Методики обучения без учителя

Хотя большинство приложений МО в наши дни основано на обучении с учителем (и как следствие именно сюда направлен основной объем инвестиций), подавляющая масса доступных данных являются непомеченными: мы имеем входные признаки **X**, но не располагаем метками **y**. Ученый в области компьютерных наук Ян Лекун превосходно отметил, что "если бы интеллект был тортом, то обучение без учителя было бы основой торта, обучение с учителем — сахарной глазурью на нем, а обучение с подкреплением — вишенкой на торте". Другими словами, обучение без учителя обладает огромным потенциалом, куда мы только начали вонзать свои зубы.

Допустим, вы хотите создать систему, которая будет делать несколько фотографий каждого изделия на производственной линии и определять, какие из них дефектные. Вы можете довольно легко создать систему, которая будет автоматически делать снимки и ежедневно выдавать вам тысячи фотографий. Всего за несколько недель вы сумеете построить достаточно крупный набор данных. Но подождите, меток-то нет! Если вы пожелаете обучить обычный двоичный классификатор, который будет прогнозировать, является изделие дефектным или нет, то вам потребуется пометить все до единого фотографии изделий как "дефектное" или "нормальное". Как правило, это будет требовать привлечения людей-экспертов, которым придется вручную перебрать все фотографии. Задача долгая, затратная и утомительная, так что она обычно будет выполняться для небольшого подмножества доступных фотографий. В результате помеченный набор данных будет довольно малым, а эффективность классификатора окажется неутешительной. Кроме того, каждый раз, когда компания вносит какие-то изменения в выпускаемые изделия, весь процесс необходимо начинать с нуля. Разве не было бы замечательно, если бы алгоритм сумел задействовать непомеченные данные, не заставляя людей помечать каждую фотографию? Вот тут на помощь и приходит обучение без учителя.

В главе 8 мы рассматривали самую распространенную задачу обучения без учителя: понижение размерности. В настоящей главе мы ознакомимся с дополнительными задачами и алгоритмами обучения без учителя.

Кластеризация

Цель заключается в группировании похожих образцов в *кластеры*. Кластеризация представляет собой великолепный инструмент для анализа данных, сегментации заказчиков, систем выдачи рекомендаций, поисковых механизмов, сегментирования изображений, частичного обучения, понижения размерности и многого другого.

Обнаружение аномалий

Целью является выяснение того, каким образом выглядят "нормальные" данные, с последующим применением этих знаний для обнаружения ненормальных образцов, таких как дефектные изделия на производственной линии или новая тенденция во временном ряде.

Оценка плотности

Представляет собой задачу оценки *функции плотности вероятности* (***probability density function — PDF***) случайного процесса, который генерирует набор данных. Оценка плотности часто используется для обнаружения аномалий: образцы, расположенные в областях с очень низкой плотностью, вероятнее всего будут аномалиями. Она также полезна для анализа данных и визуализации.

Готовы к продолжению? Мы начнем с кластеризации посредством алгоритмов *K-Means (**K-средние***) и *DBSCAN* (***Density-based Spatial Clustering of Applications with Noise*** — плоскостная пространственная кластеризация приложений с шумом), после чего обсудим *модели со смесями гауссовых распределений* (***Gaussian mixture model***), а также посмотрим, каким образом их можно применять для оценки плотности, кластеризации и обнаружения аномалий.

Кластеризация

Наслаждаясь прогулкой в горах, вы наталкиваетесь на растение, которого никогда не видели раньше. Вы оглядываетесь вокруг и замечаете еще несколько. Растения не идентичны, но достаточно похожи, чтобы вы знали, что

почти наверняка они относятся к одному и тому же виду (или, по крайней мере, к тому же самому роду). Вам может понадобиться ботаник, который сообщит точный вид, но вам определенно не нужен эксперт для идентификации групп сходно выглядящих объектов. Прием называется *кластеризацией* (***clustering***): это задача опознавания похожих образцов и назначения их *кластерам*, или группам похожих образцов.

