K-Means clustering accelerated algorithms using the triangle inequality

Alejandra Ornelas Barajas
COMP 5703 Advanced Algorithms

December 2015

1 Introduction

The k-means clustering algorithm is a very popular tool for data analysis chosen as one of the top ten data mining algorithms. The idea of the k-means optimization problem is that it seeks to partition n data points into k clusters while minimizing the distance (square distance) between each data point and the center of the cluster it belongs to [1].

Lloyd's algorithm is the standard approach for this problem [4]. However, it spends a lot of processing time computing the distances between each of the k cluster centers and the n data points. Since points usually stay in the same clusters after a few iterations, much of this work is unnecessary, making the naive implementation very inefficient (see figure 1 for pseudo-code) [6].

The total running time of Lloyd's algorithm is *O(wnkd)* for w iterations, k centers, and n points in d dimensions. In this paper we are going to present some optimizations to speed up Lloyd's algorithm using the triangle inequality.

Algorithm Lloyd's k-means algorithm—the standard algorithm for minimizing $J(X, C)$.

```
procedure LLOYD(X, C)
  while not converged do
    for all i ∈ N do {Find the closest center to each x(i).}
      a(i) ← 1
      for all j ∈ K do
        if ‖x(i) − c(j)‖ < ‖x(i) − c(a(i))‖ then
          a(i) ← j
    for all j ∈ K do {Move the centers}
      move c(j) to the mean of {x(i)|a(i) = j}
```

Figure 1: Algorithm 1

2 The Triangle Inequality in *k*-means

The triangle inequality is a very powerful tool from geometry that is applicable in the k-means algorithm in multiple ways. Phillips demonstrated how to use it to accelerate the algorithm [5].

Claim: If center c' is close to point x, and some other center c is far away from another center c', then c' must be closer than c to x.

Proof

- Consider the already computed distances $\|x - c'\|$ and $\|c - c'\|$

- By the triangle inequality we know that:

$$\|c - c'\| \leq \|x - c\| + \|x - c'\|$$
$$\|c - c'\| - \|x - c'\| \leq \|x - c\|$$

- And we also know that $2\|x - c'\| \leq \|c - c'\|$, so:

$$2\|x - c'\| - \|x - c'\| \leq \|x - c\|$$
$$\|x - c'\| \leq \|x - c\|$$

Which proves that c is not closer than c' to x without measuring the distance $\|x - c\|$

Corollary: Some point-center distances of the k-means algorithm do not need to be computed.

2.1 Maintaining Distance Bounds with the Triangle Inequality

We can use the triangle inequality to cheaply maintain upper and lower bounds on the distance between two points x and c' (where c' is the new location of the center c). Assuming that the distance $\|x - c\|$ has been calculated. After c moves to c', we measure $\|c - c'\|$. The upper and lower bounds on $\|x - c'\|$ are given by $\|x - c\| - \|c - c'\| \leq \|x - c'\| \leq \|x - c\| + \|c - c'\|$. Figure 2 illustrates these bounds by the two dashed circles center on x. Thus, c' must be inside the region bounded by these two circles[6].

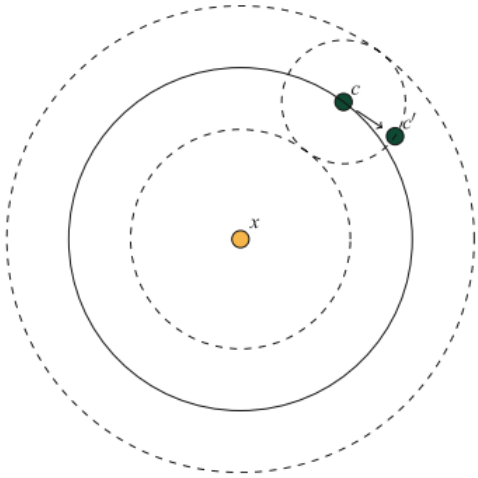

Figure 2: Upper and lower bounds

3 Accelerated Algorithms

In this section we will present three algorithms that use upper and lower bounds in order to accelerate Lloyd's algorithm.

3.1 Elkan's Algorithm

Elkan introduced an algorithm that uses one upper bound and k lower bounds for each clustered point $x(i)$ [2]. The upper bound is given by $u(i) \geq \|x(i) - c(a(i))\|$, where $\|x(i) - c(a(i))\|$ is the distance between $x(i)$ and its closest center $c(a(i))$. And each lower bounds are $l(i,j) \leq \|x(i) - c(j)\|$ where $\|x(i) - c(j)\|$ is the distance between *x(i)* and the center *c(j)*.The bounds may be efficiently updated by adding/subtracting the distance moved by each center after each k-means iteration. Figure 3 shows the pseudo-code for this algorithm.

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

```
procedure ELKAN(X, C)
    a(i) ← 1, u(i) ← ∞, ∀i ∈ N {Initialize invalid bounds, all in one cluster.}
    ℓ(i, j) ← 0, ∀i ∈ N, j ∈ K
    while not converged do
5:      compute ‖c(j) − c(j′)‖, ∀j, j′ ∈ K
        compute s(j) ← min_{j′≠j} ‖c(j) − c(j′)‖/2, ∀j ∈ K
        for all i ∈ N do
            if u(i) ≤ s(a(i)) then continue with next i
            r ← True
10:         for all j ∈ K do
                z ← max(ℓ(i, j), ‖c(a(i)) − c(j)‖/2)
                if j = a(i) or u(i) ≤ z then continue with next j
                if r then
                    u(i) ← ‖x(i) − c(a(i))‖
15:                 r ← False
                    if u(i) ≤ z then continue with next j
                ℓ(i, j) ← ‖x(i) − c(j)‖
                if ℓ(i, j) < u(i) then a(i) ← j
        for all j ∈ K do {Move the centers and track their movement}
20:         move c(j) to its new location
            let δ(j) be the distance moved by c(j)
        for all i ∈ N do {Update the upper and lower distance bounds}
            u(i) ← u(i) + δ(a(i))
            for all j ∈ K do
25:             ℓ(i, j) ← ℓ(i, j) − δ(j)
```

Figure 3: Algorithm 2

3.2 Hamerly's Algorithm

Hamerly modified Elkan's algorithm by using only one lower bound per point, $l(i)$ (see figure 4 for pseudo-code)[3]. This lower bound represents the minimum distance that any center (that is not the closest center) can be to that point. How is this better? consider the two cases following cases.

