Clustering

Advanced Concepts

K-Means recap

In general, can be more abstract spaces such as space of trees, graphs or functions

- Given a dataset, $\mathcal{X} \subseteq \mathbb{R}^d$ and number of clusters k, find a clustering $C \subseteq \mathbb{R}^d$ such that the Sum Square Distance (aka potential) is minimized.

Sum Square Distance

$$\varphi(C) = \sum_{x \in \mathcal{X}} min_{c \epsilon C} d(x, c)^2$$

Loyd's Algorithm

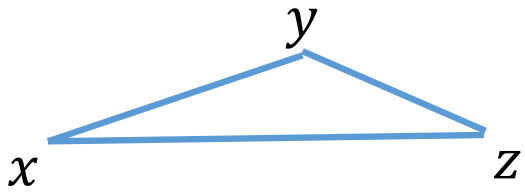
- Start with random assignments of k centroids
- Iteratively,
  - Assign each point $x \in \mathcal{X}$ to the closest center $c \epsilon C$
  - Recompute the centroids based on the cluster assignment.

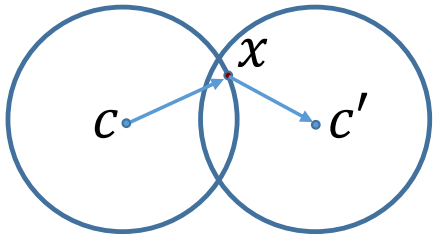
$n = |\mathcal{X}|$

$O(nkd)$

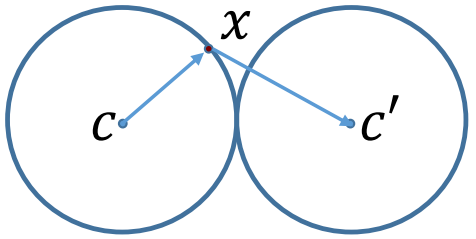
$O(nkd)$ is prohibitive for large dimension: Exploit the Triangle Inequality

$d(x,z) \leq d(x,y) + d(y,z)$





If $\mathrm{d}(x,c) \geq \frac{d(c,c')}{2}$
then $\mathrm{d}(x,c) \leq d(x,c')$
not guaranteed



If $\mathrm{d}(x,c) \leq \frac{d(c,c')}{2}$ then
$\mathrm{d}(x,c) \leq d(x,c')$ is guaranteed

Upper bound for $d(x,c)$

Also true when u $\leq \frac{d(c,c')}{2}$

Bounding the distance of x from center after the center moves from c to c'

Lower bound

$$d(x,c') \geq \max\big(0, d(x,c) - d(c,c')\big)$$
$$\geq \max\big(0, l - d(c,c')\big)$$
$$= l'$$

