Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»
(ВолгГТУ)
Кафедра иностранных языков

Семестровая работа

по английскому языку

Тема: «Using the Triangle Inequality to Accelerate k-Means»

Выполнил: студент группы
САПР-2.1п Чечеткин И. А.

Проверил: доцент, к.ф.н.
Баталин С. В.

Краткая рецензия: _____

_____

_____

_____

_____

Оценка работы _____ баллов

Волгоград, 2016

Оглавление

ОРИГИНАЛ

Using the Triangle Inequality to Accelerate k-Means by Charles Elkan

ABSTRACT

The k-means algorithm is by far the most widely used method for discovering clusters in data. We show how to accelerate it dramatically, while still always computing exactly the same result as the standard algorithm. The accelerated algorithm avoids unnecessary distance calculations by applying the triangle inequality in two different ways, and by keeping track of lower and upper bounds for distances between points and centers. Experiments show that the new algorithm is effective for datasets with up to 1000 dimensions, and becomes more and more effective as the number k of clusters increases. For $k \geqslant 20$ it is many times faster than the best previously known accelerated k-means method.

1 INTRODUCTION

The most common method for finding clusters in data used in applications is the algorithm known as k-means. K-means is considered a fast method because it is not based on computing the distances between all pairs of data points. However, the algorithm is still slow in practice for large datasets. The number of distance computations is nke where n is the number of data points, k is the number of clusters to be found, and e is the number of iterations required. Empirically, e grows sublinearly with k, n, and the dimensionality d of the data.

The main contribution of this paper is an optimized version of the standard k-means method, with which the number of distance computations is in practice closer to n than to nke.

The optimized algorithm is based on the fact that most distance calculations in standard k-means are redundant. If a point is far away from a center, it is not necessary to calculate the exact distance between the point and the center in order to know that the point should not be assigned to this center. Conversely, if a point is much closer to one center than to any other, calculating exact distances is not necessary to know that the point should be assigned to the first center. We show below how to make these intuitions concrete.

We want the accelerated k-means algorithm to be usable wherever the standard algorithm is used. Therefore, we need the accelerated algorithm to satisfy three properties. First, it should be able to start with any initial centers, so that all

existing initialization methods can continue to be used. Second, given the same initial centers, it should always produce exactly the same final centers as the standard algorithm. Third, it should be able to use any black-box distance metric, so it should not rely for example on optimizations specific to Euclidean distance.

Our algorithm in fact satisfies a condition stronger than the second one above: after each iteration, it produces the same set of center locations as the standard k-means method. This stronger property means that heuristics for merging or splitting centers (and for dealing with empty clusters) can be used together with the new algorithm. The third condition is important because many applications use a domain-specific distance metric. For example, clustering to identify duplicate alphanumeric records is sometimes based on alphanumeric edit distance (Monge & Elkan, 1996), while clustering of protein structures is often based on an expensive distance function that first rotates and translates structures to superimpose them. Even without a domain-specific metric, recent work shows that using a non-traditional L_p norm with $0 < p < 1$ is beneficial when clustering in a high-dimensional space (Aggarwal et al., 2001).

2 APPLYING THE TRIANGLE INEQUALITY

Our approach to accelerating k-means is based on the triangle inequality: for any three points x, y, and z, $d(x, z) \leqslant d(x, y) + d(y, z)$. This is the only "black box" property that all distance metrics d possess.

The difficulty is that the triangle inequality gives upper bounds, but we need lower bounds to avoid calculations. Let x be a point and let b and c be centers; we need to know that $d(x, c) \geqslant d(x, b)$ in order to avoid calculating the actual value of $d(x, c)$.

The following two lemmas show how to use the triangle inequality to obtain useful lower bounds.

Lemma 1: Let x be a point and let b and c be centers. If $d(b, c) \geqslant 2d(x, b)$ then $d(x, c) \geqslant (x, b)$.

Proof: We know that $d(b, c) \leqslant d(b, x) + d(x, c)$. So $d(b, c) - d(x, b) \leqslant d(x, c)$. Consider the left-hand side: $d(b, c) - d(x, b) \geqslant 2d(x, b) - d(x, b) = d(x, b)$. So $d(x, b) \leqslant d(x, c)$.

Lemma 2: Let x be a point and let b and c be centers. Then $d(x, c) \geqslant \max\{0, d(x, b) - d(b, c)\}$.

Proof: We know that $d(x, b) \leqslant d(x, c) + d(b, c)$, so $d(x, c) \geqslant d(x, b) - d(b, c)$. Also, $d(x, c) \geqslant 0$.

Note that Lemmas 1 and 2 are true for any three points, not just for a point and two centers, and the statement of Lemma 2 can be strengthened in various ways.

We use Lemma 1 as follows. Let x be any data point, let c be the center to which is currently assigned, and let c' be any other center. The lemma says that if $\frac{1}{2}d(c, c') \geqslant d(x, c)$, then $d(x, c') \geqslant d(x, c)$. In this case, it is not necessary to calculate $d(x, c')$.

