Data-Intensive Distributed Computing
CS 451/651 (Fall 2025)

UNIVERSITY OF WATERLOO

Finding Similar Items
(v1.00)

Week 11: November 13, 2025

Jimmy Lin
David R. Cheriton School of Computer Science
University of Waterloo

These slides are available at http://lintool.github.io/bigdata-2025f/

Key Questions

What makes certain types of clustering algorithms amenable to scale-out distributed processing?

What are the parallels between *k*-means clustering and Gaussian Mixture Models?

Theme: Similarity

How similar are two items? How "close" are two items?

Equivalent formulations: large distance = low similarity

Lots of applications!

Problem: find similar items

Offline variant: extract all similar pairs of objects from a large collection

Online variant: is this object similar to something I've seen before?

Last time!

Problem: arrange similar items into clusters

Offline variant: entire static collection available at once

~~Online variant: objects incrementally available~~

Today!

Clustering Criteria

How to form clusters?

High similarity (low distance) between items in the same cluster

Low similarity (high distance) between items in different clusters

Cluster labeling is a separate (difficult) problem!

Applications of Clustering

Important step in exploratory data analysis

"What's there?"

Important step in data selection and curation

"How do I focus on a specific type of object?"

Examples

Clustering prompts to find out what people are typing into LLMs

Clustering training data to curate topics

Clustering images to summarize search results

Clustering customers to infer habits

Clustering biological sequences to understand evolution

…

LSH: Four Steps

Specify distance metric

Jaccard, Euclidean, cosine, etc.

Compute representation

Shingling, tf.idf, etc.

Hash

Minhash, random projections, etc.

Examine collisions

Process items in same “bucket”

Clustering: Three Steps

Specify distance metric

Jaccard, Euclidean, cosine, etc.

Compute representation

Shingling, tf.idf, etc.

Apply clustering algorithm

Today: Different Clustering Algorithms

HAC

Birch

K-Means

Gaussian Mixture Models

HAC

(Hierarchical Agglomerative Clustering)

Hierarchical Agglomerative Clustering

Start with each object in its own cluster

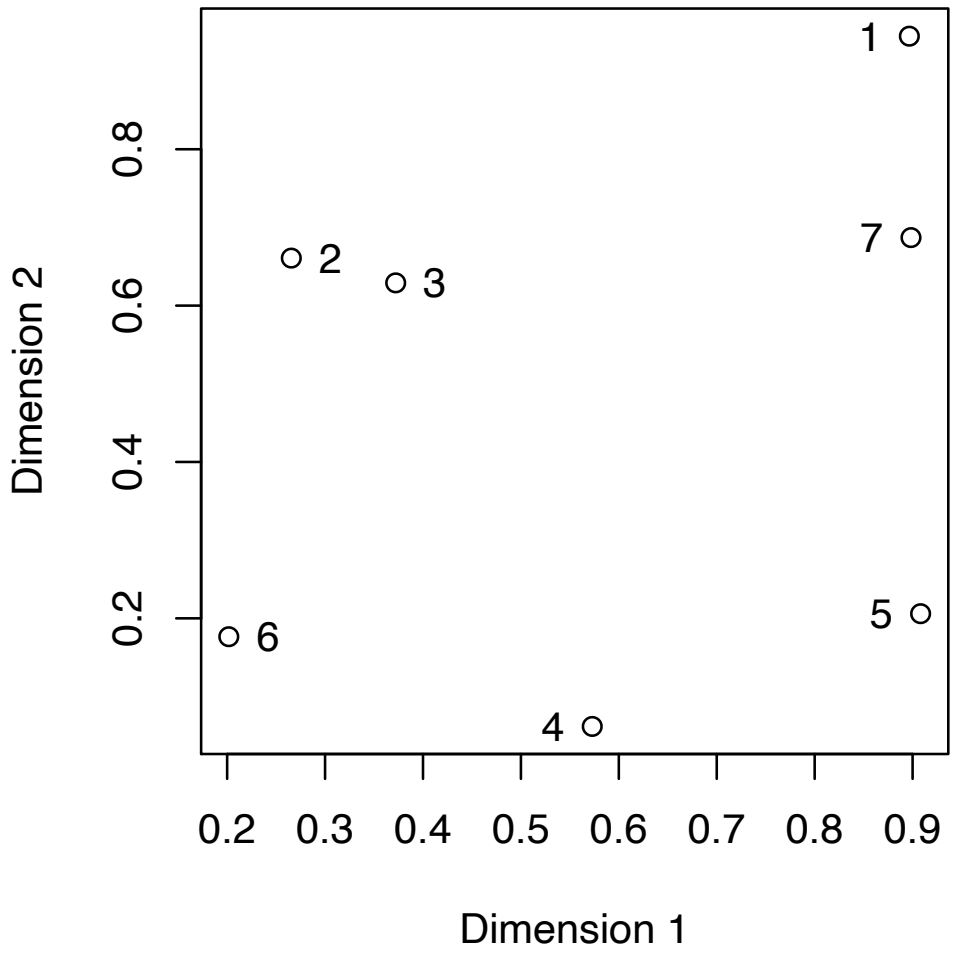
Until there is only one cluster:

Find the two clusters c_i and c_j, that are most similar

Replace c_i and c_j with a single cluster $c_i \cup c_j$

The history of merges forms the hierarchy

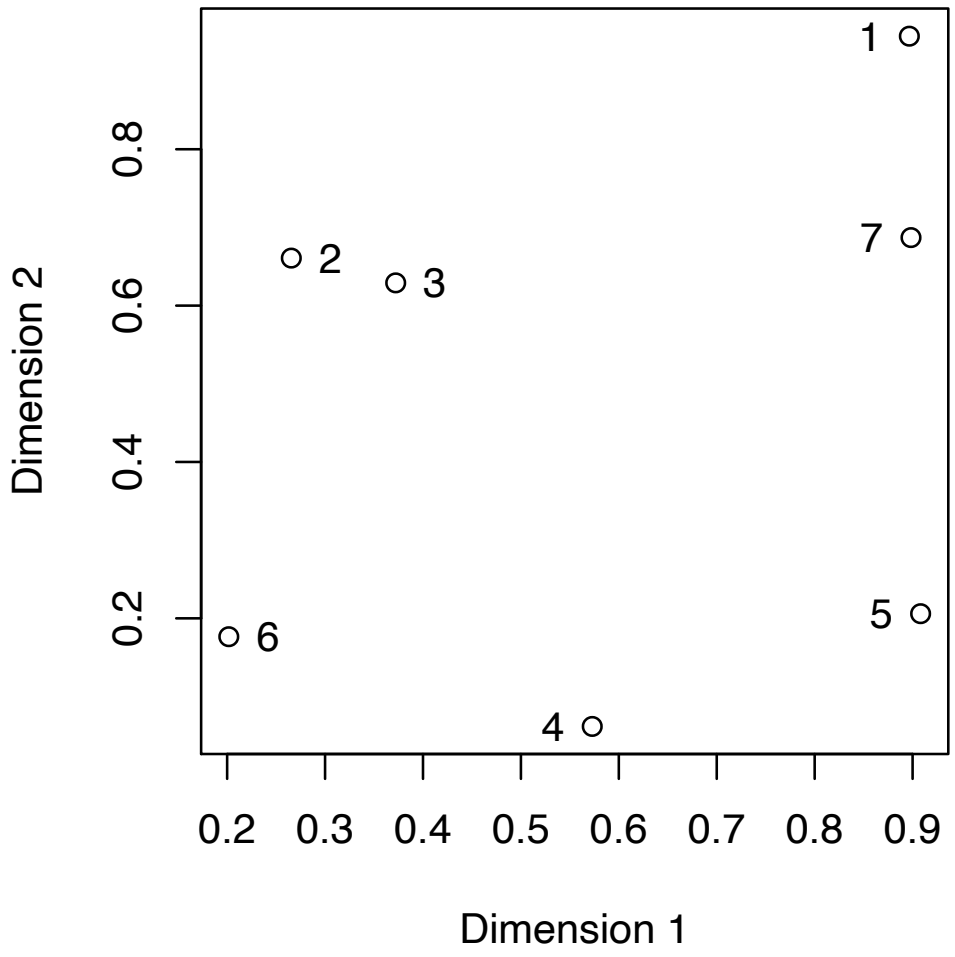
HAC in Action

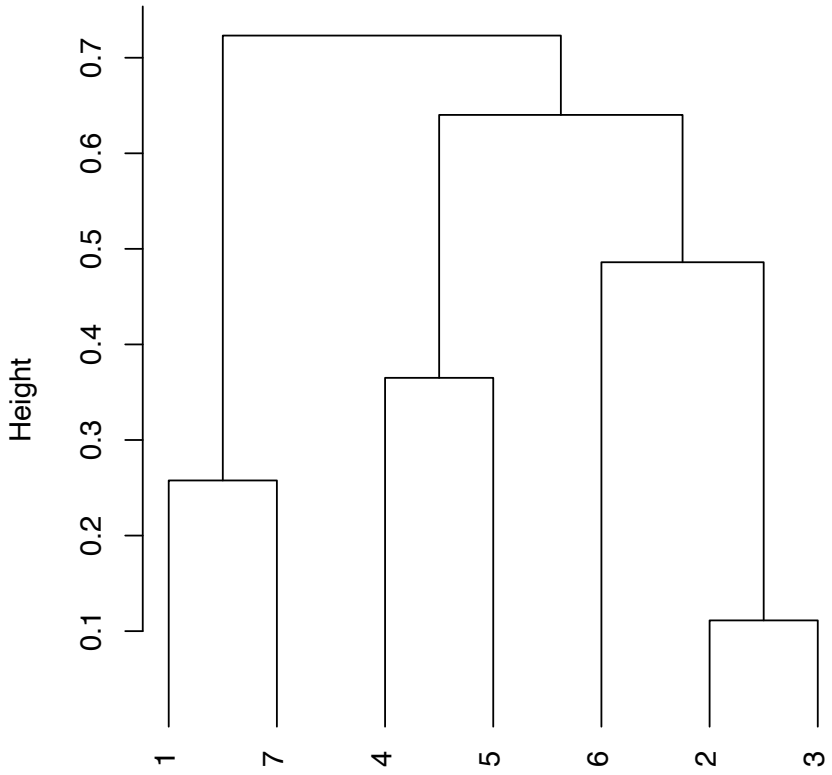


Step 1: {1}, {2}, {3}, {4}, {5}, {6}, {7}
Step 2: {1}, {2, 3}, {4}, {5}, {6}, {7}
Step 3: {1, 7}, {2, 3}, {4}, {5}, {6}
Step 4: {1, 7}, {2, 3}, {4, 5}, {6}
Step 5: {1, 7}, {2, 3, 6}, {4, 5}
Step 6: {1, 7}, {2, 3, 4, 5, 6}
Step 7: {1, 2, 3, 4, 5, 6, 7}

Source: Slides by Ryan Tibshirani

Dendrogram





Source: Slides by Ryan Tibshirani

Cluster Merging

Which two clusters do we merge?

What's the similarity between two clusters?

Single Linkage: similarity of two most similar members

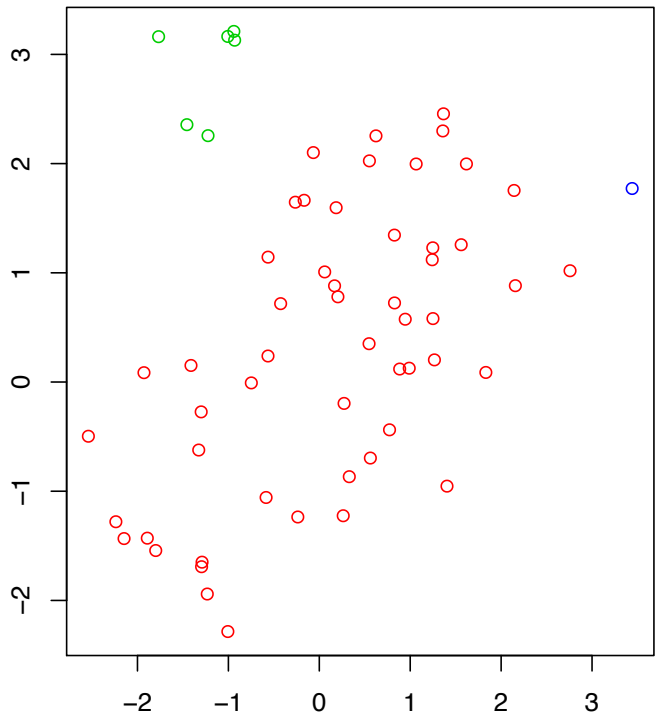
Complete Linkage: similarity of two least similar members

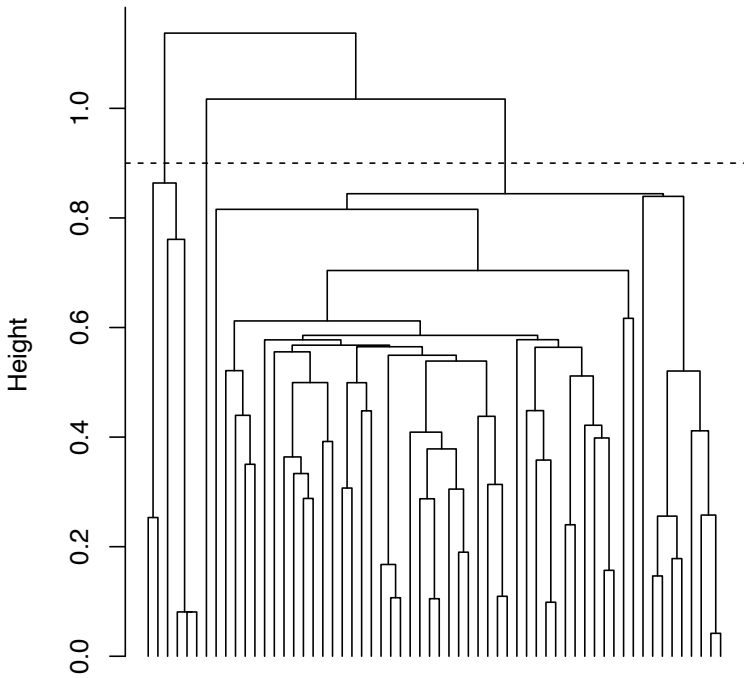
Average Linkage: average similarity between members

