Performance Evaluation of a Real-Time Clustering Algorithm

Gabriel J. Ferrer, Ph.D.
Professor of Computer Science
Department of Mathematics and Computer Science
Hendrix College

Overview

Real-Time Clustering

- One iteration of training
- \(O(1)\) (constant) time
- Cannot depend on prior number of training samples
- To train \(n\) samples, \(O(n)\) is okay
- After each sample
  - Algorithm must be updated and ready to go.
- To train \(n\) samples, we meet \(n\) deadlines.
- No amortized time!

Performance Evaluation

- One input
- Sum-of-squared-differences (SSD) between that input and its best-matching cluster
- Set of inputs
  - Mean SSD between each input and its best-matching cluster

Experiments

Clustering Problem: Color Clustering

- RGB Image Colors
- Extremely large variety in a video sequence
- Perform clustering to reduce the total number of colors
- Each clustered input is an RGB triple

Data

- Two 45-second video sequences
- Outside - Filmed outdoors on college campus
- Inside - Filmed indoors in office environment
- Frame dimensions: 1920 x 1080
- Short version - 10 frames, sampled every 4.5 seconds
- All three algorithms
  - 20,736,000 total pixels clustered
  - Long version - 90 frames, samples twice per second
  - Real-time algorithms only
  - 186,624,000 total pixels clustered
  - Memory issues with k-means++ implementation
  - Mean SSD for each RIC data point is the mean of ten runs

Boundeds Self-Organizing Clusters (BSOC) [1]

setup(max-nodes)
edges = new min-heap
nodes = array of inputs

lookup(input)
for each node
  find distance from input to node
return (node-index, distance) of closest node

train(input)
insert(input, 1)
if number of nodes exceeds max-nodes
  edge = edges.remove(smallest edge)
  (n1, n2) = endpoints of edge
  Remove n1 and n2 and their edges
  insert(merged(n1,n2), n1.count + n2.count)

merged(n1, n2)
w1 = n1.count / (n1.count + n2.count)
w2 = n2.count / (n1.count + n2.count)
img = w1 * n1.image + w2 * n2.image
return img

Random Incremental Clusters (RIC)

setup(max-nodes)
nodes = array of max-nodes inputs
attempts = 0

lookup(input)
for each node
  find distance from input to node
return (node-index, distance) of closest node

train(input)
i = random # from [0, attempts]
if i < capacity of nodes
  nodes[i] = input
attempts += 1

Conclusions

- In all cases, performance improves with increasing numbers of clusters
- K-Means++ best at minimizing SSD
- BSOC is fairly close to K-Means++
- Relatively minimal "penalty" for real-time operation
- Much better memory utilization than K-Means++
- RIC often performs well at larger numbers of clusters

Citations
