Yinyang K-Means: A Drop-In Replacement of the Classic K-Means with Consistent Speedup

Yufei Ding, Yue Zhao, Xipeng Shen
Madanlal Musuvathi, Todd Mytkowicz

North Carolina State University, USA
Microsoft Research at Redmond, USA
Popular Clustering Algorithm: K-Means

One of the most popular algorithms for clustering.

Decades of usage in various domains since proposed by Lloyd in 1957.
Classic Algorithm [by Lloyd]

Each assignment step calculates N*K distances. cause slow clustering for large problems

- Group N points into K clusters
- Set initial centers
  - Assign points to clusters based on d(p, c) ∀p,c
  - Update centers w/ new centroids
- Convergence
Are all distance calculations necessary?

Each assignment step calculates \( N \times K \) distances.

cause slow clustering for large problems

Group \( N \) points into \( K \) clusters

If \( c \) is not the closest center to \( p \), it’s safe to skip computing \( d(p, c) \).
Prior Work

• K-D Tree [Kanungo et al., 2002]
  • Slowdowns for high dimensions

• Incremental Optimization

Lloyd’s K-Means remains the dominant choice in practice!

• Many times of memory space cost
• Slowdowns in some cases (medium dim, large K & N)

• Approximation [Wang et al., 2012]
  • Different results
  • Unable to inherit the level of trust
Goal of This Work

Develop a drop-in replacement of Lloyd’s K-Means.

- **Speed:** Much faster in all settings (N, K, D)
- **Trust:** Same results as Lloyd’s K-Means produces
- **Simplicity:** Easy to implement and deploy
Overview of Yinyang K-Means

Minimize # of distance calculations:

- Carefully maintain and leverage **lower bounds** & **upper bounds**
- A 3-level filter
- Space-conscious elastic design

On average, **9.36X faster** than classic K-Means.
No slowdown regardless of N, K, dim.
**Triangular Inequality**

- Fundamental tool for getting bounds

\[
|d(x,c') - d(c',c)| \leq d(x,c) \leq d(x,c') + d(c',c)
\]

lower bound: \(\text{lb}(x, c)\)  
upper bound: \(\text{ub}(x, c)\)

- \(x\): data point
- \(C'\): center in iteration \(i\)
- \(C\): center in iteration \(i+1\)
Assignment Step

- Three levels of filters
- Overhead V.S. Benefits
Global Filtering

- One single check to decide whether a point is possible to change its center assignment.

Question: Is $c_1$ the closest center for $x$ in this iteration?

Prior iter: $G' = \{C'_1, C'_2, \ldots, C'_k\} - C'_1$

This iter: $G = \{C_1, C_2, \ldots, C_k\} - C_1$
Global Filtering

- One single check to decide whether a point is possible to change its center assignment.

Principle 1:
\[ \text{if } \text{ub}(x,c_1) \leq \text{lb}(x, G); \text{ then x does not change assignment.} \]

- Compute \( \text{ub}(x,c_1) \) and \( \text{lb}(x, G) \)
  \[ \text{ub}(x,c_1) = \text{ub}(x, c') + \Delta c_1 \]
  \[ \text{lb}(x, G) = \text{lb}(x,G') - \text{maxdelta} \]

- \( \Delta c_1 = d(c,c') \)
  \[ \text{maxdelta} = \text{Max} (\Delta(c)) \quad \forall c \]
Global Filtering

\[
\begin{align*}
\text{ub}(x, c_1) &= \text{ub}(x, c'_1) + \Delta c_1 \\
\text{lb}(x, G) &\leq \text{lb}(x, G') - \text{maxdelta}
\end{align*}
\]

Overhead:
Space Cost: \( O(n) \) to store bounds
Time Cost: \( O(k \cdot d + n) \) to compute center shifts and bounds

Benefits: \(~60\%\) redundant distance computation can be removed
• Limiting factor: \( \text{maxdelta} = \text{Max} (\Delta(c)) \quad \forall c \)
Group Filtering

• Group filtering:

- **Principle I:**
  
  If \( \text{ub}(x, c_i) \leq \text{lb}(x, G_i) \);
  
  then no center in \( G_i \) can be closest to \( x \).

- **Rules to compute \( \text{ub}(x, c_i) \) and \( \text{lb}(x, G) \)**

  \[
  \text{ub}(x, c_i) = \text{ub}(x, c'_i) + \Delta c_i
  \]

  \[
  \text{lb}(x, G_i) \leq \text{lb}(x, G'_i) - \maxdelta(G_i)
  \]

- \( \Delta c_i = d(c_i, c'_i) \)

  \[
  \maxdelta(G_i) = \text{Max} (\Delta(c)) \quad \forall c \in G_i
  \]
Group Filtering

• How to do the grouping?

Partial K-Means on initial centers
(one time preprocessing)

Overhead: $O(k \cdot m \cdot d \cdot \text{iter})$
(we set iter to 5)

Choosing $m$—Elasticity:
• $m = k/10$ if space allows
• max value otherwise
Group Filtering

• Efficiency and overhead

\[
\begin{align*}
ub(x, c_1) &= ub(x, c'_1) + \Delta(c_1) \\
lb(x, G_i) &\leq lb(x, G_i) - \text{maxdelta}(G_i)
\end{align*}
\]

Overhead:
Space Cost: \(O(n \cdot m)\) to maintain bounds across iteration
Time Cost: \(O(k \cdot d + n \cdot m)\) to compute center shifts and bounds

Efficiency:
\~80\% redundant distance computation can be removed
Local Filtering

Check each remaining center, and skip c if
\[ \min_2(x) \leq lb(x, Gi) - \Delta c_j \]

\(\min_2(x)\): the so-far 2nd shortest distance from x.

No extra space cost!
Update Step

Set initial centers

Assign points to clusters based on $d(p, c) \forall p, c$

Update centers w/ new centroids

Convergence

See paper.
Evaluation

• Compared to three other methods:
  • Classic (Lloyd’s) K-Means
  • Elkan’s K-Means [2003]
  • Drake’s K-Means [2012]
• Input: real-world data sets (with different N, K, Dim)
  • 4 from UCI machine learning repository [Bache Lichman, 2013]
  • 4 other commonly used image data sets [Wang et al., 2012].
• Implemented in GraphLab (a map-reduce framework)
• Two machines
  • 16GB memory, 8-core i7-3770K processor
  • 4GB memory, 4-core Core2 CPU
Baseline: Classic K-means

(16GB, 8-core Intel Ivy Bridge)
Baseline: Classic K-means
(16GB, 8-core)
Please read our paper for details!
Final Takeaways

• Yinyang K-Means: A Drop-In Replacement
  • Consistently much faster
    • Minimize distance calculations (via a Yinyang harmony)
  • Superior cost-benefit tradeoff
    • Elastic design
  • Inherited trust
    • Same results as classic K-Means gives

Code: http://research.csc.ncsu.edu/nc-caps/yykmeans.tar.bz2