Single Linkage

Uses maximum similarity (min distance) of pairs:

$$d_{\text{single}}(G, H) = \min_{i \in G,\, j \in H} d_{ij}$$



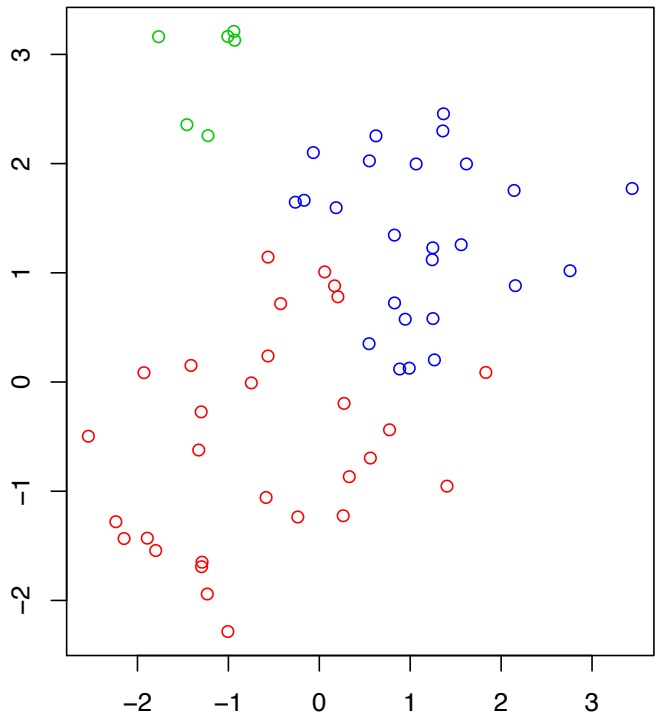


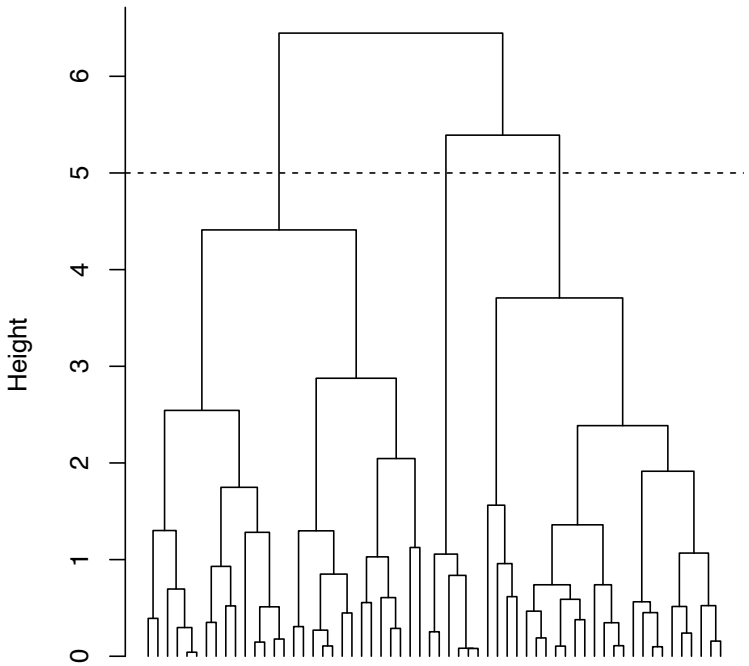
Source: Slides by Ryan Tibshirani

Complete Linkage

Uses minimum similarity (max distance) of pairs:

$$d_{\text{complete}}(G, H) = \max_{i \in G,\, j \in H} d_{ij}$$



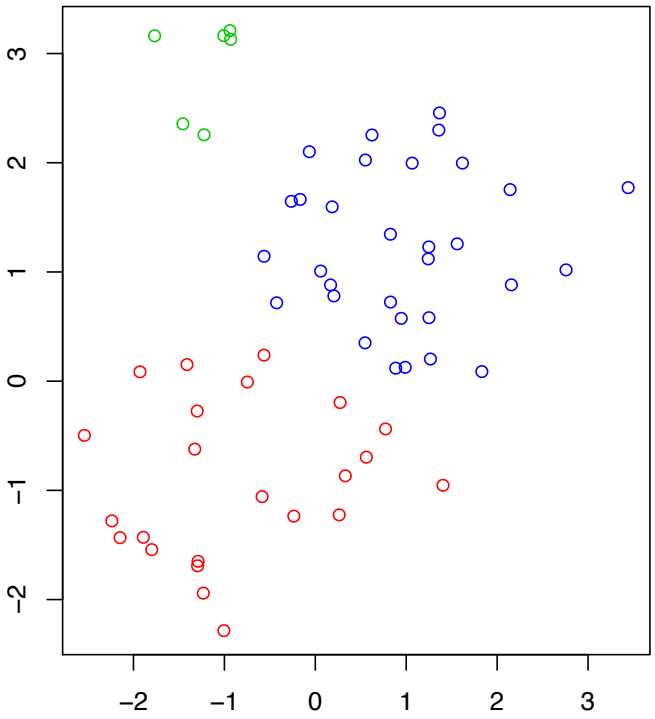


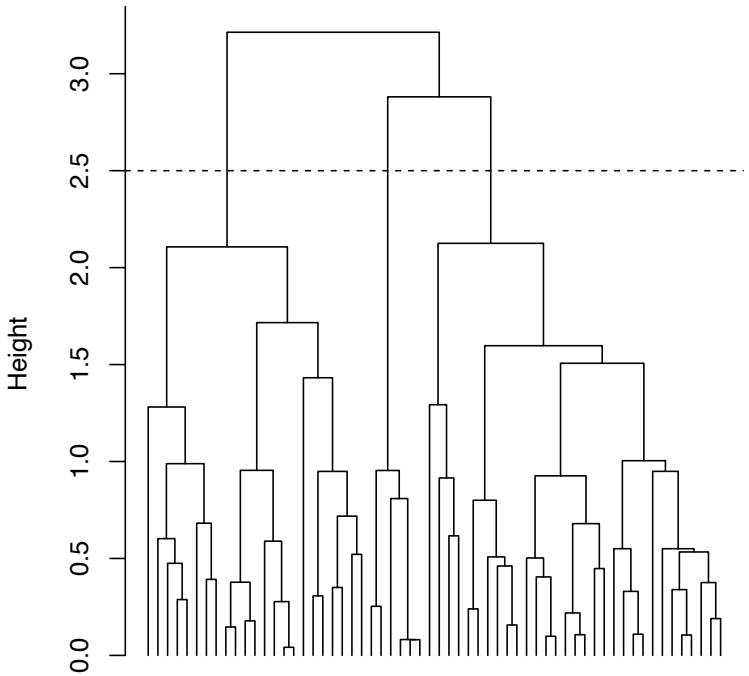
Source: Slides by Ryan Tibshirani

Average Linkage

Uses average of all pairs:

$$d_{\text{average}}(G, H) = \frac{1}{n_G \cdot n_H} \sum_{i \in G,\, j \in H} d_{ij}$$