Suppose that we do not know $d(x, c)$ exactly, but we do know an upper bound u such that $u \geqslant d(x, c)$. Then we need to compute $d(x, c')$ and $d(x, c)$ only if $u > \frac{1}{2}d(c, c')$.

If $u \leqslant \frac{1}{2}\min[d(c, c')]$ where the minimum is over all $c' \neq c$, then the point x must remain assigned to the center c, and all distance calculations for x can be avoided.

Lemma 2 is applied as follows. Let x be any data point, let b be any center, and let b' be the previous version of the same center. (That is, suppose the centers are numbered 1 through k, and b is center number j; then b' is center number j in the previous iteration.) Suppose that in the previous iteration we knew a lower bound l' such that $d(x, b') \geqslant l'$. Then we can infer a lower bound l for the current iteration:

$$d(x, b) \geqslant \max\{0, d(x, b') - d(b, b')\} \geqslant \max\{0, l' - d(b, b')\} = l.$$

Informally, if l' is a good approximation to the previous distance between x and the jth center, and this center has moved only a small distance, then l is a good approximation to the updated distance.

The algorithm below is the first k-means variant that uses lower bounds, as far as we know. It is also the first algorithm that carries over varying information from one k-means iteration to the next. According to the authors of (Kanungo et al., 2000): "The most obvious source of inefficiency in [our] algorithm is that it passes no information from one stage to the next. Presumably in the later stages of Lloyd's algorithm, as the centers are converging to their final positions, one would expect that the vast majority of the data points have the same closest center from one stage to the next. A good algorithm would exploit this coherence to improve running time." The algorithm in this paper achieves this goal. One previous algorithm also reuses information from one k-means iteration in the next, but that method, due to (Judd et al., 1998), does not carry over lower or upper bounds.

Suppose $u(x) \geqslant d(x, c)$ is an upper bound on the distance between x and the center c to which x is currently assigned, and suppose $l(x, c') \leqslant d(x, c')$ is a lower bound on the distance between x and some other center c'. If $u(x) \leqslant l(x, c')$ then

$d(x,c) \leqslant u(x) \leqslant l(x,c') \leqslant d(x,c')$ so it is necessary to calculate neither $d(x,c)$ nor $d(x,c')$. Note that it will never be necessary in this iteration of the accelerated method to compute $d(x,c')$, but it may be necessary to compute $d(x,c)$ exactly because of some other center c'' for which $u(x) \leqslant l(x,c'')$ is not true.

3 THE NEW ALGORITHM

Putting the observations above together, the accelerated k-means algorithm is as follows.

First, pick initial centers. Set the lower bound $l(x,c) = 0$ for each point x and center c. Assign each x to its closest initial center $c(x) = \operatorname{argmin}_c [d(x,c)]$, using Lemma 1 to avoid redundant distance calculations. Each time $d(x,c)$ is computed, set $l(x,c) = d(x,c)$. Assign upper bounds $u(x) = \min_c [d(x,c)]$.

Next, repeat until convergence:

1. For all centers c and c', compute $d(c,c')$. For all centers c, compute $s(c) = \frac{1}{2} \min_{c' \neq c} d(c,c')$.
2. Identify all points x such that $u(x) \leqslant s(c(x))$.
3. For all remaining points x and centers c such that

   (i) $c \neq c(x)$ and
   (ii) $u(x) > l(x,c)$ and
   (iii) $u(x) > \frac{1}{2} d(c(x),c)$:

   a If $r(x)$ then compute $d(x,c(x))$ and assign $r(x) = \text{false}$. Otherwise, $d(x,c(x)) = u(x)$.
   b If $d(x,c(x)) > l(x,c)$ or $d(x,c(x)) > \frac{1}{2} d(c(x),c)$ then

   Compute $d(x,c)$
   If $d(x,c) < d(x,c(x))$ then assign $c(x) = c$.

4. For each center c, let $m(c)$ be the mean of the points assigned to c.
5. For each point x and center c, assign $l(x,c) = \max \{l(x,c) - d(c,m(c)), 0\}$.
6. For each point x, assign $u(x) = u(x) + d(m(c(x)),c(x))$ and $r(x) = \text{true}$.
7. Replace each center c by $m(c)$.

In step (3), each time $d(x,c)$ is calculated for any x and c, its lower bound is updated by assigning $l(x,c) = d(x,c)$. Similarly, $u(x)$ is updated whenever $c(x)$ is changed or $d(x,c(x))$ is computed. In step (3a), if $r(x)$ is true then $u(x)$ is out-of-date, i.e. it is possible that $u(x) \neq d(x,c(x))$. Otherwise, computing $d(x,c(x))$

is not necessary. Step (3b) repeats the checks from (ii) and (iii) in order to avoid computing $d(x, c)$ if possible.

The fundamental reason why the algorithm above is effective in reducing the number of distance calculations is that at the start of each iteration, the upper bounds $u(x)$ and the lower bounds $l(x, c)$ are tight for most points x and centers c. If these bounds are tight at the start of one iteration, the updated bounds tend to be tight at the start of the next iteration, because the location of most centers changes only slightly, and hence the bounds change only slightly.