Lower bound for $d(x,c)$

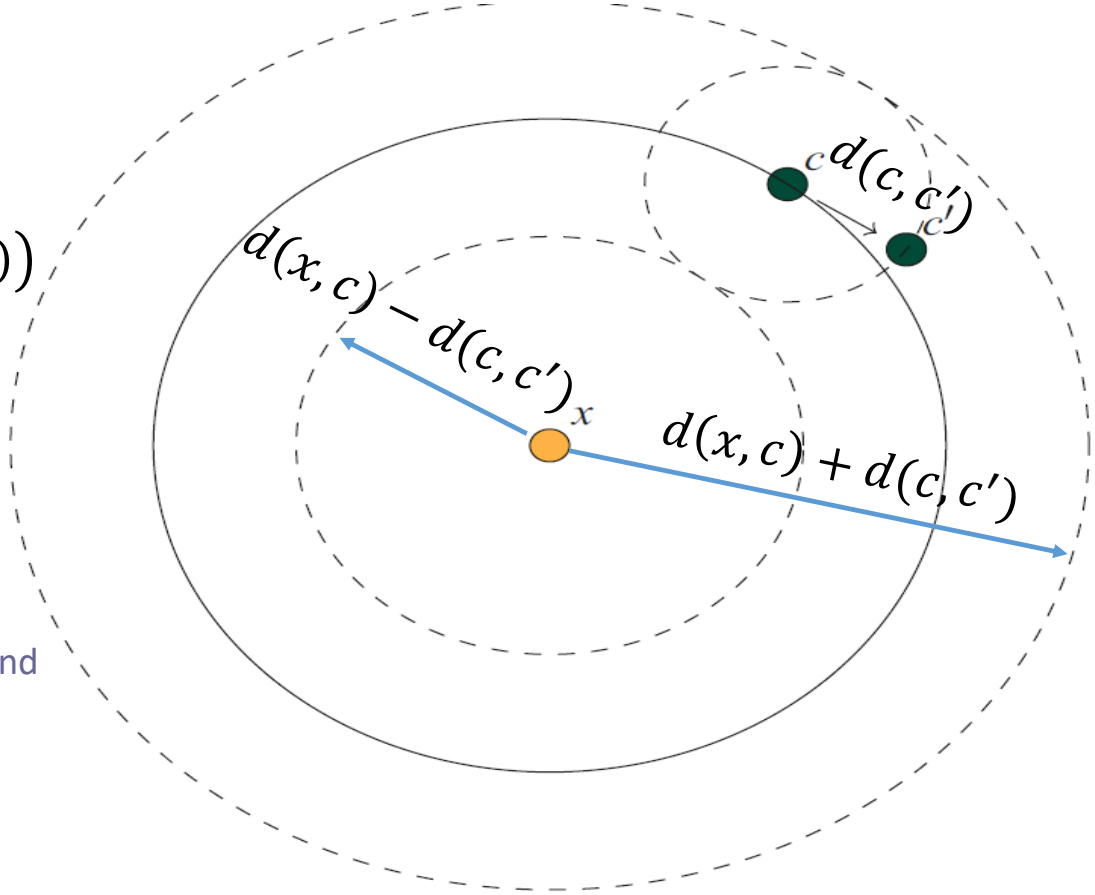
Lower bound for $d(x,c')$

Upper bound

$$d(x,c') \leq d(x,c) + d(c,c')$$
$$\leq u + d(c,c')$$
$$= u'$$

Upper bound for $d(x,c)$

Upper bound for $d(x,c')$



Elkan's accelerated K-means

Requires computation of pairwise distances between the centroids beforehand
$O(k^2)$ distances computes

Pruning principle:

i^{th} point is assigned to the right cluster if $u(i) \leq s\big(a(i)\big)$. No distance involving the i^{th} point needs to be computed.

i^{th} point cannot be assigned to j^{th} cluster if $u(i) \leq l(i,j)$ or $u(i) \leq$ half the distance between $c(a(i))$ and $c(j)$. The distance between the i^{th} point and the j^{th} centroid need not be computed.

$a(i)$: index of the cluster assigned to the i^{th} point.

$l(i,j)$: lower bound of the distance of the i^{th} point to the j^{th} cluster centroid, matrix ($n \times k$ dimensional).

$u(i)$: upper bound of distance of the i^{th} point to it's closest cluster centroid, vector (n dimensional).

$c(j)$: j^{th} cluster centroid,

$s(j)$: half the distance between j^{th} centroid and its closest centroid

Algorithm 3 Elkan's algorithm—using k lower bounds per point and k^2 center-center distances

