Towards the world’s fastest k-means algorithm

Greg Hamerly
Associate Professor
Computer Science Department
Baylor University

Joint work with Jonathan Drake

May 15, 2014
Towards the world’s fastest k-means algorithm

1. The k-means clustering algorithm
   - Objective function and optimization
   - Lloyd’s algorithm

2. Opportunities to speed up Lloyd’s algorithm

3. Algorithms that avoid distance calculations

4. Experimental results

5. Finally
Visual representation of k-means

Input

Output
Popularity and applications of k-means

Google searches (May 2014):

<table>
<thead>
<tr>
<th>Search query</th>
<th># hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-means clustering</td>
<td>2.6M, 316k</td>
</tr>
<tr>
<td>support vector machine classifier</td>
<td>1.7M, 477k</td>
</tr>
<tr>
<td>nearest neighbor classifier</td>
<td>0.5M, 103k</td>
</tr>
<tr>
<td>logistic regression classifier</td>
<td>0.3M, 61k</td>
</tr>
</tbody>
</table>

Applications:
- Discovering groups/structure in data
- Lossy data compression (e.g. color quantization, voice coding, representative sampling)
- Initialize more expensive algorithms (e.g. Gaussian mixtures)
The k-means clustering algorithm
Opportunities to speed up Lloyd’s algorithm
Algorithms that avoid distance calculations
Experimental results
Finally

Objective function and optimization
Lloyd’s algorithm

**Optimization criteria and NP-hardness**

K-means is not really an algorithm, it’s a criterion for clustering quality.

Criterion: \[ J(C, X) = \sum_{x \in X} \min_{c \in C} ||x - c||^2 \]

Goal: Find C that minimizes \( J(C, X) \)

- NP-hard in general.
- There are lots of approaches to finding ‘good enough’ solutions.
Hill-climbing approaches

The most popular algorithms rely on hill-climbing:
- Choose an initial set of centers.
- Repeat until convergence:
  - Move the centers to better locations.

Because $J(C, X)$ is non-convex, hill-climbing won’t in general find optimal solutions.
Lloyd’s algorithm

The most popular algorithm for k-means (Lloyd 1982)

Batch version:

- Choose an initial set of centers.
- Repeat until convergence:
  - Assign each point $x \in X$ to its currently closest center.
  - Move each center $c \in C$ to the average of its assigned points.
The k-means clustering algorithm
Opportunities to speed up Lloyd’s algorithm
Algorithms that avoid distance calculations
Experimental results
Finally

Objective function and optimization
Lloyd’s algorithm

Example

K-means (k=3, n=100) iteration 1

K-means (k=3, n=100) iteration 2

K-means (k=3, n=100) iteration 3

K-means (k=3, n=100) iteration 4

K-means (k=3, n=100) iteration 5

Towards the world’s fastest k-means algorithm
The $k$-means clustering algorithm
Opportunities to speed up Lloyd’s algorithm
Algorithms that avoid distance calculations
Experimental results
Finally

Objective function and optimization
Lloyd’s algorithm

Example

K-means ($k=3, n=100$) iteration 1

K-means ($k=3, n=100$) iteration 2

Towards the world’s fastest $k$-means algorithm
The k-means clustering algorithm

Opportunities to speed up Lloyd's algorithm

Algorithms that avoid distance calculations

Experimental results

Finally

Objective function and optimization

Lloyd's algorithm

Example

K-means (k=3, n=100) iteration 1

K-means (k=3, n=100) iteration 2

K-means (k=3, n=100) iteration 3

Towards the world’s fastest k-means algorithm
The \textit{k}-means clustering algorithm

Opportunities to speed up Lloyd’s algorithm

Algorithms that avoid distance calculations

Experimental results

Finally

\textbf{Example}

\begin{tabular}{ccc}
\textbf{K-means (k=3, n=100) iteration 1} & \textbf{K-means (k=3, n=100) iteration 2} & \textbf{K-means (k=3, n=100) iteration 3} \\
\includegraphics[width=0.3\textwidth]{iteration1} & \includegraphics[width=0.3\textwidth]{iteration2} & \includegraphics[width=0.3\textwidth]{iteration3} \\
\textbf{K-means (k=3, n=100) iteration 4} & & \\
\includegraphics[width=0.3\textwidth]{iteration4} & & \\
\end{tabular}
Example