The initialization step of the algorithm assigns each point to its closest center immediately. This requires relatively many distance calculations, but it leads to exact upper bounds $u(x)$ for all x and to exact lower bounds $l(x, c)$ for many (x, c) pairs. An alternative initialization method is to start with each point arbitrarily assigned to one center. The initial values of $u(x)$ and $l(c, x)$ are then based on distances calculated to this center only. With this approach, the initial number of distance calculations is only n, but $u(x)$ and $l(c, x)$ are much less tight initially, so more distance calculations are required later. (After each iteration each point is always assigned correctly to its closest center, regardless of how inaccurate the lower and upper bounds are at the start of the iteration.) Informal experiments suggest that both initialization methods lead to about the same total number of distance calculations.

Logically, step (2) is redundant because its effect is achieved by condition (iii). Computationally, step (2) is beneficial because if it eliminates a point x from further consideration, then comparing $u(x)$ to $l(x, c)$ for every c separately is not necessary. Condition (iii) inside step (3) is beneficial despite step (2), because $u(x)$ and $c(x)$ may change during the execution of step (3).

We have implemented the algorithm above in Matlab. When step (3) is implemented with nested loops, the outer loop can be over x or over c. For efficiency in Matlab and similar languages, the outer loop should be over c since $k \ll n$ typically, and the inner loop should be replaced by vectorized code that operates on all relevant x collectively.

Step (4) computes the new location of each cluster center c. Setting $m(c)$ to be the mean of the points assigned to c is appropriate when the distance metric in use is Euclidean distance. Otherwise, $m(c)$ may be defined differently. For example, with k-medians the new center of each cluster is a representative member of the

cluster.

4 EXPERIMENTAL RESULTS

This section reports the results of running the new algorithm on six large datasets, five of which are high-dimensional. The datasets are described in Table 1, while Table 2 gives the results.

Our experimental design is similar to the design of (Moore, 2000), which is the best recent paper on speeding up the k-means algorithm for high-dimensional data. However, there is only one dataset used in (Moore, 2000) for which the raw data are available and enough information is given to allow the dataset to be reconstructed. This dataset is called "covtype." Therefore, we also use five other publicly available datasets. None of the datasets have missing data.

In order to make our results easier to reproduce, we use a fixed initialization for each dataset X. The first center is initialized to be the mean of X. Subsequent centers are initialized according to the "furthest first" heuristic: each new center is $\text{argmax}_{x \in X} \min_{c \in C} d(x, c)$ where C is the set of initial centers chosen so far (Dasgupta, 2002).

Following the practice of past research, we measure the performance of an algorithm on a dataset as the number of distance calculations required. All algorithms that accelerate k-means incur overhead to create and update auxiliary data structures. This means that speedup compared to k-means is always less in clock time than in number of distance calculations. Our algorithm reduces the number of distance calculations so dramatically that its overhead time is often greater than the time spent on distance calculations. However, the total execution time is always much less than the time required by standard k-means. The overhead of the l and u data structures will be much smaller with a C implementation than with the Matlab implementation used for the experiments reported here. For this reason, clock times are not reported.

Perhaps the most striking observation to be made from Table 2 is that the relative advantage of the new method increases with k. The number of distance calculations grows only slowly with k and with e (the number of passes over the data, called "iterations" in Table 2). So much redundant computation is eliminated that the total number of distance calculations is closer to n than to nke as for standard k-means.

A related observation is that for $k \geqslant 20$ we obtain a much better speedup than with the anchors method (Moore, 2000). The speedups reported by Moore for the "covtype" dataset are 24.8, 11.3, and 19.0 respectively for clustering with 3, 20,

and 100 centers. The speedups we obtain are 8.60, 107, and 310. We conjecture that the improved speedup for $k \geqslant 20$ arises in part from using the actual cluster centers as adaptive "anchors," instead of using a set of anchors fixed in preprocessing. The worse speedup for $k = 3$ remains to be explained.

Another striking observation is that the new method remains effective even for data with very high dimensionality. Moore writes "If there is no underlying structure in the data (e.g. if it is uniformly distributed) there will be little or no acceleration in high dimensions no matter what we do. This gloomy view, supported by recent theoretical work in computational geometry (Indyk et al., 1999), means that we can only accelerate datasets that have interesting internal structure." While this claim is almost certainly true asymptotically as the dimension of a dataset tends to infinity, our results on the "random" dataset suggest that worthwhile speedup can still be obtained up to at least 1000 dimensions. As expected, the more clustered a dataset is, the greater the speedup obtained. Random projection makes clusters more Gaussian (Dasgupta, 2000), so speedup is better for the "mnist50" dataset than for the "mnist784" dataset.

name	cardinality	dimensionality	description
birch	100000	2	10 by 10 grid of Gaussian clusters, DS1 in (Zhang et al., 1996)
covtype	150000	54	remote soil cover measurements, after (Moore, 2000)
kddcup	95413	56	KDD Cup 1998 data, un-normalized
mnist50	60000	50	random projection of NIST handwritten digit training data
mnist784	60000	784	original NIST handwritten digit training data
random	10000	1000	uniform random data

Table 1. Datasets used in experiments.