procedure ELKAN(X, C)
  $a(i) \leftarrow 1, u(i) \leftarrow \infty, \forall i \in N$ {Initialize invalid bounds, all in one cluster.}
  $\ell(i, j) \leftarrow 0, \forall i \in N, j \in K$
  while not converged **do**
    compute $\|c(j) - c(j')\|, \forall j, j' \in K$
    compute $s(j) \leftarrow \min_{j' \neq j} \|c(j) - c(j')\|/2, \forall j \in K$
    for all $i \in N$ **do**
      if $u(i) \leq s(a(i))$ **then** continue with next i
      $r \leftarrow$ True
      for all $j \in K$ **do**
        $z \leftarrow \max(\ell(i, j), \|c(a(i)) - c(j)\|/2)$
        if $j = a(i)$ or $u(i) \leq z$ **then** continue with next j
        if r **then**
          $u(i) \leftarrow \|x(i) - c(a(i))\|$
          $r \leftarrow$ False
          if $u(i) \leq z$ **then** continue with next j
        $\ell(i, j) \leftarrow \|x(i) - c(j)\|$
        if $\ell(i, j) < u(i)$ **then** $a(i) \leftarrow j$
    for all $j \in K$ **do** {Move the centers and track their movement}
      move $c(j)$ to its new location
      let $\delta(j)$ be the distance moved by $c(j)$
    for all $i \in N$ **do** {Update the upper and lower distance bounds}
      $u(i) \leftarrow u(i) + \delta(a(i))$
      for all $j \in K$ **do**
        $\ell(i, j) \leftarrow \ell(i, j) - \delta(j)$ $\max(0, l(i,j) - \delta(j))$

Limitations of Elkan

e is the number of iterations

- Updating the l matrix takes $O(nke)$, even though time spent computing distances is reduced from to $O(nd)$ from $O(nkde)$ empirically (not in worst case);
- Storing the l matrix ($n \times k$ dimension) can be a bottleneck for large k.
- Each iteration spents $O(k^2 d)$ time computing between centroid distances.
- Since Elkan goes over the entire dataset to compute new centers, it requires $O(nd)$ time per iteration doing that.

Results for Elkan

		$k = 3$	$k = 20$	$k = 100$
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

name	cardinality	dimensionality	description
birch	100000	2	10 by 10 grid of Gaussian
covtype	150000	54	remote soil cover measure
kddcup	95413	56	KDD Cup 1998 data, un-n
mnist50	60000	50	random projection of NIST
mnist784	60000	784	original NIST handwritten
random	10000	1000	uniform random data

Results for Elkan

name	cardinality	dimensionality	description
birch	100000	2	10 by 10 grid of Gaussian c
covtype	150000	54	remote soil cover measuren
kddcup	95413	56	KDD Cup 1998 data, un-nc
mnist50	60000	50	random projection of NIST
mnist784	60000	784	original NIST handwritten
random	10000	1000	uniform random data

		$k = 3$	$k = 20$	$k = 100$
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

Hamerly's accelerated K-means

Main difference from Elkan:
$l(i)$ instead of $l(i, j)$.

Maintains one lower bound per point instead of k.

$l(i)$: lower bound of the distance of the i^{th} point to the second closest centroid

Pruning principle:

i^{th} point is assigned to the right cluster if $u(i) \leq s\big(a(i)\big)$ or $u(i) \leq l(i)$. No distance involving the i^{th} point needs to be computed.

Tradeoff

$O(n)$ instead of $O(n \times k)$

- Less memory for storing lower bounds.
- Fewer computations for updating lower bounds.
- However, there is less pruning and consequently more distance computation.

Algorithm 4 Hamerly's algorithm—using 1 lower bound per point

procedure HAMERLY(X, C)
  $a(i) \leftarrow 1, u(i) \leftarrow \infty, \ell(i) \leftarrow 0, \forall i \in N$ {Initialize invalid bounds, all in one cluster.}
  while not converged **do**
    compute $s(j) \leftarrow \min_{j' \neq j} \|c(j) - c(j')\|/2, \forall j \in K$
    for all $i \in N$ **do**
      $z \leftarrow \max(\ell(i), s(a(i)))$
      if $u(i) \leq z$ **then** continue with next i
      $u(i) \leftarrow \|x(i) - c(a(i))\|$ {Tighten the upper bound}
      if $u(i) \leq z$ **then** continue with next i
      Find $c(j)$ and $c(j')$, the two closest centers to $x(i)$, as well as the distances to each.
      if $j \neq a(i)$ **then**
        $a(i) \leftarrow j$
        $u(i) \leftarrow \|x(i) - c(a(i))\|$
      $\ell(i) \leftarrow \|x(i) - c(j')\|$
    for all $j \in K$ **do** {Move the centers and track their movement}
      move $c(j)$ to its new location
      let $\delta(j)$ be the distance moved by $c(j)$
    $\delta' \leftarrow \max_{j \in K} \delta(j)$
    for all $i \in N$ **do** {Update the upper and lower distance bounds}
      $u(i) \leftarrow u(i) + \delta(a(i))$
      $\ell(i) \leftarrow \ell(i) - \delta'$ $\max(0, l(i,j) - \delta')$