Как и в классификации, каждый образец назначается группе. Однако в отличие от классификации кластеризация является задачей обучения без учителя. Взгляните на рис. 9.1: слева показан набор данных `iris` (введенный в главе 4), где вид каждого образца (т.е. его класс) представлен с помощью отличающегося маркера. Это помеченный набор данных, для которого хорошо подходят такие алгоритмы классификации, как логистическая регрессия, методы опорных векторов или случайный лес. Справа изображен тот же набор данных, но без меток, а потому использовать какой-то алгоритм классификации больше нельзя. Именно теперь в игру вступают алгоритмы кластеризации: многие из них способны легко обнаружить левый нижний кластер. Не настолько очевидно, но также довольно легко увидеть нашими собственными глазами, что правый верхний кластер состоит из двух отдельных подкластеров. Тем не менее, набор данных имеет два дополнительных признака (длину и ширину чашелистика), которые здесь не представлены, а алгоритмы кластеризации могут эффективно задействовать все признаки, поэтому они хорошо идентифицируют три кластера (например, в случае применения модели со смесью гауссовых распределений только 5 образцов из 150 назначаются неправильному кластеру).

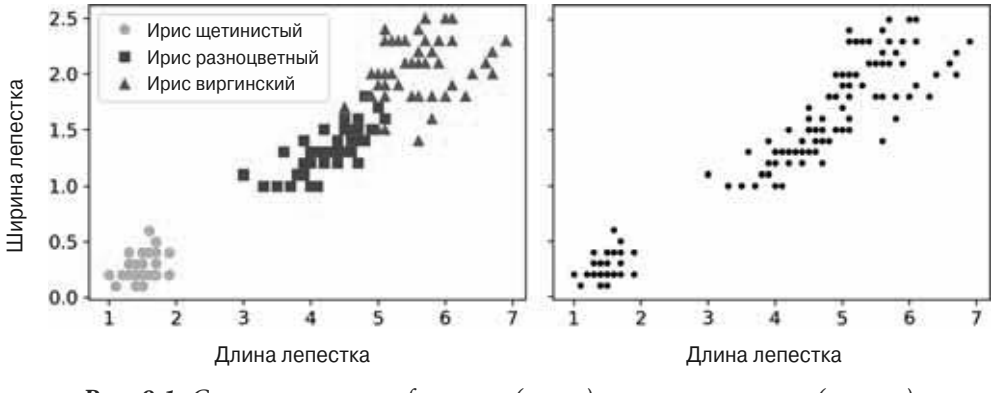

Рис. 9.1. *Сравнение классификации (слева) и кластеризации (справа)*

Кластеризация используется в разнообразных приложениях, включая перечисленные ниже.

Для сегментации заказчиков

Вы можете кластеризировать своих заказчиков на основе их покупок и активности на вашем веб-сайте. Это полезно для понимания, кто ваши заказчики и в чем они нуждаются, так что вы сумеете адаптировать свои товары и маркетинговые кампании к каждому сегменту. Скажем, сегментация заказчиков может быть полезной в *системах выдачи рекомендаций* для предложения содержимого, которое понравилось другим пользователям в том же самом кластере.

Для анализа данных

Когда вы анализируете новый набор данных, может быть полезно запустить алгоритм кластеризации и затем проанализировать каждый кластер по отдельности.

В качестве методики понижения размерности

После кластеризации набора данных обычно появляется возможность измерить *похожесть* (***affinity***) каждого образца с каждым кластером (похожесть — это любая мера того, насколько хорошо образец подгоняется к кластеру). Затем вектор признаков $\mathbf{x}$ каждого экземпляра может быть заменен вектором его похожестей с кластерами. Если есть k кластеров, тогда этот вектор будет k-мерным. Обычно он будет иметь гораздо меньшую размерность, чем исходный вектор признаков, но способен предохранять достаточно информации для дальнейшей обработки.