		$k = 3$	$k = 20$	$k = 100$
birch	iterations	17	38	56
	standard	5.100e+06	7.600e+07	5.600e+08
	fast	4.495e+05	1.085e+06	1.597e+06
	speedup	11.3	70.0	351
covtype	iterations	18	256	152
	standard	8.100e+06	7.680e+08	2.280e+09
	fast	9.416e+05	7.147e+06	7.353e+06
	speedup	8.60	107	310
kddcup	iterations	34	100	325
	standard	9.732e+06	1.908e+08	3.101e+09
	fast	6.179e+05	3.812e+06	1.005e+07
	speedup	15.4	50.1	309
mnist50	iterations	38	178	217
	standard	6.840e+06	2.136e+08	1.302e+09
	fast	1.573e+06	9.353e+06	3.159e+07
	speedup	4.35	22.8	41.2
mnist784	iterations	63	60	165
	standard	1.134e+07	7.200e+07	9.900e+08
	fast	1.625e+06	7.396e+06	3.055e+07
	speedup	6.98	9.73	32.4
random	iterations	52	33	18
	standard	1.560e+06	6.600e+06	1.800e+07
	fast	1.040e+06	3.020e+06	5.348e+06
	speedup	1.50	2.19	3.37

Table 2. Rows labeled “standard” and “fast” give the number of distance calculations performed by the unaccelerated k-means algorithm and by the new algorithm. Rows labeled “speedup” show how many times faster the new algorithm is, when the unit of measurement is distance calculations.

ПЕРЕВОД

Использование неравенства треугольника для ускорения работы алгоритма k-средних. Автор: Чарльз Элкан.

РЕФЕРАТ

Алгоритм k-средних является наиболее широко используемым методом для обнаружения кластеров в данных. Мы покажем, как кардинально ускорить его работу, всегда получая при этом тот же результат, что и при использовании стандартного алгоритма. Ускоренный алгоритм избегает ненужных вычислений расстояний, применяя неравенство треугольника двумя разными способами и отслеживая нижние и верхние границы расстояний между точками и центрами кластеров. Эксперименты показывают, что новый алгоритм эффективен для наборов данных размерностью вплоть до 1000 измерений, и что он становится всё более эффективным при увеличении числа кластеров k. Для $k \geqslant 20$ новый алгоритм работает во много раз быстрее, нежели самый лучший из известных способов ускорения метода k-средних.

1 ВВЕДЕНИЕ

Наиболее распространенным методом для поиска кластеров в данных, используемых в приложениях, является метод k-средних. Метод k-средних считается быстрым, поскольку он не основан на вычислении расстояний между всеми парами точек. Тем не менее, алгоритм по-прежнему является медленным на практике работы с большими объемами данных. Количество дистанционных расчетов равно nke, где n – это число точек, k – число кластеров и e – число требуемых итераций для схождения метода. Эмпирически, число итерациий растет сублинейно с k, n и размерностью d данных.

Основным вкладом данной работы является оптимизированная версия стандартного метода k-средних, в котором число дистанционных расчетов на практике ближе к n, чем к nke.

Оптимизированный алгоритм основан на том факте, что большинство дистанционных измерений в стандартном методе k-средних является лишним. Если точка находится далеко от центра кластера, то и не нужно считать точное расстояние между точкой и центром для того, чтобы узнать, что точка не принадлежит этому кластеру. И наоборот, если точка находится гораздо ближе к одному центру, чем к любому другому, то не нужно считать точные

расстояния от точки до остальных центров, чтобы узнать, что точка принадлежит ближнему кластеру. Далее мы покажем, как сделать эти интуитивные высказывания более конкретными.

Мы хотим, чтобы ускоренным метод k-средних мог быть использован в любых приложениях, в которых может использоваться стандартный алгоритм. Таким образом, необходимо, чтобы ускоренный алгоритм удовлетворял трем свойствам. Во-первых, он должен работать с любыми начальными положениями центров, так что все существующие методы инициализации могут использоваться для работы с ускоренным алгоритмом. Во-вторых, при подаче одинаковых данных на входы нового алгоритма и стандартного алгоритма, на выходах мы должны получить одинаковые результаты работы. В-третьих, должна быть возможность использовать любую метрику расстояний, так что алгоритм не должен быть приспособлен для работы с конкретной метрикой, например, эвклидовой.

Наш алгоритм, по факту, удовлетворяет второму требованию несколько сильнее: после каждой итерации он выдает те же центры кластеров, что и стандартный алгоритм. Это свойство означает, что эвристики для слияния или разделения кластеров (и для работы с пустыми кластерами) могут использоваться вместе с новым алгоритмом. Третье условие является очень важным, поскольку множество приложений используют специфичную для конкретной предметной области метрику. Например, кластеризация для определения дубликатов буквенно-цифровых записей иногда основано на расстоянии буквенно-цифровых преобразований (Монж и Элкан, 1996), в то время как кластеризация белковых структур часто основано на сложной функции расстояния, которая сначала поворачивает и транслирует структуру молекулы, чтобы потом наложить их. Даже без специфичной метрики, недавние работы показывают, что использование нетрадиционных L_p норм с $0 < p < 1$ является полезным, когда кластеризация идет в многомерном пространстве (Аггарваль и др., 2001).

