ECE 592
Topics in Data Science

Dror Baron
Associate Professor
Dept. of Electrical and Computer Engr.
North Carolina State University, NC, USA

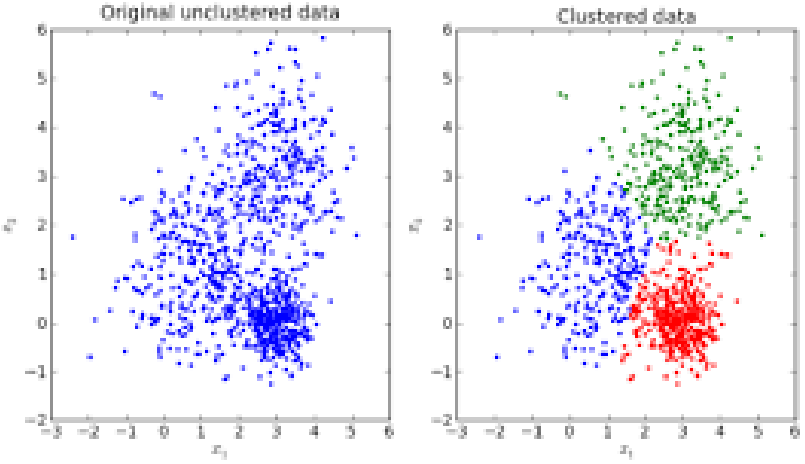
Clustering and Project 1

[Hastie et al., Section 14.3]

Keywords: clustering, unsupervised learning

Goals

- Want to group data into “clusters” that seem related



- Central notion – degree of similarity between different clusters

- Typical algorithmic approach is iterative; move points between clusters, recalculate cluster centers

Main steps

- Part 1: Upload image
  - Will compress image as we proceed

- Part 2: initial compression approach
  - Partition image into blocks/patches
  - Arrange all patches into list
  - Run clustering on list of patches
  - For each cluster, *representation patch* will be average of patches in that cluster

Part 3: Rate-distortion (RD) trade-off

- Compression vs. image quality
  - Each patch in image belongs to some cluster in dictionary
  - Encoder (compressor) will output index of cluster
  - Requires $\log_2$(#clusters) bits per patch
  - Decoder (decompressor) maps index to patch in dictionary

- More clusters → less *distortion* (between image & compressed version) but larger coding *rate*
- Trade-off between rate and distortion
- Fundamental information theoretic limits studied in rate distortion theory

Part 4: Patch size and RD performance

- Use bigger patches → captures more dependencies between pixels → better RD performance
- Problem1: bigger patches may increase computation
- Problem2: if patches are too big, might not be enough → clustering process won't work as well

Clustering via K-means algoirthm

Keywords: K-means, clustering

K means algorithm

- Initialize K cluster centers
  - Select K points among training data
- Iterate until convergence:
  - Associate each training datum with nearest cluster center
  - Recompute cluster centers as average of training data in cluster
- Sensitive to initialization (can get stuck in local optimum)
- Other clustering algos use model for cluster

More about K means

- Map datum x_n to cluster C(n)=k to representation level r_k, k=k(n)=C(n)
- Squared error between x_n and r_k

$$d(x_n, r_{k)}) = \sum_{p=1}^{P}(x_{np} - rkp)^2 = \|x_n - r_k\|^2$$

- Want $r_k = \min_{x \in \mathbb{R}^P} \sum_{\{n:C(n)=k\}} \|x_n - rk\|^2$
- Select *cluster center,* $r_k = \frac{1}{|\{n:C(n)=k\}|} \sum_{\{n:C(n)=k\}} x_n$
- Summed square errors for mapping C

$$\text{Error}(C) = \sum_{n=1}^{N} \left\|x_n - r_{k(n)}\right\|^2 \qquad \textit{sum over N data}$$

$$= \sum_{k=1}^{K} \sum_{\{n:C(n)=k\}} \|x_n - r_k\|^2 \qquad \textit{sum over K clusters}$$

Project 2

Keywords: Dijkstra’s algorithm

Main idea - “GPS algorithm”

- Will download map of North Carolina (NC) with distances between several hundred locations
- Remove long distances (e.g. >20 miles) to go between nearby locations directly
  - Reduces number of edges
- Select two locations; run Dijkstra’s algorithm to find shortest path

Dijkstra's algorithm

- Finds shortest paths from target t node to others
- Maintain set of un/visited nodes; initialize all as unvisited
- Maintain set of distances
  - Initialize as zero for node t, infinity for others
  - While node is unvisited, distance is tentative

- While set of unvisited nodes is non-empty:
  - Select unvisited node n with shortest distance
  - For v neighbors of n, $dis_v = \min(dis_v, d_n + edge(n,v))$
  - Mark n as visited

Project 3

Keywords: merge sort

Improving merge sort

- Will consider different ways to improve our merge sort implementation

- Approach 1: moving sequences back and forth between routines is costly
  - Can be shown to require O(n) per function call → aggregating over various function calls O(nlog(n))
  - This extra cost increases constant in complexity term
  - Can reduce to O(1) per function call by passing indices into array

More improvements

- Approach 2: functions in same file
  - Routines in different files require the software system to spend resources opening and reading files
  - May accelerate runtime to put everything in same file
  - May further accelerate runtime to inline merge function within merge sort routine

- Approach 3: larger basis case
  - Current implementation considers basis case for n=1 (already sorted)
  - Performing n=2 (i.e., output=(min(input),max(input))) will halve number of function calls, likely accelerating runtime
  - Can consider larger basis cases too, possibly with insert sort