Для обнаружения аномалий (также называемого обнаружением выбросов)

Любой образец, который имеет низкую похожесть со всеми кластерами, может быть аномалией. Например, если вы кластеризируете пользователей своего веб-сайта на основе их поведения, то сможете выявлять пользователей с необычным поведением, таким как нестандартное количество запросов в секунду. Обнаружение аномалий особенно полезно при выявлении дефектов в производстве или при *обнаружении мошенничества*.

Для частичного обучения

Если вы располагаете незначительным числом меток, то могли бы выполнить кластеризацию и распространить метки на все образцы в од-

ном и том же кластере. Такая методика способна значительно увеличить количество меток, доступных для последующего алгоритма обучения с учителем, и тем самым увеличить его эффективность.

Для поисковых механизмов

Определенные поисковые механизмы позволяют искать изображения, которые похожи на опорное изображение. Для построения такой системы вы сначала примените алгоритм кластеризации ко всем изображениям в вашей базе данных; похожие изображения окажутся в одном кластере. Затем, когда пользователь предоставит опорное изображение, вам придется лишь воспользоваться обученной моделью кластеризации для нахождения кластера этого изображения и просто возвратить все изображения из найденного кластера.

Для сегментирования изображений

За счет кластеризации пикселей в соответствии с их цветом и последующей замены цвета каждого пикселя усредненным цветом его кластера становится возможным значительное сокращение количества разных цветов в изображении. Сегментирование изображений применяется во многих системах обнаружения и отслеживания объектов, т.к. оно облегчает распознавание контуров каждого объекта.

Не существует универсального определения того, что такое кластер: это действительно зависит от контекста, и разные алгоритмы будут захватывать отличающиеся виды кластеров. Одни алгоритмы ищут образцы, центрированные возле конкретной точки, называемой *центроидом*. Другие занимаются поиском непрерывных областей плотно упакованных образцов: такие кластеры могут принимать любую форму. Некоторые алгоритмы являются иерархическими и ищут кластеры кластеров. Список можно продолжить.

В текущем разделе мы исследуем два популярных алгоритма кластеризации, K-Means и DBSCAN, а также ознакомимся с их приложениями вроде нелинейного понижения размерности, частичного обучения и обнаружения аномалий.

K-Means

Рассмотрим непомеченный набор данных, представленный на рис. 9.2: вы можете четко заметить пять пятен образцов. Алгоритм K-Means — это простой алгоритм, который способен выполнить кластеризацию набора данных

такого рода очень быстро и эффективно, часто всего лишь за несколько итераций. Он был предложен Стюартом Ллойдом из Bell Labs в 1957 году как методика импульсно-кодовой модуляции, но опубликован за пределами компании только в 1982 году (`https://homl.info/36`)[1]. В 1965 году Эдвард Форги опубликовал фактически тот же самый алгоритм и потому K-Means иногда называют алгоритмом Ллойда-Форги.

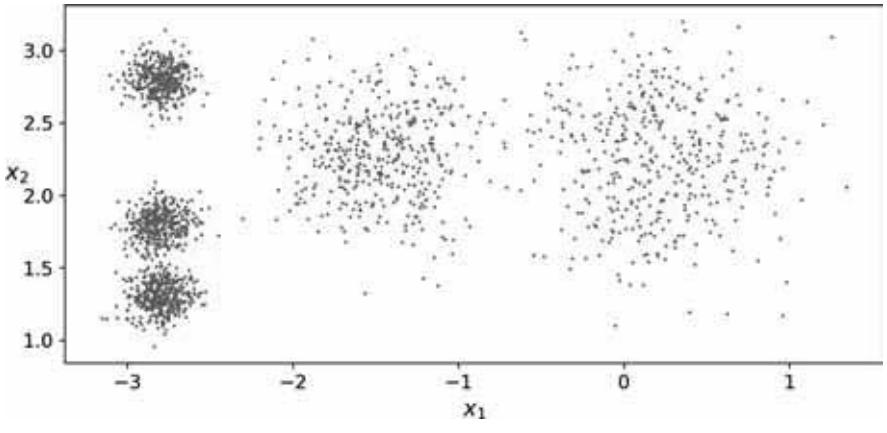

Рис. 9.2. *Непомеченный набор данных, состоящий из пяти пятен образцов*

Давайте обучим кластеризатор K-Means на таком наборе данных. Он будет пытаться отыскать центр каждого пятна и назначить каждый образец ближайшему пятну:

```
from sklearn.cluster import KMeans
k = 5
kmeans = KMeans(n_clusters=k)
y_pred = kmeans.fit_predict(X)
```