2 ПРИМЕНЕНИЕ НЕРАВЕНСТВА ТРЕУГОЛЬНИКА

Наш подход к укорению метода k-средних основан на неравенстве треугольника: для любых трех точек x, y и z длина одной стороны треугольника меньше или равна сумме двух других: $d(x, z) \leqslant d(x, y) + d(y, z)$. Это единственное свойство, которым обладают абсолютно все метрики d.

Сложность состоит в том, что неравенство треугольника дает верние границы, а для избавления от ненужных вычислений нам необходимы нижние

границы. Пусть x является точкой, а b и c – центрами; тогда нам нужно знать, что $d(x,c) \geqslant d(x,b)$, для ухода от расчета фактического значения $d(x,c)$.

Следующие две леммы покажут, как использовать неравенство треугольника для получения полезных нижних границ.

Лемма 1: Пусть x является точкой, а b и c – центрами. Если $d(b,c) \geqslant 2d(x,b)$, то $d(x,c) \geqslant (x,b)$.

Доказательство: Мы знаем, что $d(b,c) \leqslant d(b,x) + d(x,c)$. Так что $d(b,c) - d(x,b) \leqslant d(x,c)$. Применим условие леммы в левой части неравенства: $d(b,c) - d(x,b) \geqslant 2d(x,b) - d(x,b) = d(x,b)$. Таким образом, $d(x,b) \leqslant d(x,c)$, что и требовалось доказать.

Лемма 2: Пусть x является точкой, а b и c – центрами. Then $d(x,c) \geqslant \max\{0, d(x,b) - d(b,c)\}$.

Доказательство: Мы знаем, что $d(x,b) \leqslant d(x,c) + d(b,c)$, поэтому $d(x,c) \geqslant d(x,b) - d(b,c)$. Также, $d(x,c) \geqslant 0$.

Заметим, что леммы 1 и 2 справедливы для любых трех точек, а не только для точки и двух центров, и то, что утверждение леммы 2 может быть усилено различными способами.

Мы используем лемму 1 следующим образом. Пусть x является любой точкой из набора данных, пусть c является центром кластера, к которому в текущий момент принадлежит точка, и пусть c' является центром любого другого кластера. Из лемма следует, что если $\frac{1}{2}d(c,c') \geqslant d(x,c)$, то $d(x,c') \geqslant d(x,c)$. В этом случае, нам не нужно вычислять $d(x,c')$.

Предположим, что мы точно не знаем $d(x,c)$, но знаем верхнюю границу u такую, что $u \geqslant d(x,c)$. Тогда нам нужно вычислять $d(x,c')$ и $d(x,c)$ только в том случае, если $u > \frac{1}{2}d(c,c')$.

Если $u \leqslant \frac{1}{2}\min[d(c,c')]$, где минимум берется по всем центрам $c' \neq c$, то точка x должна остаться принадлежащей кластеру c, а все остальные расчеты расстояний для точки x могут быть пропущены.

Лемма 2 применяется следующим образом. Пусть x является любой точкой из набора данных, пусть b является центром любого кластера, и пусть b' является предыдущей версией центра того же кластера. (То есть, предпологая что все кластеры пронумерованы от 1 до k, а b – это кластер с номером j; то b' – это кластер номер j на предыдущей итерации алгоритма.) Предположим, что на прошлой итерации алгоритма была известна нижняя граница l' такая, что $d(x,b') \geqslant l'$. Тогда можно сделать вывод, что для нижней границы l на

текущей итерации будет справедливо неравенство:

$$d(x,b) \geqslant \max\{0, d(x,b') - d(b,b')\} \geqslant \max\{0, l' - d(b,b')\} = l.$$

Вообще, если l' является хорошим приближением к предыдущему расстоянию между x и jтым кластером, и этот центр сдвинулся только на небольшое расстояние, то l является хорошим приближением к обновленной дистанции.

Новый алгоритм является, насколько известно, первым вариантом метода k-средних, в котором используются нижние границы. Это также первый алгоритм, который переносит различную информацию от одной итерации к следующей. По мнению авторов (Канунго и др., 2000): «самый очевидный источник неэффективности [нашего] алгоритма – это то, что он не переносит информации от одного этапа к следующему. Предположительно, на поздних стадиях алгоритма Ллойда, таких как схождение центров к их конечным положениям, можно было бы ожидать, что подавляющее большинство точек имеет один и тот же ближайший центр кластера при переходе на следующую итерацию. В хорошем алгоритме эта согласованность будет использована для улучшения его времени работы.» Алгоритм в данной статье достигает этой цели. Один из известных алгоритмов также переносит информацию от одной итерации метода k-средних к следующей, но тот алгоритм, в силу (Джудд и др., 1998), не переносит нижние или верхние границы.