Table 1: This table gives the overhead (in time and memory) for each examined algorithm. Each entry represents the asymptotic overhead spent by that algorithm *beyond* Lloyd's algorithm. The initialization time (column 2) is extra time needed to allocate memory and create data structures. Time/iteration is the extra time spent during each k-means iteration, and memory accounts for all extra memory used.

This is worst case, actual performance is better because of pruning

	init. time	time/iteration	memory
k-d tree	$nd + n\log(n)$	-	nd
elkan	$ndk + dk^2$	dk^2	$nk + k^2$
hamerly	ndk	dk^2	n

Some considerations

- Effect of data distribution
  - More clustered data, more pruning
  - More uniform data, less pruning

Dataset		Total user CPU Seconds (User CPU seconds per iteration)							
		$k = 3$		$k = 20$		$k = 100$		$k = 500$	
uniform random	iterations	44		227		298		710	
$n = 1250000$	lloyd	4.0	(0.058)	61.4	(0.264)	320.2	(1.070)	3486.9	(4.909)
$d = 2$	kd-tree	3.5	(**0.006**)	**11.8**	(**0.035**)	34.6	(0.102)	338.8	(0.471)
	elkan	7.2	(0.133)	75.2	(0.325)	353.1	(1.180)	2771.8	(3.902)
	hamerly	**2.7**	(0.031)	14.6	(0.058)	**28.2**	(**0.090**)	**204.2**	(**0.286**)
uniform random	iterations	121		353		312		1405	
$n = 1250000$	lloyd	21.8	(0.134)	178.9	(0.491)	660.7	(2.100)	13854.4	(9.857)
$d = 8$	kd-tree	117.5	(0.886)	622.6	(1.740)	2390.8	(7.633)	46731.5	(33.254)
	elkan	14.1	(0.071)	130.6	(0.354)	591.8	(1.879)	11827.9	(8.414)
	hamerly	**10.9**	(**0.045**)	**40.4**	(**0.099**)	**169.8**	(**0.527**)	**1395.6**	(**0.989**)
uniform random	iterations	137		4120		2096		2408	
$n = 1250000$	lloyd	66.4	(0.323)	5479.5	(1.325)	12543.8	(5.974)	68967.3	(28.632)
$d = 32$	kd-tree	208.4	(1.324)	29719.6	(7.207)	74181.3	(35.380)	425513.0	(176.697)
	elkan	48.1	(0.189)	1370.1	(0.327)	2624.9	(1.242)	14245.9	(5.907)
	hamerly	**46.9**	(**0.180**)	**446.4**	(**0.103**)	**1238.9**	(**0.581**)	**9886.9**	(**4.097**)
birch	iterations	52		179		110		99	
$n = 100000$	lloyd	0.53	(0.004)	4.60	(0.024)	11.80	(0.104)	48.87	(0.490)
$d = 2$	kd-tree	**0.41**	(<**0.001**)	0.96	(**0.003**)	2.67	(0.021)	17.68	(0.173)
	elkan	0.58	(0.005)	4.35	(0.023)	11.80	(0.104)	54.28	(0.545)
	hamerly	0.44	(0.002)	**0.90**	(**0.003**)	**1.86**	(**0.014**)	**7.81**	(**0.075**)
covtype	iterations	19		204		320		111	
$n = 150000$	lloyd	3.52	(0.048)	48.02	(0.222)	322.25	(0.999)	564.05	(5.058)
$d = 54$	kd-tree	6.65	(0.205)	266.65	(1.293)	2014.03	(6.285)	3303.27	(29.734)
	elkan	3.07	(0.022)	11.58	(0.044)	70.45	(0.212)	**152.15**	(**1.347**)
	hamerly	**2.95**	(**0.019**)	**7.40**	(**0.024**)	**42.83**	(**0.126**)	169.53	(1.505)
kddcup	iterations	39		55		169		142	
$n = 95412$	lloyd	4.74	(0.032)	12.35	(0.159)	116.63	(0.669)	464.22	(3.244)
$d = 56$	kd-tree	9.68	(0.156)	58.55	(0.996)	839.31	(4.945)	3349.47	(23.562)
	elkan	4.13	(0.012)	6.24	(0.049)	32.27	(0.169)	**132.39**	(**0.907**)
	hamerly	**3.95**	(**0.011**)	**5.87**	(**0.042**)	**28.39**	(**0.147**)	197.26	(1.364)
mnist50	iterations	37		249		190		81	
$n = 60000$	lloyd	2.92	(0.018)	23.18	(0.084)	75.82	(0.387)	162.09	(1.974)
$d = 50$	kd-tree	4.90	(0.069)	100.09	(0.393)	371.57	(1.943)	794.51	(9.780)
	elkan	2.42	(0.005)	7.02	(0.019)	**21.58**	(**0.101**)	**55.61**	(**0.660**)
	hamerly	**2.41**	(**0.004**)	**4.54**	(**0.009**)	21.95	(0.104)	77.34	(0.928)