Обратите внимание, что вы должны указать количество кластеров k, которые алгоритм обязан найти. Взглянув на данные в текущем примере, вполне очевидно, что k необходимо установить в 5, но в целом все не так легко. Вскоре мы обсудим суть более подробно.

Каждый образец был назначен одному из пяти кластеров. В контексте кластеризации *метка* образца является индексом кластера, который алгоритм назначил образцу: ее не следует путать с метками классов в классифи-

[1] Стюарт Ллойд, *Least Squares Quantization in PCM* (Квантование методом наименьших квадратов в импульсно-кодовой модуляции), *IEEE Transactions on Information Theory* 28, выпуск 2 (1982 г.): с. 129–137.

кации (вспомните, что кластеризация представляет собой задачу обучения без учителя). Экземпляр `KMeans` сохраняет копию меток образцов, на которых он обучался, доступную через переменную экземпляра `labels_`:

```
>>> y_pred
array([4, 0, 1, ..., 2, 1, 0], dtype=int32)
>>> y_pred is kmeans.labels_
True
```

Вы также можете посмотреть на пять центроидов, которые обнаружил алгоритм:

```
>>> kmeans.cluster_centers_
array([[-2.80389616,  1.80117999],
       [ 0.20876306,  2.25551336],
       [-2.79290307,  2.79641063],
       [-1.46679593,  2.28585348],
       [-2.80037642,  1.30082566]])
```

Вы легко можете назначить новые образцы кластеру, чей центроид находится ближе всего:

```
>>> X_new = np.array([[0, 2], [3, 2], [-3, 3], [-3, 2.5]])
>>> kmeans.predict(X_new)
array([1, 1, 2, 2], dtype=int32)
```

Если вы вычертите границы решений кластеров, то получите диаграмму (или замощение) Вороного (Voronoi tessellation), показанную на рис. 9.3, где каждый центроид представлен посредством X.

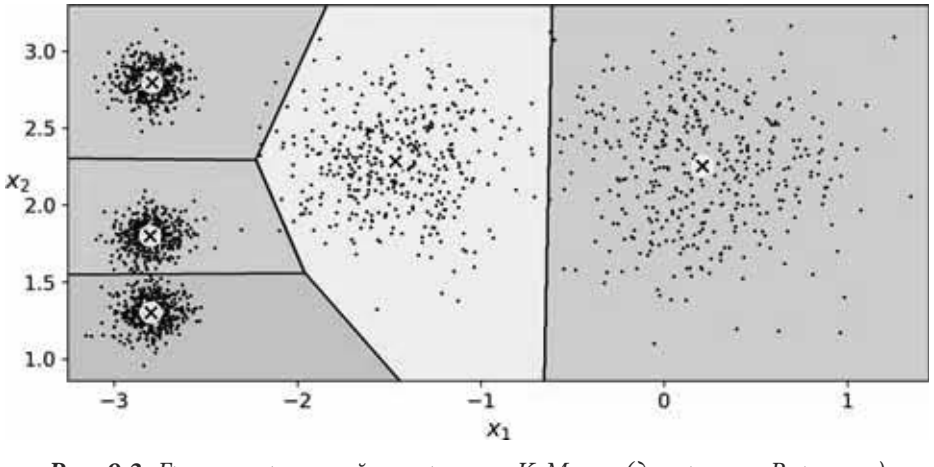

Рис. 9.3. *Границы решений алгоритма K-Means (диаграмма Вороного)*

Подавляющее большинство образцов были четко назначены надлежащим кластерам, но несколько образцов оказались, возможно, неправильно помеченными (особенно вблизи границы между левым верхним и центральным кластерами). В действительности алгоритм K-Means не очень хорошо работает, когда пятна имеют сильно отличающиеся диаметры, поскольку при назначении образца кластеру его заботит только расстояние до центроида.

