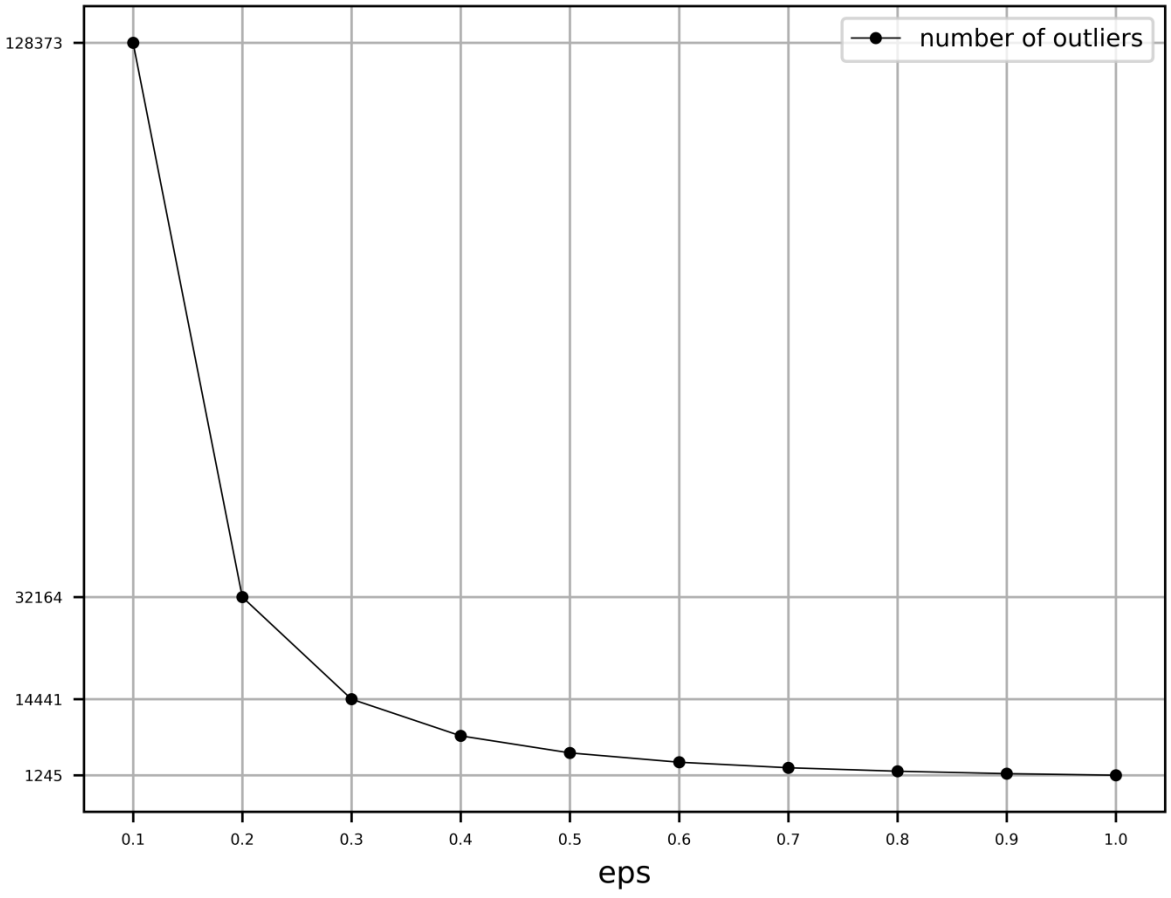
Алгоритм DBSCAN также был протестирован с большими данными. Разумеется, при работе с большими данными использовалось ускорение с помощью k-d деревьев. DBSCAN детерминирован, так что замеры времени работы производились по одному запуску процедуры. Выборку из миллиона объектов в двумерном пространстве алгоритм DBSCAN кластеризовал за 21 секунду с качеством 0.9972 по внешней метрике ARI. Параметры eps и minPts были равны соответственно 1 и 4. При этом алгоритм обнаружил 98 кластеров и 1245 выбросов, хотя датасет генерировался с тремя четко выраженными скоплениями точек. Это объясняется тем, что DBSCAN на краях трех основных кластеров образовал много маленьких кластеров, а еще более удаленные точки отнес к выбросам. Эту проблему данного алгоритма можно решить последующим анализом полученных кластеров. Датасет из миллиона объектов в 10-мерном пространстве DBSCAN кластеризовал за 18 минут. Провести эксперименты с бóльшим количеством данных не получилось, так как данная реализация алгоритма оказалась очень требовательной к объему оперативной памяти. Во время работы с миллионом объектов в двумерном пространстве процесс использовал 8 гигабайт памяти, а при работе с двумя миллионами объектов в двумерном пространстве было использовано все 16 гигабайт памяти и выделено еще 10 гигабайт файла подкачки, поэтому вычисления заняли очень много времени. Это вызвано тем, что алгоритм сначала предвычисляет и сохраняет всех соседей каждой точки в заданной окрестности. Для решения этой проблемы надо искать соседей для каждой точки во время её обработки. Это немного увеличит время работы алгоритма, зато сделает его намного

более эффективным в потреблении памяти. В ходе экспериментов было замечено, что время работы DBSCAN очень сильно зависит от параметра eps (радиус поиска соседей). Это объясняется тем, что чем меньше eps, тем меньше ветвей k-d дерева надо обходить при поиске соседей для каждой точки. Параметр minPts практически не влияет на скорость работы алгоритма. Ниже представлены графики зависимости времени работы (в секундах), ARI score, количества кластеров и количества выбросов от параметра eps на одних и тех же данных. Выборка состоит из миллиона точек в двумерном пространстве, minPts равняется 4, eps меняется от 0.1 до 1.0 с шагом 0.1.






number of clusters
13989
3466
1507
98
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
eps


number of outliers
128373
32164
14441
1245
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
eps

Вывод

В данном реферате были рассмотрены некоторые алгоритмы кластеризации для больших данных. Перечислим плюсы и минусы рассмотренных процедур.

- K-means и его модификации хорошо работают с данными, которые образуют кластеры сферической и эллиптической формы, можно запускать параллельно несколько процессов с разной начальной инициализацией центров кластеров. Mini Batch K-Means очень быстро работает с большими данными.

- K-means и его модификации крайне плохо работают с данными, которые образуют кластеры необычной формы, не определяют выбросы, требуют количество кластеров в качестве входного параметра.

- DBSCAN очень хорошо работает со сгустками данных произвольной формы, автоматически определяет количество кластеров, находит выбросы.

- DBSCAN работает достаточно медленно с большими данными по сравнению с EM-алгоритмами, требует подбора параметров ε и minPts, плохо работает с данными, которые образуют области различной плотности. Особенно это ощущается на данных высокой размерности.

Список литературы

[1] К. В. Воронцов «Лекции по алгоритмам кластеризации и многомерного шкалирования» (21.12.2007)

[2] https://scikit-learn.org/stable/modules/classes.html?highlight=metrics#module-sklearn.metrics

[3] https://www.kaggle.com/yuanyuwendymu/airline-delay-and-cancellation-data-2009-2018

[4] https://scikit-learn.org/stable/modules/clustering.html#mini-batch-kmeans

[5] https://scikit-learn.org/stable/modules/clustering.html#adjusted-rand-score

[6] S.Vijayalaksmi, M. Punithavalli «A Fast Approach to Clustering Datasets using DBSCAN and Pruning Algorithms». International Journal of Computer Applications (0975 – 8887). Volume 60– No.14, December 2012