The **$k$-means clustering algorithm**

- Opportunities to speed up Lloyd’s algorithm
- Algorithms that avoid distance calculations
- Experimental results
- Finally

**Objective function and optimization**

**Lloyd’s algorithm**

**Example**

- **K-means ($k=3$, $n=100$) iteration 1**
- **K-means ($k=3$, $n=100$) iteration 2**
- **K-means ($k=3$, $n=100$) iteration 3**
- **K-means ($k=3$, $n=100$) iteration 4**
- **K-means ($k=3$, $n=100$) iteration 5**

Towards the world’s fastest $k$-means algorithm
Lloyd’s algorithm is ‘fast enough’ most of the time:

- Each iteration is $O(nkd)$ in the size of the data
  - number of points $n$, clusters $k$, dimension $d$
- The number of iterations is *usually* small...
  - Theoretically it can be superpolynomial: $2^{\Omega(\sqrt{n})}$ (Vassilvitskii and Arthur 2006).
Lloyd’s algorithm is deterministic (given the same initialization).

A ‘good’ initialization is ‘close to’ the global optimum.

- What if the initialization is at the local (or global) optimum?

Common practice: try many initializations, keep the best.

k-means++ is a really good initialization, with provably optimal expected quality (Arthur and Vassilvitskii 2007).
Towards the world’s fastest k-means algorithm

1. The k-means clustering algorithm

2. Opportunities to speed up Lloyd’s algorithm
   - Many unnecessary distance calculations
   - Three key ideas

3. Algorithms that avoid distance calculations

4. Experimental results

5. Finally

Greg Hamerly / Baylor University
Lloyd’s algorithm spends the vast majority of its time determining distances.

For each point, what is its closest center?

- Naively, this is $O(kd)$ for each point.

Many (most!) of these distance calculations are unnecessary.
Reinventing the wheel

If you’re like me, you want to implement algorithms to understand them.


- None of these implement the accelerations presented here.

Let’s do something about this.
Lloyd’s algorithm is pervasive.

Therefore, we have a strong desire to create a fast version.

The algorithms I’ll talk about give exactly the same answer, but much faster.

This work is not about approximation, which can of course be many times faster still.
Key idea #1: Caching previous distances

From one iteration to the next, if a center doesn’t move much, the $O(n)$ distances to that center won’t change much either.

Could we save the distances computed in iteration $t$ to use in iteration $t + 1$?

- Not directly...
Key idea #2: Distances are sufficient but not necessary

What if we didn’t have distances, but we had an oracle that can answer the question:

- Given a point $x$, what is its closest $c \in C$?

We could still run Lloyd’s k-means algorithm!

Point: distances are unnecessary; we only need which center is closest.
Key idea #3: Triangle inequality

\[ \|a - b\| \leq \|a - c\| + \|b - c\| \]
Key idea #3: Triangle inequality

\[ \|a - b\| \leq \|a - c\| + \|b - c\| \]

We can apply this to moving centers.

If we know \( \|x - c\| \), and \( c \) moves to \( c' \), then

\[ \|x - c'\| \leq \|x - c\| + \|c - c'\| \]
Key idea #3: Triangle inequality

\[|a - b| \leq |a - c| + |b - c|\]

We can apply this to moving centers.

If we know \(|x - c|\), and \(c\) moves to \(c'\), then

\[|x - c'| \leq |x - c| + |c - c'|\]

This is an upper bound; we can also construct a lower bound.
Combining these three ideas

We can maintain bounds on the distance between \( x \) and each center \( c \in C \).

- Upper bound between \( x \) and its closest center.
- Lower bound(s) between \( x \) and other centers.

Efficiently update bounds when centers move, using the triangle inequality.

Use the bounds to prune point-center distance computations.