Вместо назначения каждого образца одиночному кластеру, что называется *жесткой кластеризацией* (***hard clustering***), временами полезно выдавать каждому образцу оценку для каждого кластера, что называется *мягкой кластеризацией* (***soft clustering***). Оценкой может быть расстояние между образцом и центроидом; и наоборот, это может быть оценка близости (или похожести), такая как *гауссова радиальная базисная функция* (***Gaussian RBF***), представленная в главе 5. Метод `transform()` класса `KMeans` измеряет расстояние от каждого образца до всех центроидов:

```
>>> kmeans.transform(X_new)
array([[2.81093633, 0.32995317, 2.9042344 , 1.49439034, 2.88633901],
       [5.80730058, 2.80290755, 5.84739223, 4.4759332 , 5.84236351],
       [1.21475352, 3.29399768, 0.29040966, 1.69136631, 1.71086031],
       [0.72581411, 3.21806371, 0.36159148, 1.54808703, 1.21567622]])
```

В рассмотренном примере первый образец в `X_new` расположен на расстоянии 2.81 от первого центроида, 0.33 от второго центроида, 2.90 от третьего центроида, 1.49 от четвертого центроида и 2.89 от пятого центроида. Если вы трансформируете подобным образом имеющийся многомерный набор данных, то в итоге получите k-мерный набор данных: такая трансформация может быть очень эффективной методикой нелинейного понижения размерности.

Алгоритм K-Means

Итак, каким же образом работает алгоритм K-Means? Предположим, что вы получили центроиды. Вы могли бы легко снабдить метками все образцы в наборе данных, назначая каждый из них кластеру, чей центроид находится ближе всего. И наоборот, если бы вам предоставили все метки образцов, тогда вы легко смогли бы найти все центроиды, вычислив среднее по образцам для каждого кластера. Но вам не дают ни меток, ни центроидов — как вы сможете продолжить? Хорошо, просто начните со случайного размещения центроидов (скажем, выберите k образцов произвольным образом и исполь-

зуйте их местоположения в качестве центроидов). Затем снабдите метками образцы, обновите центроиды, снабдите метками образцы, обновите центроиды и продолжайте делать это до тех пор, пока центроиды прекратят перемещаться. Алгоритм гарантированно сходится за конечное число шагов (обычно довольно малое); он не будет колебаться вечно[2].

На рис. 9.4 алгоритм демонстрируется в действии: центроиды инициализируются случайным образом (слева вверху), затем помечаются образцы (справа вверху), далее центроиды обновляются (слева по центру), образцы помечаются повторно (справа по центру) и т.д. Как видите, всего лишь за три итерации алгоритм добился кластеризации, которая выглядит близкой к оптимальной.