Предположим, что $u(x) \geqslant d(x,c)$ является верхней границей расстояния между точкой x и центром кластера c, которому точк x принадлежит на текущей итерации. Предположим, что $l(x,c') \leqslant d(x,c')$ является нижней границей расстояния между точкой x и центром какого-либо другого кластера c'. Если $u(x) \leqslant l(x,c')$, то $d(x,c) \leqslant u(x) \leqslant l(x,c') \leqslant d(x,c')$, и тогда нет необходимости вычислять ни $d(x,c)$, ни $d(x,c')$. Заметим, что на текущей итерации никогда не будет необходимости в вычислении $d(x,c')$, но может возникнуть необходимость в вычислении $d(x,c)$, но только из-за того, что для какого-то другого кластера c'' неравенство $u(x) \leqslant l(x,c'')$ не является верным.

3 НОВЫЙ АЛГОРИТМ

Сведя все приведенные выше наблюдения воедино, можно записать ускоренный алгоритм метода k-средних.

Во-первых, нуобходимо выбрать начальные центры кластеров и поставить нижние границы $l(x,c) = 0$ для каждой точки x и центра c. Присвоить x ближайшему начальному кластеру $c(x) = \mathrm{argmin}_c\,[d(x,c)]$, используя лемму

1 для избегания избыточных дистанционных вычислений. Каждый раз, когда вычисляется $d(x, c)$, устанавливать $l(x, c) = d(x, c)$. Присвоить верхним границам $u(x) = \min_c d(x, c)$.

Далее, повторять пока метод не сойдется:

1. Для всех центров c и c', вычислить $d(c, c')$. Для всех центров c, вычислить $s(c) = \frac{1}{2} \min_{c' \neq c} d(c, c')$.
2. Определить все точки x, для которых $u(x) \leqslant s(c(x))$.
3. Для всех остальных точек x и таких центров c, для которых

   (i) $c \neq c(x)$, и
   (ii) $u(x) > l(x, c)$, и
   (iii) $u(x) > \frac{1}{2} d(c(x), c)$:

   а если $r(x)$, то вычислить $d(x, c(x))$ и установить $r(x) = \text{false}$. Иначе, установить $d(x, c(x)) = u(x)$.
   б Если $d(x, c(x)) > l(x, c)$ или $d(x, c(x)) > \frac{1}{2} d(c(x), c)$, то

   вычислить $d(x, c)$;
   если $d(x, c) < d(x, c(x))$, то присвоить $c(x) = c$.

4. Для каждого центра c, рассчитать $m(c)$ как среднее всех точек, принадлежащих кластеру c.
5. Для каждой точки x и центра c, установить

$$l(x, c) = \max \{l(x, c) - d(c, m(c)), 0\} .$$

6. Для каждой точки x, установить $u(x) = u(x) + d(m(c(x)), c(x))$ и $r(x) = \text{true}$.
7. Заменить каждый центр c на $m(c)$.

На шаге (3), каждый раз когда вычисляется расстояние $d(x, c)$ для любых x и c, его нижняя граница обновляется присвоением $l(x, c) = d(x, c)$. Точно так же, $u(x)$ обновляется всякий раз, когда изменяется $c(x)$ или вычисляется $d(x, c(x))$. На шаге (3а), если $r(x)$ имеет значение true, то $u(x)$ признается устаревшим, то есть существует вероятность того, что $u(x) \neq d(x, c(x))$. В противном случае, вычисление $d(x, c(x))$ не является необходимым. Стадия (3б) повторяет проверки (ii) и (iii) для того, чтобы по возможности избегать вычислений $d(x, c)$.

Основная причина, по которой алгоритм, приведенный выше, эффективен в уменьшении числа дистанционных измерений, заключается в том, что в начале каждой итерации верхние $u(x)$ и нижние $l(x,c)$ границы жестки для большинства точек x и центров c. Если эти границы являются жесткими в начале одной итерации, обновленные границы, как правило, будут жесткими в начале следующей итерации, поскольку местоположение большинства центров меняется незначительно, а значит и границы изменятся незначительно.

Начальный этап алгоритма сразу прикрепляет каждую точку к его ближайшему кластеру. Это требует относительно много дистанционных вычислений, но это приводит к определению верхних границ $u(x)$ для всех x и определению $l(x,c)$ для большинства пар (x,c). Альтернативным способом инициализации является прикрепление точек к произвольно выбранному кластеру. Начальные значения $u(x)$ и $l(c,x)$ основаны только на расстояниях до выбранного центра. При таком подходе, изначальное число дистанционных измерений сводится к n, но $u(x)$ и $l(c,x)$ не так жестко определены, так что в дальнейшем понадобится большее число дистанционных измерений. (После каждой итерации всякая точка всегда верно прикреплена к ближайшему центру, независимо от того насколько неточны значения нижних и верхних границ в начале итерации.) Неофициальные эксперименты показывают, что оба способа инициализации ведут в среднем к одному и тому же значению количества дистанционных измерений.