- Between points and far-away centers.
Towards the world’s fastest k-means algorithm

1. The k-means clustering algorithm
2. Opportunities to speed up Lloyd’s algorithm
3. Algorithms that avoid distance calculations
4. Experimental results
5. Finally
Avoiding distance calculations

K-d tree:
- Pelleg & Moore (1999)
- Kanungo et. al (1999)

Triangle inequality:

**Triangle inequality plus distance bounds (today’s talk):**
- Elkan (2003)
- Hamerly (2010)
- Drake (2012)
- Annular (2014)
- Heap (2014)
Elkan’s k-means

Elkan (2003) proposed using:

- $\ell(x, c)$: $k$ lower bounds per point (one for each center)
- $u(x)$: one lower bound per point (for its assigned center)
- $k^2$ inter-center distances
- $s(c)$: distance from $c$ to the closest other center
Elkan’s k-means

Elkan (2003) proposed using:

- \( \ell(x, c) \): \( k \) lower bounds per point (one for each center)
- \( u(x) \): one lower bound per point (for its assigned center)
- \( k^2 \) inter-center distances
- \( s(c) \): distance from \( c \) to the closest other center

Several ways to apply these bounds. Key ones are:

- if \( u(x) \leq s(a(x))/2 \), then \( a(x) \) is closest to \( x \)
Elkan’s k-means

Elkan (2003) proposed using:

- \( \ell(x, c) \): \textbf{k lower bounds} per point (one for each center)
- \( u(x) \): one \textbf{lower bound} per point (for its assigned center)
- \( k^2 \) \textbf{inter-center distances}
- \( s(c) \): distance from \( c \) to the closest other center

Several ways to apply these bounds. Key ones are:

- if \( u(x) \leq s(a(x))/2 \), then \( a(x) \) is closest to \( x \)
- if \( u(x) \leq \ell(x, c) \), then \( a(x) \) is closer than \( c' \) to \( x \)
Hamerly’s k-means

Hamerly (2010) proposed the following simplifications of Elkan’s algorithm:

- $\ell(x)$: only one lower bound per point (for the second-closest center)
- no inter-center distances for pruning

Advantages:

- Simpler ($u(x) \leq \ell(x)$)
- Lower memory footprint
- Better at skipping innermost loop over centers
- Faster in practice in low dimension
Drake’s k-means

Drake (2012) proposed a bridge between Elkan and Hamerly’s algorithms:

- $\ell(x, c)$: $b$ lower bounds per point ($1 < b < k$), for the $b$ closest centers

Advantages:
- Tunable parameter $b$
- Faster in practice for moderate dimensions
Annular k-means

Hamerly and Drake (2014) proposed an extra acceleration on Hamerly’s algorithm.

Each iteration, order the centers by distance from the origin.

When searching for the closest center, use distance bounds to prune the search.

Advantages:
- Negligible extra memory and overhead
- Large benefit in low dimension
Heap k-means

Hamerly and Drake (2014) inverted the order of loops using $k$ min-heaps.

- For each center $c$
  - For each point $x$ assigned to $c$
    - Find the closest center to $x$
Heap k-means

Hamerly and Drake (2014) inverted the order of loops using $k$ min-heaps.

- For each center $c$
  - For each point $x$ assigned to $c$
    - Find the closest center to $x$

Idea: Use priority queues to prune those points close to their assigned centers.
Heap k-means

Hamerly and Drake (2014) inverted the order of loops using $k$ min-heaps.

- For each center $c$
  - For each point $x$ assigned to $c$
    - Find the closest center to $x$

Idea: Use priority queues to prune those points close to their assigned centers.

Each cluster has a heap, ordered by the difference between the lower and upper bounds: $\ell(x) - u(x)$.