- If $u(i) \leq l(i)$: It is not possible for any center to be closer than the assigned center. Thus, the algorithm can skip the computation of the distances between *x(i)* and the k centers.
- If $l(i) < u(i)$: It might be that the closest center for *x(i)* has change. The algorithm first tightens the upper bound by computing the exact distance $\left\|x(i) - c(a(i))\right\|$. Then it checks again if $u(i) \leq l(i)$ to skip the distance calculations between *x(i)* and the k centers. If not, then it must compute those distances.

Having the single lower bound allows it to avoid entering the innermost loop more often than Elkan's algorithm. But Elkan's algorithm computes fewer distances than Hamerly's, and it doesn't work well with high dimensions because all centers tend to move a lot due the curse of dimensionality.

3.3 Drake's Algorithm

Drake and Hamerly combine the first two algorithms (Elkan's and Hamerly's) using $1 < b < k$ lower bounds on the b closest centers to each point. The value of b can be selected in advance or adaptively learned while the algorithms runs [1].

The first $b-1$ lower bounds for a point represent the lower bounds to the associated points that are ranked 2 through b in increasing distance from the point. The last lower bound (number b, furthest from the center) represents the lower bound on all the furthest $k-b$ centers (this is like Hamerly's one lower bound, but only for the outermost centers). Figure 5 shows the pseudo-code for this algorithm.

Algorithm 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}
      $u(i) \leftarrow u(i) + \delta(a(i))$
      $\ell(i) \leftarrow \ell(i) - \delta'$

Figure 4: Algorithm 3

Increasing b incurs more computation overhead to update the bound values and sort each point's closest centers by their bound, but it is also more likely that one of the bounds will prevent searching over all k centers. Experimentally, Drake determined that for $k > 8$, $k/8$ is a good floor for b.

Algorithm Drake's algorithm—using b lower bounds per point

procedure DRAKE(X, C, b)
  $a(i) \leftarrow 1, u(i) \leftarrow \infty, \forall i \in N$ {Initialize invalid bounds, all in one cluster.}
  $\ell(i, j) \leftarrow 0, \forall i \in N, j \in \{1, \ldots, b\}$
  while not converged **do**
    $m \leftarrow b$
    for all $i \in N$ **do**
      $j \leftarrow \arg\max_{1 \leq j' \leq b} u(i) \leq \ell(i, j')$
      if $j < b$ **then** {The bounds pruned the outer centers.}
        compute distances and reorder the j centers closest to $x(i)$
      else if $j = b$ or $\ell(i, b) < u(i)$ **then** {Bounds were ineffective.}
        compute distances from $x(i)$ to all centers and sort the b closest
      $m \leftarrow \max(m, j)$
    $b \leftarrow \max(k/8, m)$ {Reduce b if possible}
    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}
      $u(i) \leftarrow u(i) + \delta(a(i))$
      $\ell(i, b) \leftarrow \ell(i, b) - \delta'$
      for $j = b - 1$ down to 1 **do**
        let $c(z)$ be the center that is the jth closest to $x(i)$
        $\ell(i, j) \leftarrow \min(\ell(i, j) - \delta(z), \ell(i, j + 1))$

Figure 5: Algorithm 4

4 Bibliographic Notes

Lloyd's algorithm was first introduced in 1982 [4] and it has been very popular ever since. A lot of optimizations have been done base on his algorithm. In 2002 Phillips demonstrated how to use the triangle inequality to accelerate the algorithm [5]. Then, in 2003 Elkan used the concept of upper and lower bounds to provide another optimization for this algorithm [2]. Continuing with the approach using these bounds, Hamerly provided another algorithm in [3]. And 2 years after he collaborated with Drake to come up with an improved algorithm combining his work with Elkan's [1].

5 Exercises

1. Prove that the total running time of Lloyd's algorithm is *O(wnkd)* for w iterations, k centers, and n points in d dimensions.

2. Use the triangle inequality to prove that centers that are far from point's assigned center are also far from the point.

3. Prove that the upper and lower bounds on the given distance $\|x - c'\|$ are given by $\|x - c\| - \|c - c'\| \leq \|x - c'\| \leq \|x - c\| + \|c - c'\|$

References

[1] Jonathan Drake and Greg Hamerly. Accelerated k-means with adaptive distance bounds. *the 5th NIPS Workshop on Optimization for Machine Learning, OPT2012*, pages 2–5, 2012.

[2] Charles Elkan. Using the Triangle Inequality to Accelerate -Means. *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*, pages 147–153, 2003.

[3] Greg Hamerly. Making k-means even faster. *Computer*, pages 130–140, 2010.

[4] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[5] StevenJ. Phillips. *Acceleration of K-Means and Related Clustering Algorithms*, volume 2409 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2002.

[6] Rui Xu and Donald C. Wunsch. *Partitional Clustering*. 2008.