Source: Slides by Ryan Tibshirani

Link Functions

Single linkage:

Uses maximum similarity (min distance) of pairs

Weakness: "straggly" (long and thin) clusters due to *chaining effect*

Clusters may not be compact

Complete linkage:

Uses minimum similarity (max distance) of pairs

Weakness: *crowding effect* (prefer small, dense clusters) + sensitivity to outliers

Clusters may not be far apart

Average linkage:

Uses average of all pairs

Tries to strike a balance – compact and far apart

Weakness: similarity more difficult to interpret

Scaling HAC

Isn't "big data" friendly

Need to keep all points in memory

Fundamentally, a sequential algorithm – difficult to parallelize

Isn't MapReduce/Spark friendly

Fundamentally, a sequential algorithm – difficult to parallelize

Not clear how to parallelize computation of which clusters to merge

Sequential dependence between merge steps

BIRCH

(Balanced Iterative Reducing and Clustering using Hierarchies)

Challenges w/ HAC

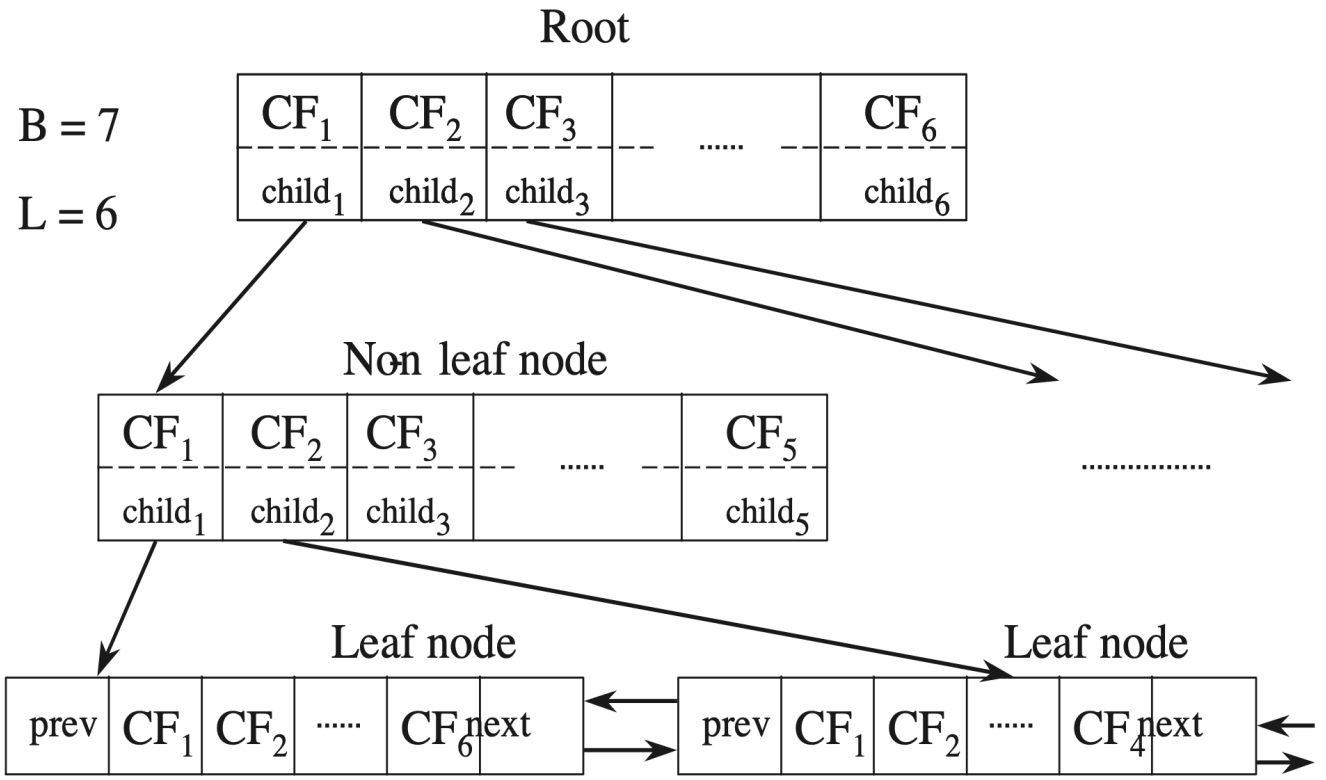
Must keep all data points in memory

Solution: only keep cluster information

Only allows binary branching

Solution: support larger branching factors (fan-out)