Naively, heap priorities change with each center move. Efficient updates are an interesting problem.
### Summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Year</th>
<th>Upper bound</th>
<th>Lower bounds</th>
<th>Closest other center</th>
<th>Sorting</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare-means (1)</td>
<td>2002</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Sort-means (1)</td>
<td>2002</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>k(^2) centers</td>
<td>(2)</td>
</tr>
<tr>
<td>Elkan</td>
<td>2003</td>
<td>1</td>
<td>k</td>
<td>x</td>
<td>-</td>
<td>(2)</td>
</tr>
<tr>
<td>Hamerly</td>
<td>2010</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Annular</td>
<td>2014</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>centers</td>
<td>-</td>
</tr>
<tr>
<td>Heap</td>
<td>2014</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>lower - upper</td>
<td>-</td>
</tr>
<tr>
<td>Drake</td>
<td>2012</td>
<td>1</td>
<td>b</td>
<td>x</td>
<td>lower bounds</td>
<td>-</td>
</tr>
</tbody>
</table>

1. Phillips
2. \(k^2\) center-center distances
Towards the world’s fastest k-means algorithm

1. The k-means clustering algorithm
2. Opportunities to speed up Lloyd’s algorithm
3. Algorithms that avoid distance calculations
4. Experimental results
   - Speedup
   - Effect of dimension
   - Bound effectiveness
   - Parallelism
   - Memory use
Datasets

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Number of points ( n )</th>
<th>Dimension ( d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniform-2/8/32</td>
<td>synthetic, uniform distribution</td>
<td>1,000,000</td>
<td>2/8/32</td>
</tr>
<tr>
<td>clustered-2/8/32</td>
<td>synthetic, 50 separated spherical Gaussian clusters</td>
<td>1,000,000</td>
<td>2/8/32</td>
</tr>
<tr>
<td>BIRCH</td>
<td>10 × 10 grid of Gaussian clusters</td>
<td>100,000</td>
<td>2</td>
</tr>
<tr>
<td>MNIST-50</td>
<td>random projection from mnist784</td>
<td>60,000</td>
<td>50</td>
</tr>
<tr>
<td>Covertype</td>
<td>soil cover measurements</td>
<td>581,012</td>
<td>54</td>
</tr>
<tr>
<td>KDD Cup 1998</td>
<td>response rates for fundraising campaign</td>
<td>95,412</td>
<td>56</td>
</tr>
<tr>
<td>MNIST-784</td>
<td>raster images of handwritten digits</td>
<td>60,000</td>
<td>784</td>
</tr>
</tbody>
</table>

Towards the world’s fastest k-means algorithm
Experimental platform

Linux running on 8-12 core machines with 16 GB of RAM per machine.

Software written in C++ with a lot of shared code for similar algorithms.
The k-means clustering algorithm
Opportunities to speed up Lloyd’s algorithm
Algorithms that avoid distance calculations
Experimental results
Finally

Speedup (relative to naive algorithm) for clustered data

- 50 true Gaussians, $n = 10^6$.
- $K$ varies from 2 to 128.
The k-means clustering algorithm
Opportunities to speed up Lloyd’s algorithm
Algorithms that avoid distance calculations
Experimental results
Finally

Speedup (relative to naive algorithm) for uniform data

- Uniform distribution, $n = 10^6$.
- K varies from 2 to 128.
The $k$-means clustering algorithm
Opportunities to speed up Lloyd’s algorithm
Algorithms that avoid distance calculations
Experimental results

Finally

Speedup (relative to naive algorithm) for Covtype, KDD Cup

$\begin{align*}
d &= 54, & n &= 581012 \\

\end{align*}$

$\begin{align*}
d &= 56, & n &= 95412 \\

\end{align*}$
Speedup (relative to naive algorithm) for MNIST

\[ d = 784, \quad n = 60000 \]

\[ d = 50, \quad n = 60000 \]
### Curse of dimensionality

<table>
<thead>
<tr>
<th>$d$</th>
<th>Uniform data, $k = 128$.</th>
<th>Reporting number of distance calculations.</th>
<th>Algorithms which use bounds do much better.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td><img src="image1.png" alt="Graph" /></td>
<td><img src="image2.png" alt="Graph" /></td>
<td><img src="image3.png" alt="Graph" /></td>
</tr>
<tr>
<td>8</td>
<td><img src="image1.png" alt="Graph" /></td>
<td><img src="image2.png" alt="Graph" /></td>
<td><img src="image3.png" alt="Graph" /></td>
</tr>
<tr>
<td>32</td>
<td><img src="image1.png" alt="Graph" /></td>
<td><img src="image2.png" alt="Graph" /></td>
<td><img src="image3.png" alt="Graph" /></td>
</tr>
</tbody>
</table>
Elkan and Drake’s algorithms use multiple lower bounds.