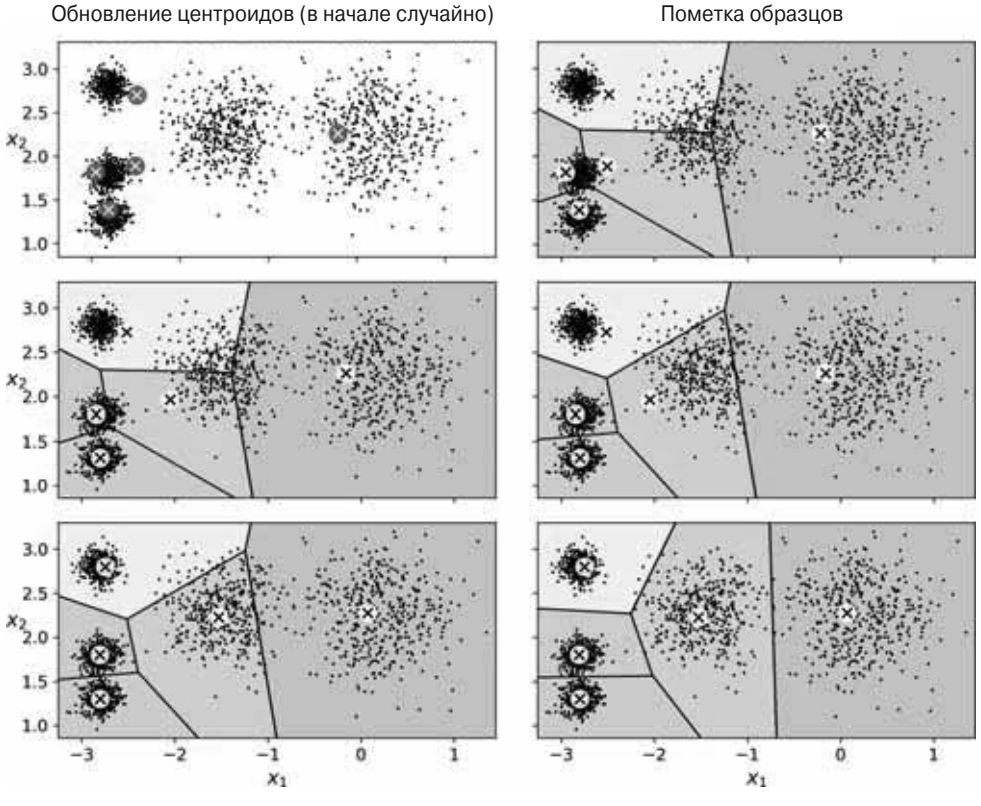

Рис. 9.4. *Алгоритм K-Means в действии*

[2] Причина в том, что среднеквадратическое расстояние между образцами и их ближайшими центроидами на каждом шаге может только уменьшаться.

Вычислительная сложность алгоритма обычно линейна в отношении количества образцов m, количества кластеров k и количества измерений n. Однако утверждение справедливо, только когда данные обладают кластерной структурой. Если кластерной структуры нет, то в худшем случае сложность может расти экспоненциально с увеличением количества образцов. На практике подобное происходит редко и, как правило, K-Means является одним из самых быстрых алгоритмов кластеризации.

Хотя алгоритм гарантированно сходится, он может не сойтись в правильное решение (т.е. сойтись в локальный оптимум): все зависит от инициализации центроидов. На рис. 9.5 показаны два субоптимальных решения, в которые может сходиться алгоритм, если шаг случайной инициализации окажется неудачным.

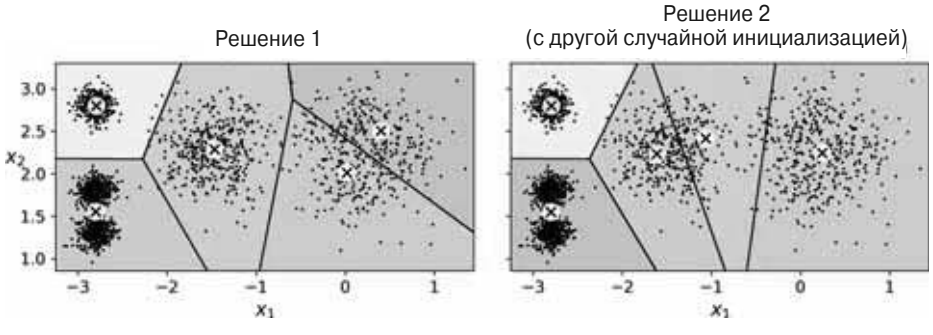

Рис. 9.5. *Субоптимальные решения из-за неудачной инициализации центроидов*

Давайте рассмотрим несколько способов уменьшения риска попадания в такую ситуацию путем совершенствования инициализации центроидов.