CF (Cluster Feature) Tree

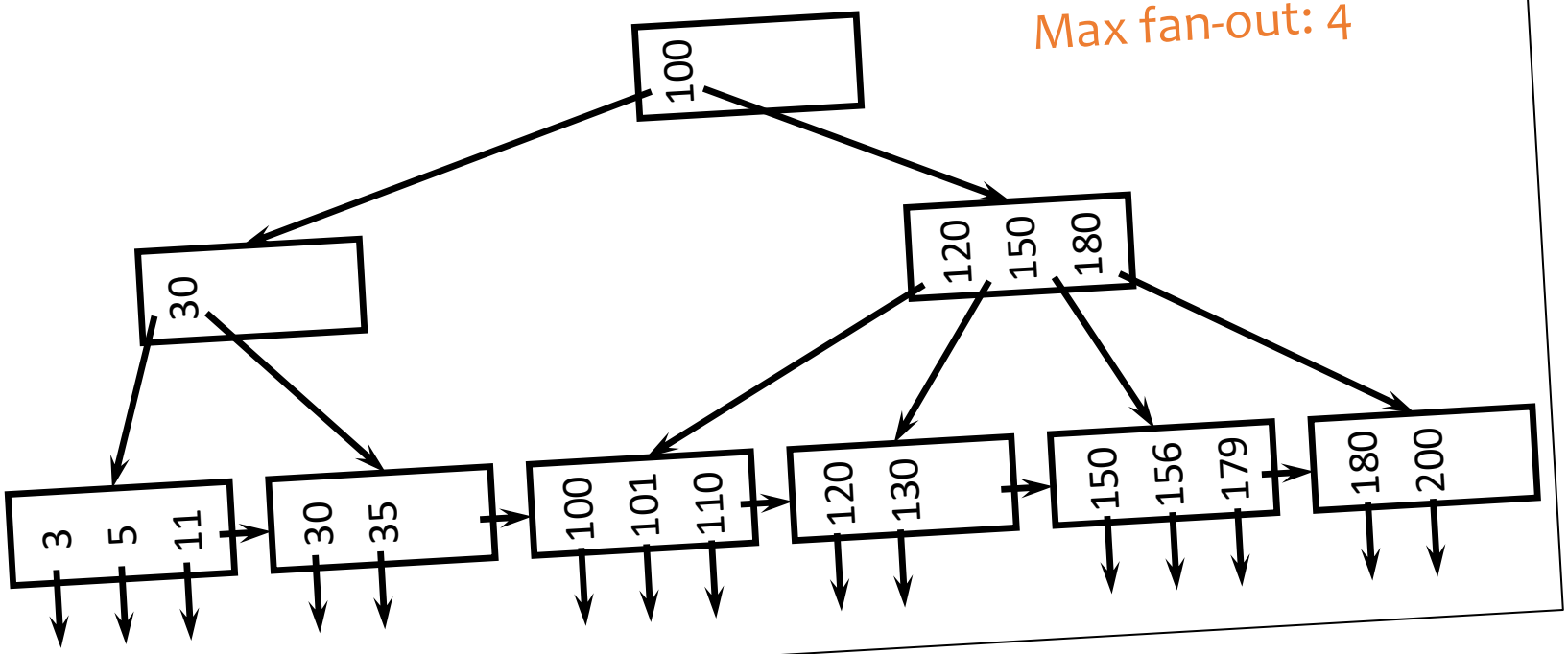


It's like a B+-Tree…

From CS 348…

B+-tree

- A hierarchy of nodes with intervals
- Balanced: good performance guarantee
- Disk-based: one node per page; large fan-out



Intuition: CF Trees

Keep sufficient statistics (CFs) on clusters

Answer "does this point belong to this cluster?" using only CFs

Compute distances + perform merges using only CFs

Only need to stream through all data points once!

Update CF if point belongs in the cluster

Add new CF if not, splitting nodes as necessary

Scaling BIRCH

Is "big data" friendly

No need to keep all points in memory

Too many CFs? Start over with larger clusters…

Isn't MapReduce/Spark friendly

Fundamentally, a sequential algorithm – difficult to parallelize

The CF tree represents global state

k-Means

K-Means Algorithm

Select *k* random instances $\{s_1, s_2, \ldots s_k\}$ as initial centroids

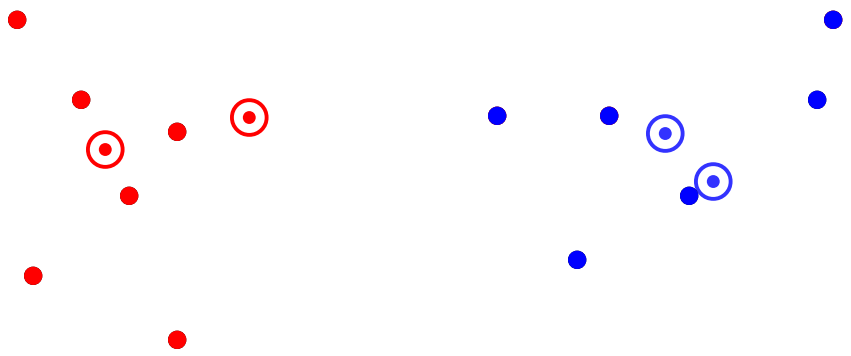
Iterate:

Assign each instance to closest centroid

Update centroids based on assigned instances

$$\mu(c) = \frac{1}{|c|} \sum_{\mathrm{x} \in c} \mathrm{x}$$

K-Means Clustering Example



Pick seeds

Reassign clusters

Compute centroids

Reassign clusters

Compute centroids

Reassign clusters

Converged!

Basic MapReduce Implementation

```
class Mapper  {
  def setup() = {
    clusters = loadClusters()
  }

  def map(id: Int, vector: Vector) = {
    context.write(clusters.findNearest(vector), vector)
  }
}

class Reducer {
  def reduce(clusterId: Int, values: Iterable[Vector]) = {
    for (vector <- values) {
      sum += vector
      cnt += 1
    }
    emit(clusterId, sum/cnt)
  }
}
```

$$\mu(c) = \frac{1}{|c|} \sum_{\mathrm{x} \in c} \mathrm{x}$$

Basic MapReduce Implementation

Conceptually, what's happening?

Given current cluster assignment, assign each vector to closest cluster

Group by cluster

Compute updated clusters

What's the cluster update?

Computing the mean!

What about Spark?

Implementation Notes

Standard setup of iterative MapReduce algorithms

- Driver program sets up MapReduce job
- Waits for completion
- Checks for convergence
- Repeats if necessary

Must be able keep cluster centroids in memory

- With large *k*, large feature spaces, potentially an issue
- Memory requirements of centroids grow over time!

Variant: *k*-medoids

How do you select initial seeds?

How do you select *k*?