Which bounds are most effective?

Hamerly showed the single lower bound can avoid 80+% of innermost loops, regardless of dataset and dimension.

Drake showed:
- In early iterations (<10), the first several bounds are most effective.
- After that, the first bound prevents 90+% of avoided distance calculations.
K-means has natural parallelism

- Used pthreads on 12-core machine.
- Naive algorithm embarrassingly parallel within an iteration.
- Partition data over threads, replicate centers.
- Acceleration can cause work imbalance and add synchronization.

$k = 32$

$k = 128$
Memory overhead

- Uniform dataset, \( d = 32 \).
- Algorithms using 1 lower bound use negligible extra memory.
- Drake & Elkan’s algorithms use significantly more memory when \( k \) is large.
Towards the world’s fastest k-means algorithm

1. The k-means clustering algorithm
2. Opportunities to speed up Lloyd’s algorithm
3. Algorithms that avoid distance calculations
4. Experimental results
5. Finally
Discussion

Key to acceleration: cached bounds, updated using the triangle inequality.

More lower bounds avoids more distances, giving better performance in high dimension.

Low dimension ($< 50$) really only needs one lower bound.

Memory impact is negligible for one lower bound.
Future work

Theoretical lower bounds on required number of distance calculations.

Other clever ways to avoid doing work; e.g. other bounds.

Accelerating other algorithms using these techniques.
  - Dynamic nearest neighbor search.
  - Clustering of dynamic datasets – any takers?
Conclusion

K-means is popular, and easy to implement.

- Therefore, everyone implements it... slowly.

Simple acceleration methods exist that use little extra memory.

- Key ideas: caching, triangle inequality, and distance bounds.

Software (C++) is available, just email me.
hamerly@cs.baylor.edu

Questions?
Accelerating k-means:


Other references:

Towards the world’s fastest k-means algorithm

Other acceleration methods
Pelleg & Moore, Kanungo & Mount (1999) each separately proposed using k-d trees to accelerate k-means.

Works well in low dimension, but slow above about 8 dimensions.
Pelleg & Moore, Kanungo & Mount (1999) each separately proposed using k-d trees to accelerate k-means.

Works well in low dimension, but slow above about 8 dimensions.

Moore (2000) proposed a new structure, the anchors hierarchy, based on the triangle inequality. Uses carefully-chosen ‘anchors’.

Built middle-out (rather than top-down).

Common disadvantages: extra structure, complicated, preprocessing, don’t adapt to changing centers.
Sampling-based approximations

Downsample the dataset and cluster just that sample.

Stochastic gradient descent (Bottou and Bengio 1995): move centers after considering each example.

Mini-batch (Sculley 2010): stochastic gradient descent using small samples.
Other acceleration methods

Projection to combat dimensionality problems

The curse of dimensionality limits acceleration algorithms.

Random projection (see Dasgupta 2000) is an excellent way to reduce the dimension of data for clustering.

- fast – linear time
- tends to produce spherical, well-separated clusters

Applying random projection:

- generate a random projection matrix $P$
- project the data using $P$
- cluster in the low-dimension space
- project clusterings back to original space using assignments
- finish clustering in original space (if desired)
A good initialization leads to few k-means iterations.

K-means++ is the best current initialization method, but it is slow.

- Runs in time $O(nkd)$.
- Can apply triangle inequality to reduce the $d$ factor.
- Can we do it faster? (Current work.)
Partial distance search can prune parts of distance calculations, especially in high dimension.

Suppose $d$ is dimension, $d' < d$, and $x, a, b$ are $d$-dimensional points:

$$\sum_{i=1}^{d}(x_i - a_i)^2 \leq \sum_{i=1}^{d'}(x_i - b_i)^2$$

Then we know $a$ is closer than $b$ to $x$, even before computing the distance between $x$ and $b$.

This works for k-means: any known distance (or upper bound) can prune the search.