Методы инициализации центроидов

Если так случилось, что вы знаете, где приблизительно должны располагаться центроиды (например, по причине того, что ранее запускали еще один алгоритм кластеризации), тогда можете установить гиперпараметр `init` в массив NumPy, содержащий список центроидов, и установить `n_init` в `1`:

```
good_init = np.array([[-3, 3], [-3, 2], [-3, 1], [-1, 2], [0, 2]])
kmeans = KMeans(n_clusters=5, init=good_init, n_init=1)
```

Другое решение предусматривает многократный запуск алгоритма с разными случайными инициализациями и сохранение наилучшего решения. Количество

случайных инициализаций управляется гиперпараметром `n_init`: по умолчанию он равен `10`, т.е. при вызове `fit()` описанный ранее полный алгоритм выполняется 10 раз и библиотека Scikit-Learn сохраняет наилучшее решение. Но как в точности она узнает, какое решение является наилучшим? Библиотека использует показатель эффективности! Такой показатель называется *инерцией* модели, которая представляет собой среднеквадратическое расстояние между каждым образцом и его ближайшим центроидом. Оно составляет приблизительно 223.3 для модели слева на рис. 9.5, 237.5 для модели справа на рис. 9.5 и 211.6 для модели на рис. 9.3. Класс `KMeans` запускает алгоритм `n_init` раз и сохраняет модель с самой низкой инерцией. В данном примере будет выбрана модель на рис. 9.3 (если только крайне не повезет с `n_init` последовательными случайными инициализациями). Если вам интересно, то инерция модели доступна через переменную экземпляра `inertia_`:

```
>>> kmeans.inertia_
211.59853725816856
```

Метод `score()` возвращает отрицательную инерцию. Почему отрицательную? Потому что метод `score()` прогнозатора обязан всегда соблюдать правило "больше значит лучше" библиотеки Scikit-Learn: если один прогнозатор лучше другого, то его метод `score()` должен возвращать более высокую оценку.

```
>>> kmeans.score(X)
-211.59853725816856
```

Дэвид Артур и Сергей Васильвитский в своей статье 2006 года (`https://homl.info/37`)[3] предложили важное усовершенствование алгоритма K-Means под названием *K-Means++*. Они ввели более интеллектуальный шаг инициализации, имеющий тенденцию выбирать центроиды, которые находятся далеко друг от друга, и такое усовершенствование значительно снижает вероятность схождения алгоритма K-Means в субоптимальное решение. Авторы статьи показали, что дополнительные вычисления, требуемые на более интеллектуальном шаге инициализации, вполне того стоят, поскольку они позволяют радикально сократить количество запусков алгоритма для нахождения оптимального решения.

[3] Дэвид Артур и Сергей Васильвитский, *K-Means++: The Advantages of Careful Seeding* (K-Means++: преимущества осмотрительной инициализации), *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms* (2007 г.): с. 1027–1035.

Ниже приведен алгоритм инициализации K-Means++.

1. Взять один центроид $\mathbf{c}^{(1)}$, выбирая равномерно случайно из набора данных.
2. Взять новый центроид $\mathbf{c}^{(i)}$, выбирая образец $\mathbf{x}^{(i)}$ с вероятностью $D\left(\mathbf{x}^{(i)}\right)^2 / \sum_{j=1}^{m} D\left(\mathbf{x}^{(j)}\right)^2$, где $D(\mathbf{x}^{(i)})$ — расстояние между образцом $\mathbf{x}^{(i)}$ и ближайшим центроидом, который уже был выбран. Такое распределение вероятностей гарантирует, что в качестве центроидов с гораздо большей вероятностью будут выбираться образцы, расположенные дальше от уже выбранных центроидов.
3. Повторять предыдущий шаг до тех пор, пока не будут выбраны все k центроидов.

Класс `KMeans` применяет описанный метод инициализации по умолчанию. Если вы хотите, чтобы он применял исходный метод (т.е. случайный выбор k образцов для определения начальных центроидов), тогда можете установить гиперпараметр `init` в `"random"`. Поступать так придется редко.

Ускоренный K-Means и мини-пакетный K-Means

Чарльз Элкан в своей статье 2003 года (`https://homl.info/38`)[4] предложил еще одно важное усовершенствование алгоритма K-Means. Оно значительно ускоряет алгоритм, избегая многих ненужных расчетов расстояний. Элкан достиг такого результата за счет использования неравенства треугольника (т.е. что кратчайшим расстоянием между двумя точками всегда является прямая линия[5]), а также отслеживания нижней и верхней границ для расстояний между образцами и центроидами. Именно данный алгоритм класс `KMeans` применяет по умолчанию (вы можете принудительно использовать исходный алгоритм, установив гиперпараметр `algorithm` в `"full"`, но вряд ли это когда-либо понадобится).