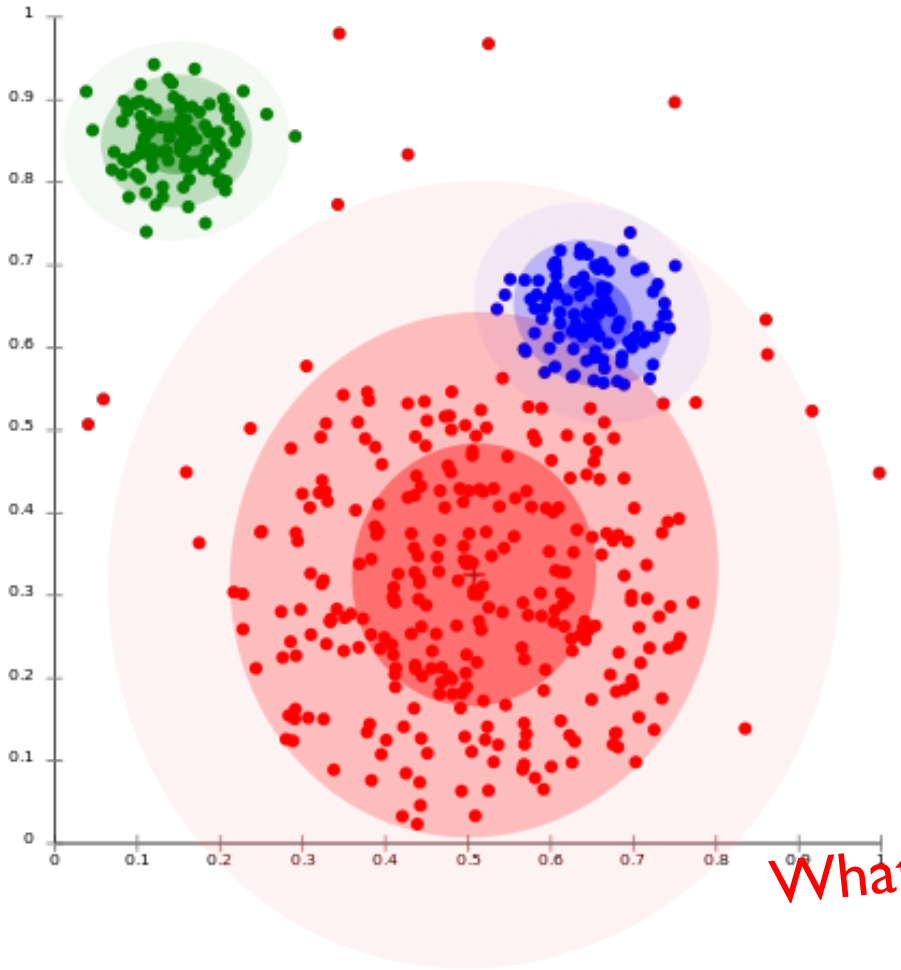
Gaussian Mixture Models

tl;dr – Don't be afraid of math
It's exactly like *k*-means
"fake it till you make it"

Clustering w/ Gaussian Mixture Models

Model data as a mixture of Gaussians

Given data, recover model parameters



What's with models?

Source: Wikipedia (Cluster analysis)

K-Means vs. GMMs

	K-Means	GMMs
Map	Compute distance of points to centroids	E-step: compute expectation of *z* indicator variables
Reduce	Recompute new centroids	M-step: update values of model parameters

Gaussian Distributions

Univariate Gaussian (i.e., Normal):

$$p(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$

A random variable with such a distribution we write as:

$$x \sim \mathcal{N}(\mu,\sigma^2)$$

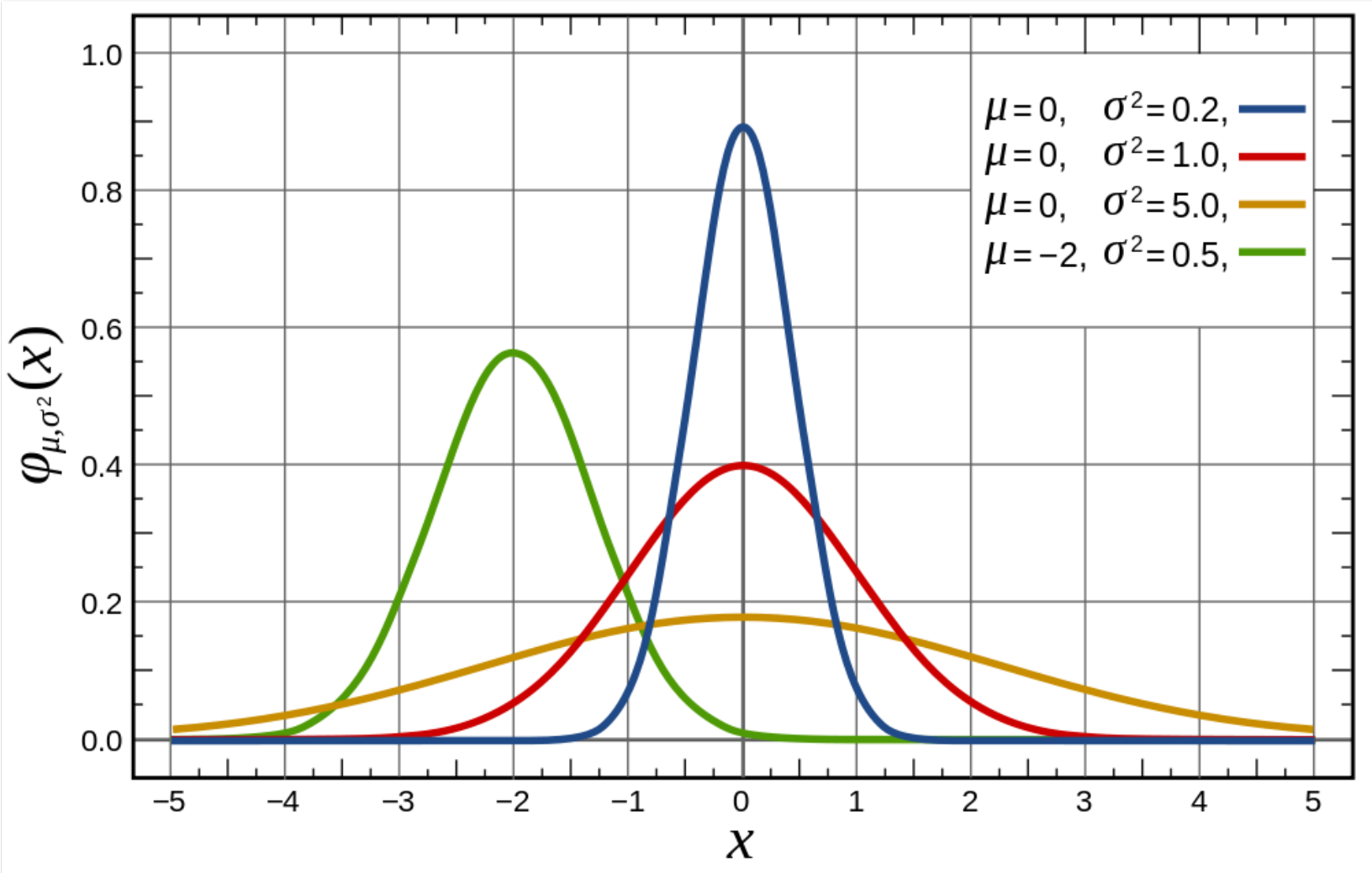
Multivariate Gaussian:

$$p(\mathrm{x};\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(\mathrm{x}-\mu)^T\Sigma^{-1}(\mathrm{x}-\mu)\right)$$

A random variable with such a distribution we write as:

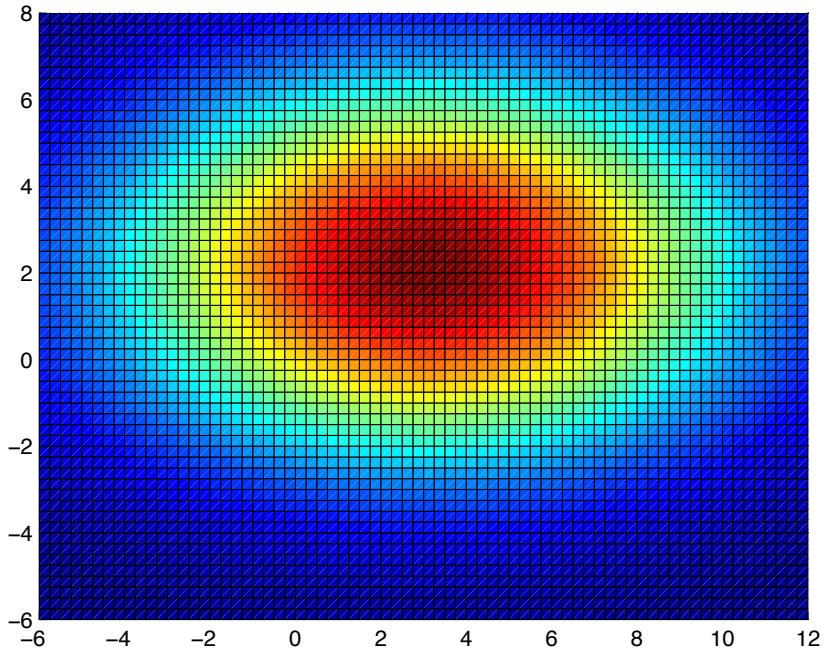
$$\mathrm{x} \sim \mathcal{N}(\mu,\Sigma)$$

Univariate Gaussian

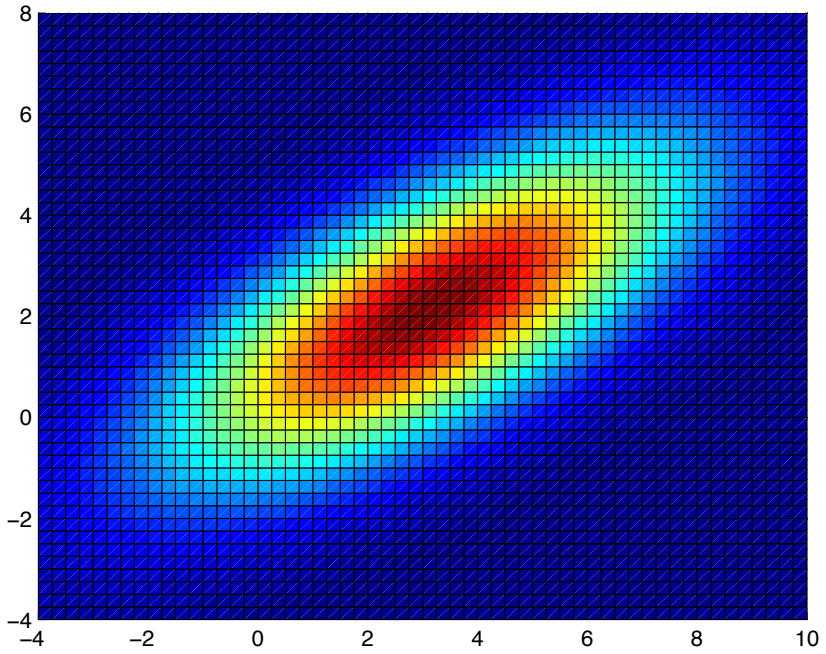


Source: Wikipedia (Normal Distribution)

Multivariate Gaussians



$$\mu = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 25 & 0 \\ 0 & 9 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 10 & 5 \\ 5 & 5 \end{bmatrix}$$

Source: Lecture notes by Chuong B. Do (IIT Delhi)

Gaussian Mixture Models

Model Parameters

Number of components: K

"Mixing" weight vector: π

For each Gaussian, mean and covariance matrix: $\mu_{1:K}$ $\Sigma_{1:K}$

The generative story?
(yes, that's a technical term)

Problem: Given the data, recover the model parameters

Varying constraints on co-variance matrices

Spherical vs. diagonal vs. full

Tied vs. untied

Learning for Simple Univariate Case

Problem setup:

Given number of components: K

Given points: $x_{1:N}$

Learn parameters: $\pi, \mu_{1:K}, \sigma^2_{1:K}$

Model selection criterion: maximize likelihood of data

Introduce indicator variables:

$$z_{n,k} = \begin{cases} 1 & \text{if } x_n \text{ is in cluster } k \\ 0 & \text{otherwise} \end{cases}$$

Likelihood of the data:

$$p(x_{1:N}, z_{1:N,1:K} | \mu_{1:K}, \sigma^2_{1:K}, \pi)$$

EM to the Rescue!

We're faced with this:

$$p(x_{1:N}, z_{1:N,1:K} | \mu_{1:K}, \sigma^2_{1:K}, \pi)$$

It'd be a lot easier if we knew the z's!

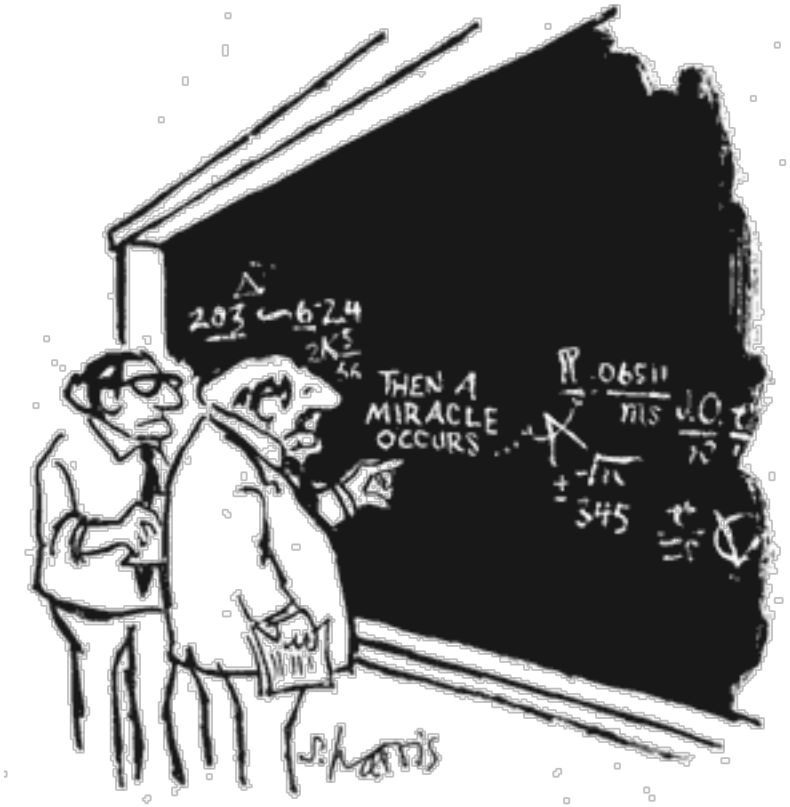
Expectation Maximization

Guess the model parameters

E-step: Compute posterior distribution over latent (hidden) variables given the model parameters

M-step: Update model parameters using posterior distribution computed in the E-step

Iterate until convergence



"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

EM for Univariate GMMs

Initialize: $\pi, \mu_{1:K}, \sigma^2_{1:K}$

Iterate:

E-step: compute expectation of z variables

$$\mathbb{E}[z_{n,k}] = \frac{\mathcal{N}(x_n|\mu_k, \sigma_k^2) \cdot \pi_k}{\sum_{k'} \mathcal{N}(x_n|\mu_{k'}, \sigma_{k'}^2) \cdot \pi_{k'}}$$

M-step: compute new model parameters

$$\pi_k = \frac{1}{N} \sum_n z_{n,k}$$

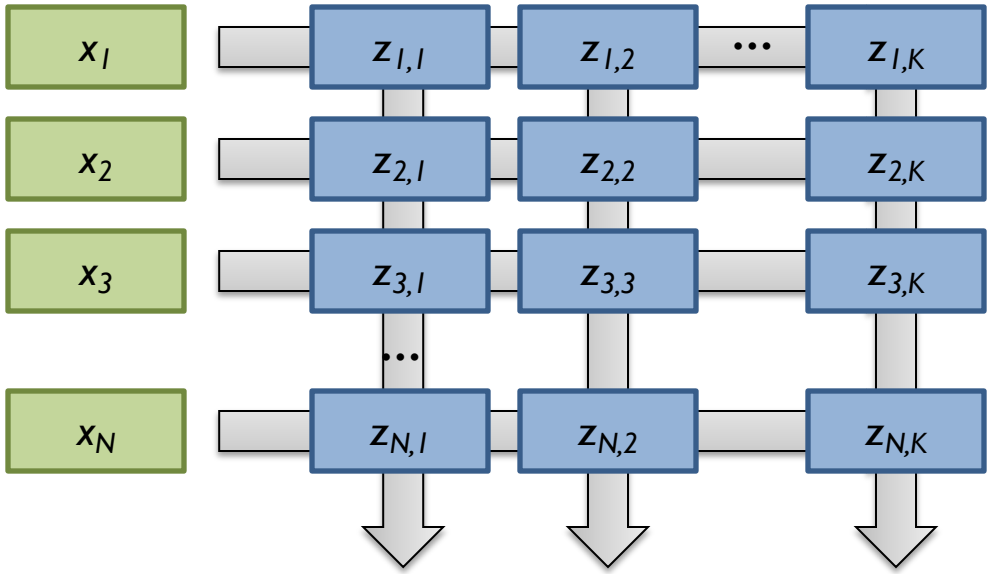
$$\mu_k = \frac{1}{\sum_n z_{n,k}} \sum_n z_{n,k} \cdot x_n$$

$$\sigma_k^2 = \frac{1}{\sum_n z_{n,k}} \sum_n z_{n,k} ||x_n - \mu_k||^2$$

MapReduce Implementation

Map

$$\mathbb{E}[z_{n,k}] = \frac{\mathcal{N}(x_n|\mu_k, \sigma_k^2) \cdot \pi_k}{\sum_{k'} \mathcal{N}(x_n|\mu_{k'}, \sigma_{k'}^2) \cdot \pi_{k'}}$$



Reduce

$$\pi_k = \frac{1}{N} \sum_n z_{n,k}$$

$$\mu_k = \frac{1}{\sum_n z_{n,k}} \sum_n z_{n,k} \cdot x_n$$

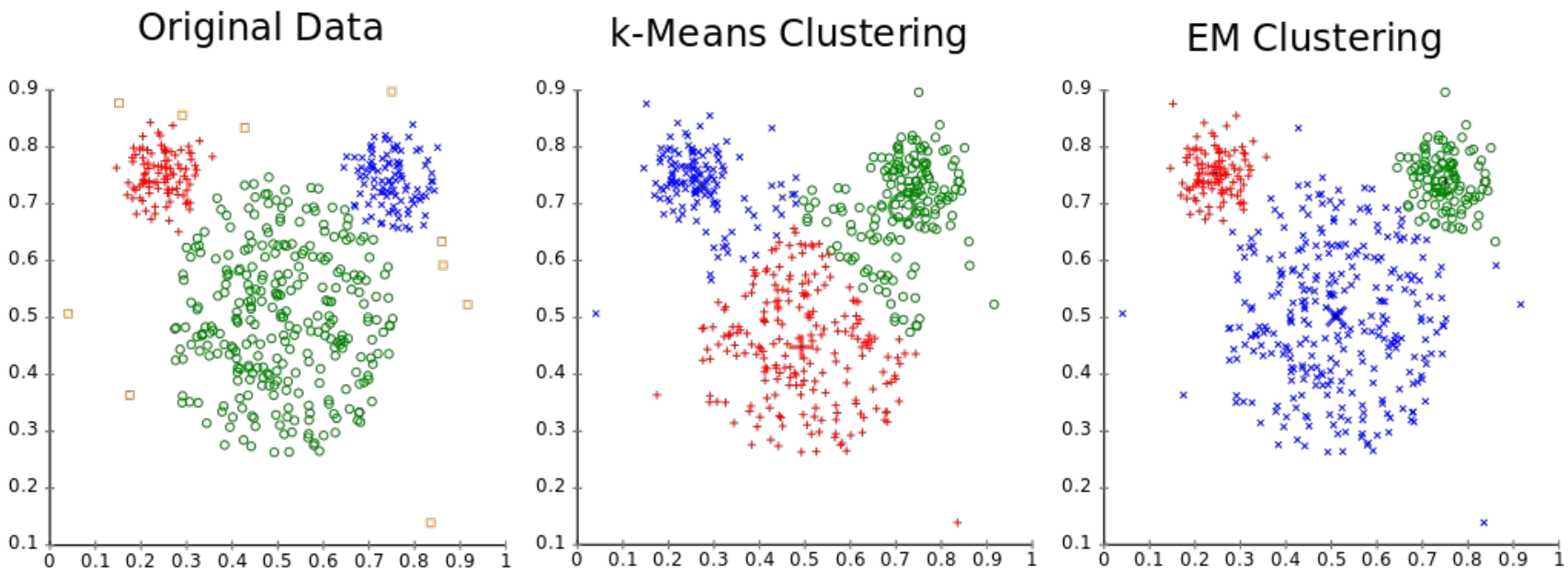
$$\sigma_k^2 = \frac{1}{\sum_n z_{n,k}} \sum_n z_{n,k} ||x_n - \mu_k||^2$$

What about Spark?

K-Means vs. GMMs

	K-Means	GMM
Map	Compute distance of points to centroids	E-step: compute expectation of *z* indicator variables = which cluster does this point belong to?
Reduce	Recompute new centroids	M-step: update values of model parameters = recompute new cluster mean, variance, etc.

Different cluster analysis results on "mouse" data set:



Source: Wikipedia (*k*-means clustering)

Key Questions

What makes certain types of clustering algorithms amenable to scale-out distributed processing?

What are the parallels between *k*-means clustering and Gaussian Mixture Models?


富嶽三十六景
神奈川沖
浪裏