Table 3: These results show the fraction of times that each algorithm was able to skip the innermost loop on data of different dimensions (values closer to 1 are better). These results are averaged over runs using k = 3, 20, 100, and 500 (one run for each k). The randX datasets are uniform random hypercube data with X dimensions.

dataset	rand2	rand8	rand32	rand128
elkan	0.56	0.01	0.00	0.00
hamerly	0.97	0.88	0.91	0.83

dataset	birch	covtype	kddcup	mnist50
elkan	0.52	0.34	0.18	0.22
hamerly	0.94	0.89	0.82	0.82

Memory requirements

		Megabytes			
Dataset	Algorithm	k=3	k=20	k=100	k=500
uniform random n=1.25M d=2	lloyd	7.5	7.5	7.5	7.5
	kd-tree	32.1	32.1	32.1	32.1
	elkan	19.8	60.3	251.0	1205.2
	hamerly	14.7	14.7	14.7	14.7
uniform random n=1.25M d=8	lloyd	21.9	21.9	21.9	21.9
	kd-tree	54.8	54.8	54.8	54.8
	elkan	34.1	74.6	265.3	1219.5
	hamerly	29.0	29.0	29.0	29.0
uniform random n=1.25M d=32	lloyd	79.1	79.1	79.1	79.1
	kd-tree	145.2	145.2	145.2	145.3
	elkan	91.3	131.8	322.6	1276.8
	hamerly	86.2	86.2	86.2	86.3
birch n=100K d=2	lloyd	1.4	1.1	1.1	1.3
	kd-tree	2.9	2.9	2.8	2.7
	elkan	2.1	5.2	20.6	97.3
	hamerly	1.5	1.7	1.6	1.5
covtype n=150K d=54	lloyd	16.2	16.2	16.1	16.4
	kd-tree	27.2	27.2	27.2	27.3
	elkan	17.4	22.5	45.3	160.4
	hamerly	17.0	17.0	16.8	17.2
kddcup n=95412 d=56	lloyd	10.9	10.8	11.1	11.2
	kd-tree	18.8	18.9	19.1	19.0
	elkan	11.9	15.1	29.6	103.1
	hamerly	11.6	11.6	11.3	11.7
mnist50 n=60K d=50	lloyd	6.3	6.6	6.4	6.8
	kd-tree	10.5	10.4	10.6	10.7
	elkan	7.0	9.1	18.4	64.8
	hamerly	6.9	6.9	6.9	6.8

Summary

- For moderate k (around 100), Hamerly is well-suited (has smaller time and memory footprint).
- Large k (greater than 100), Elkan might be better (has smaller time footprint, in spite of large memory requirements).