Логически, шаг (2) является излишним, поскольку его эффект достигает условием (iii). С вычислительной точки зрения, шаг (2) является выгодным, поскольку если он устраняет точку x от дальнейшего рассмотрения, то сравнение $u(x)$ и $l(x,c)$ для каждого c по отдельности является ненужным. Условие (iii) внутри шага (3) полезно не смотря на шаг (2), поскольку $u(x)$ и $c(x)$ могут измениться в ходе выполнения шага (3).

Мы реализовали алгоритм, представленный выше, в среде Matlab. Когда шаг (3) реализован с использованием вложенных циклов, внешний цикл может быть или по x, или по c. Для повышения эффективности в Matlab и похожих языках, внешний цикл должен быть по c, поскольку обычно $k \ll n$, а внутренний цикл должен быть заменен векторизированным кодом, который работает над всеми соответствующими x совместно.

Шаг (4) вычисляет новое положение центра каждого кластера c. Расчет $m(c)$ как среднего всех точек, принадлежащих c используется тогда, когда в качестве метрики используется евклидово расстояние. В ином случае, $m(c)$

может рассчитываться по-другому. Например, в методе k-медиан новые центры кластеров являются характерными представителями точек, принадлежащих кластеру.

4 ЭКСПЕРИМЕНТАЛЬНЫЕ ДАННЫЕ

В этом разделе представлены результаты работы нового алгоритма на шести больших наборах данных, пять из которых являются высокоразмерными. Наборы данных описаны в таблице 1, а результаты работы алгоритма приведены в таблице 2.

Построение нашего эксперимента схоже с построением эксперимента в статье (Мур, 2000), которая является лучшей работой последнего времени по ускорению работы метода k-средних на высокоразмерных данных. Тем не менее, в (Мур, 2000) использован только один набор данных, необработанные данные которого находятся в открытом доступе, и о котором есть достаточно информации для построения полной картины данных. Этот набор данных назван «covtype». Таким образом, мы также используем еще пять публично доступных наборов данных. Ни один из этих наборов данных не имеет каких-либо недостающих данных.

Для того, чтобы облегчить воспроизводимость наших результатов, мы использовали фиксированную инициализацию для каждого набора данных X. Первый центр инициализируется как среднее значение точек в наборе X. Последующие центры инициализируются в соответствии с эвристикой «дальше от первого»: каждый новый центр рассчитывается как $\operatorname{argmax}_{x \in X} \min_{c \in C} d(x, c)$, где C – это набор полученных на текущий момент центров (Дасгупта, 2002).

Следуя практике прошлых исследований, мы измеряем производительность алгоритма на наборе данных как количество необходимых дистанционных измерений. Все алгоритмы, ускоряющие работу k-средних несут дополнительные расходы на создание и обновление вспомогательных структур данных. Это означает, что ускорение по тактам вычислений всегда меньше, чем по количеству расчетов. Наш алгоритм уменьшает количество дистанционных измерений настолько сильно, что часто его служебное время работы больше, чем время, затраченное на измерения. Тем не менее, общее время выполнения всегда гораздо меньше, чем время, затрачиваемое стандартным методом k-средних. Накладные расходы на создание структур данных l и u будут гораздо меньше при реализации алгоритма на C, чем на Matlab, который использовался для проведения экспериментов, представленных здесь. По этой

причине, время работы в тактах здесь не сообщается.

Пожалуй, наиболее поразительное наблюдение, сделанное из таблицы 2, состоит в том, что эффективность работы нового алгоритма увеличивается с увеличением k. Число дистанционных измерений растет только с k и e (количеств проходов по данным, называемых «итерациями» в таблице 2). Убирается настолько много излишних вычислений, что общее количество измерений становится ближе к n, чем к nke для стандартного метода k-средних.

Относительное наблюдение состоит в том, что для $k \geqslant 20$ мы получаем гозадо лучшее ускорение, чем с якорным методом, описанном в (Мур, 2000). Ускорения, полученные Муром для набора данных «covtype»: 24.8, 11.3, и 19.0 для кластеризации соответственно 3, 20 и 100 кластеров. Ускорения, полученные нами: 8.60, 107 и 310. Мы предполагаем, что увеличение ускорения при $k \geqslant 20$ возникает отчасти от использования фактических центров кластеров как адаптивных «якорей», вместо того, чтобы использовать набор закрепленных якорей при предварительной обработке. Ухудшение ускорения при $k = 3$ остается необъясненным.

Еще одним ярким наблюдением является эффективность нового метода даже для данных с высокой размерностью. Мур писал: «если в данных нет базовой структуры (например, они равномерно распределены), то при рассмотрении данных высоких размерностей ускорение будет либо малым, либо отсутствовать вовсе. Этот мрачный взгляд, поддержанный в недавней теоретической работе (Индик и др., 1999), означает, что мы можем ускорить обработку данных, имеющих интересную внутреннюю структуру». В то время, как это утверждение почти наверняка верно для данных, размерность которых асимптотически приближается к бесконечности, наши результаты по набору данных «random» показывают, что, по крайней мере при 1000-й размерности данных, ускорение всё еще может быть получено. Как ожидалось, что чем больше кластеров в данных, тем больше получаемое ускорение. Случайная проекция делает кластеры более гауссиановыми (Дасгупта, 2000), так что ускорение лучше для набора данных «mnist50», чем для «mnist784».