Дэвид Скалли в своей статье 2010 года (`https://homl.info/39`)[6] предложил очередной важный вариант алгоритма K-Means. Вместо применения

[4] Чарльз Элкан, *Using the Triangle Inequality to Accelerate K-Means* (Использование неравенства треугольника для ускорения алгоритма K-Means), *Proceedings of the 20th International Conference on Machine Learning* (2003 г.): с. 147–153.

[5] Неравенство треугольника выглядит как AC ≤ AB + BC, где A, B и C — три точки, а AB, AC и BC — расстояния между этими точками.

[6] Дэвид Скалли, *Web-Scale K-Means Clustering* (Масштабирование кластеризации K-Means для веб-приложений), *Proceedings of the 19th International Conference on World Wide Web* (2010 г.): с. 1177–1178.

на каждой итерации полного набора данных алгоритм способен использовать мини-пакеты, лишь слегка перемещая центроиды на каждой итерации. Прием ускоряет алгоритм обычно в три или четыре раза и позволяет кластеризировать гигантские наборы данных, которые не умещаются в память. В библиотеке Scikit-Learn этот алгоритм реализован в классе `MiniBatchKMeans`. Его можно применять аналогично классу `KMeans`:

```
from sklearn.cluster import MiniBatchKMeans

minibatch_kmeans = MiniBatchKMeans(n_clusters=5)
minibatch_kmeans.fit(X)
```

Если набор данных не умещается в памяти, то простейшим вариантом будет использование класса `memmap`, как делалось для инкрементного анализа главных компонентов в главе 8. В качестве альтернативы можно передавать методу `partial_fit()` по одному мини-пакету за раз, но такой подход требует намного больше работы, потому что придется самостоятельно выполнять множество инициализаций и выбирать наилучшую из них (ищите пример в разделе "Mini-Batch K-Means" ("Мини-пакетный K-Means") тетради Jupyter для настоящей главы).

Несмотря на то что мини-пакетный алгоритм K-Means гораздо быстрее обычного алгоритма K-Means, его инерция чуть хуже, особенно при увеличении количества кластеров. Вы можете видеть это на рис. 9.6: на графике слева сравнивается инерция моделей мини-пакетного K-Means и обычного K-Means, обученных на предыдущем наборе данных с применением разного количества кластеров *k*.

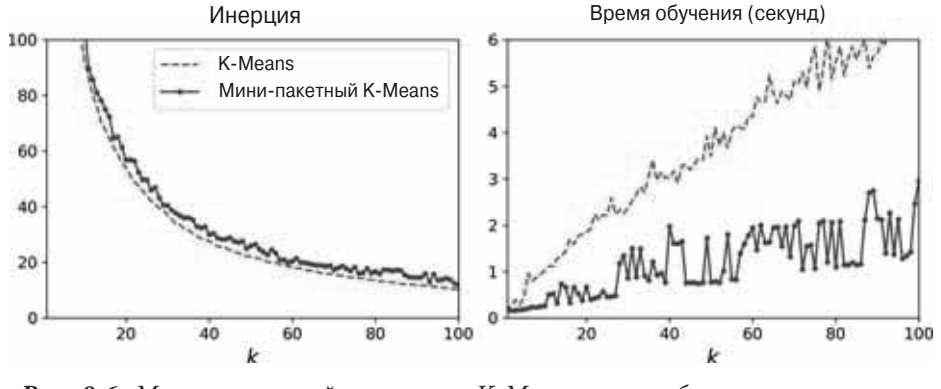

Рис. 9.6. *Мини-пакетный алгоритм K-Means имеет более высокую инерцию, чем обычный K-Means (график слева), но гораздо быстрее (график справа), особенно при увеличении k*