Picking the initialization cluster centers: a significant issue

$\varphi = \varphi(C)$ where C is the set of centroids that K-means converges to

φ_{opt} is the minimum value $\varphi(C)$ can attain

- It is the speed and simplicity of the k-means method that make it appealing, not its accuracy. Indeed, there are many natural examples for which the algorithm generates arbitrarily bad clustering (i.e., $\frac{\varphi}{\varphi_{opt}}$ is unbounded even when n and k are fixed). This does not rely on an adversarial placement of the starting centers, and in particular, it can hold with high probability even if the centers are chosen uniformly at random from the data points.

Sum square distance

$$\varphi(C) = \sum_{x \in X} min_{c \epsilon C} d(x, c)^2$$

Furthest first

- Pick first center to be the mean of the data
- For the subsequent centers iteratively pick the point whose distance to its closest cluster is largest.

$$c_{j+1} \leftarrow argmax_{x \in \mathcal{X}} \; min_{c \epsilon C_j} d(x, c)$$
$$C_{j+1} \leftarrow C_j \cup \{c_{j+1}\}$$

Problem: Outliers get chosen as centers.

K-Means ++

1a. Take one center c_1, chosen uniformly at random from $\mathcal{X}$.

1b. Take a new center c_i, choosing $x \in \mathcal{X}$ with probability $\frac{D(x)^2}{\sum_{x \in \mathcal{X}} D(x)^2}$.

1c. Repeat Step 1b. until we have taken k centers altogether.

2-4. Proceed as with the standard `k-means` algorithm.

Theorem 3.1. *If $\mathcal{C}$ is constructed with* `k-means++`*, then the corresponding potential function ϕ satisfies, $E[\phi] \leq 8(\ln k + 2)\phi_{\text{OPT}}$.*

k	Average ϕ		Minimum ϕ		Average T	
	k-means	k-means++	k-means	k-means++	k-means	k-means++
10	135512	126433	119201	111611	0.14	0.13
25	48050.5	15.8313	25734.6	15.8313	1.69	0.26
50	5466.02	14.76	14.79	14.73	3.79	4.21

Table 2: Experimental results on the *Norm-25* dataset (n = 10000, d = 15)

k	Average ϕ		Minimum ϕ		Average T	
	k-means	k-means++	k-means	k-means++	k-means	k-means++
10	7553.5	6151.2	6139.45	5631.99	0.12	0.05
25	3626.1	2064.9	2568.2	1988.76	0.19	0.09
50	2004.2	1133.7	1344	1088	0.27	0.17

Table 3: Experimental results on the *Cloud* dataset (n = 1024, d = 10)

k	Average ϕ		Minimum ϕ		Average T	
	k-means	k-means++	k-means	k-means++	k-means	k-means++
10	$3.45 \cdot 10^8$	$2.31 \cdot 10^7$	$3.25 \cdot 10^8$	$1.79 \cdot 10^7$	107.5	64.04
25	$3.15 \cdot 10^8$	$2.53 \cdot 10^6$	$3.1 \cdot 10^8$	$2.06 \cdot 10^6$	421.5	313.65
50	$3.08 \cdot 10^8$	$4.67 \cdot 10^5$	$3.08 \cdot 10^8$	$3.98 \cdot 10^5$	766.2	282.9

Table 4: Experimental results on the *Intrusion* dataset (n = 494019, d = 35)

k-means via specialized datastructures
k-d Tree