название	количество	размерность	описание
birch	100000	2	решетка 10 на 10 гауссиановых кластеров, DS1 в (Чжан и др., 1996)
covtype	150000	54	измерения удаленного почвенного покрова (Мур, 2000)
kddcup	95413	56	ненормализованные данные соревнования KDD Cup 1998
mnist50	60000	50	случайная проекция данных цифрового обучения НИСТ
mnist784	60000	784	оригинальные рукописные данные цифрового обучения НИСТ
random	10000	1000	равномерные случайные данные

Таблица 1. Наборы данных, используемые в экспериментах.

		$k = 3$	$k = 20$	$k = 100$
birch	итераций	17	38	56
	стандартный	5.100e+06	7.600e+07	5.600e+08
	быстрый	4.495e+05	1.085e+06	1.597e+06
	ускорение	11.3	70.0	351
covtype	итераций	18	256	152
	стандартный	8.100e+06	7.680e+08	2.280e+09
	быстрый	9.416e+05	7.147e+06	7.353e+06
	ускорение	8.60	107	310
kddcup	итераций	34	100	325
	стандартный	9.732e+06	1.908e+08	3.101e+09
	быстрый	6.179e+05	3.812e+06	1.005e+07
	ускорение	15.4	50.1	309
mnist50	итераций	38	178	217
	стандартный	6.840e+06	2.136e+08	1.302e+09
	быстрый	1.573e+06	9.353e+06	3.159e+07
	ускорение	4.35	22.8	41.2
mnist784	итераций	63	60	165
	стандартный	1.134e+07	7.200e+07	9.900e+08
	быстрый	1.625e+06	7.396e+06	3.055e+07
	ускорение	6.98	9.73	32.4
random	итераций	52	33	18
	стандартный	1.560e+06	6.600e+06	1.800e+07
	быстрый	1.040e+06	3.020e+06	5.348e+06
	ускорение	1.50	2.19	3.37

Таблица 2. Строки “стандартный” and “быстрый” отображают количество дистанционных измерений совершенных соответственно стандартным методом k-средних и новым алгоритмом. Строки “ускорение” показывают во сколько раз быстрее работает новый алгоритм, при условии что единицей измерения является количество дистанционных измерений.

СЛОВА

Acceleration	Ускорение
According	В соответствии
Achieve	Достигать
Algorithm	Алгоритм
Apply	Применить
Approach	Подход
Approximation	Приближение
Assign	Назначение
Avoid	Избегание
Beneficial	Выгодный
Bound	Граница
Calculate	Вычислить
Carry	Перенос
Case	Случай
Change	Менять
Check	Проверять
Cluster	Кластер
Coherence	Согласованность
Collectively	Коллективно
Common	Общий
Computation	Вычисление
Condition	Условие
Consider	Рассматривать
Continue	Продолжать
Contribution	Вклад
Converge	Сходиться
Currently	В настоящее время
Dataset	Набор данных
Define	Определять
Design	Проектировать
Despite	Несмотря
Dimension	Измерение

Distance	Расстояние
Domain-specific	Предметно-ориентированный
Dramatically	Кардинально
Exactly	Точно
Expect	Ожидать
Exploit	Использовать
Follow	Следовать
Fundamental	Фундаментальный
Hence	Следовательно
Immediately	Немедленно
Implementation	Реализация
Improve	Улучшать
Inequality	Неравенство
Infer	Делать вывод
Informally	Неформально
Initialization	Инициализация
Iteration	Итерация
Limit	Предел
Linear	Линейно
Loop	Цикл
Majority	Большинство
Mean	Средний
Method	Метод
Metric	Метрика
Necessary	Необходимый
Nested	Вложенный
Note	Замечать
Number	Нумеровать
Observation	Наблюдение
Obtain	Получать
Obvious	Очевидный
Operate	Управлять
Order	Порядок
Otherwise	Иначе

Perhaps	Возможно
Possess	Обладать
Possible	Возможный
Practice	Практика
Previously	Ранее
Produce	Производить
Proof	Доказательство
Property	Свойство
Protein	Белок
Reason	Причина
Recent	Недавний
Redundant	Излишний
Relative	Относительный
Relevant	Соответствующий
Remain	Оставаться
Replace	Замещать
Representative	Представительный
Require	Требовать
Similarly	Подобным образом
Stage	Стадия
Superimpose	Накладывать
Suppose	Предполагать
Tend	Иметь тенденцию
Tight	Туго
Track	След
Triangle	Треугольник
Unnecessary	Ненужный
Until	До тех пор пока
Update	Обновить
Usable	Удобный
Vast	Подавляющий
Version	Версия
Wherever	Где бы ни
While	В то время как