UNIVERSITY OF WATERLOO

Big Data Infrastructure

CS 489/698 Big Data Infrastructure (Winter 2016)

Week 9: Data Mining (4/4)

March 10, 2016

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

These slides are available at http://lintool.github.io/bigdata-2016w/

What's the Problem?

- Arrange items into 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 problem

Compare/Contrast

- Finding similar items
  - Focus on individual items
- Clustering
  - Focus on groups of items
  - Relationship between items in a cluster is of interest

Evaluation?

- Classification
- Finding similar items
- Clustering

Clustering

Source: Wikipedia (Star cluster)

Clustering

- Specify distance metric
  - Jaccard, Euclidean, cosine, etc.
- Compute representation
  - Shingling, tf.idf, etc.
- Apply clustering algorithm

Distances

Source: www.flickr.com/photos/thiagoalmeida/250190676/

Distance Metrics

1. Non-negativity:

$$\mathrm{d}(\mathrm{x}, \mathrm{y}) \geq 0$$

2. Identity:

$$\mathrm{d}(\mathrm{x}, \mathrm{y}) = 0 \iff \mathrm{x} = \mathrm{y}$$

3. Symmetry:

$$\mathrm{d}(\mathrm{x}, \mathrm{y}) = \mathrm{d}(\mathrm{y}, \mathrm{x})$$

4. Triangle Inequality

$$\mathrm{d}(\mathrm{x}, \mathrm{y}) \leq \mathrm{d}(\mathrm{x}, \mathrm{z}) + \mathrm{d}(\mathrm{z}, \mathrm{y})$$

Distance: Jaccard

- Given two sets A, B
- Jaccard similarity:

$$\mathrm{J}(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

$$\mathrm{d}(A,B) = 1 - \mathrm{J}(A,B)$$

Distance: Hamming

- Given two bit vectors
- Hamming distance: number of elements which differ

Distance: Norms

- Given: $$\mathrm{x} = [x_1, x_2, \ldots x_n]$$ $$\mathrm{y} = [y_1, y_2, \ldots y_n]$$

- Euclidean distance (L_2-norm)

$$\mathrm{d}(\mathrm{x}, \mathrm{y}) = \sqrt{\sum_{i=0}^{n} (x_i - y_i)^2}$$

- Manhattan distance (L_1-norm)

$$\mathrm{d}(\mathrm{x}, \mathrm{y}) = \sum_{i=0}^{n} |x_i - y_i|$$

- L_r-norm

$$\mathrm{d}(\mathrm{x}, \mathrm{y}) = \left[\sum_{i=0}^{n} |x_i - y_i|^r \right]^{1/r}$$

Distance: Cosine

- Given: $\mathrm{x} = [x_1, x_2, \ldots x_n]$
  $\mathrm{y} = [y_1, y_2, \ldots y_n]$

- Idea: measure distance between the vectors

$$\cos\theta = \frac{\mathrm{x} \cdot \mathrm{y}}{|\mathrm{x}||\mathrm{y}|}$$

- Thus:

$$\mathrm{sim}(\mathrm{x},\mathrm{y}) = \frac{\sum_{i=0}^{n} x_i y_i}{\sqrt{\sum_{i=0}^{n} x_i^2}\sqrt{\sum_{i=0}^{n} y_i^2}}$$

$$\mathrm{d}(\mathrm{x},\mathrm{y}) = 1 - \mathrm{sim}(\mathrm{x},\mathrm{y})$$

Advantages over others?

Representations

Representations: Text

- Unigrams (i.e., words)
- Shingles = *n*-grams
  - At the word level
  - At the character level
- Feature weights
  - boolean
  - tf.idf
  - BM25
  - …

Representations: Beyond Text

- For recommender systems:
  - Items as features for users
  - Users as features for items
- For graphs:
  - Adjacency lists as features for vertices
- With log data:
  - Behaviors (clicks) as features

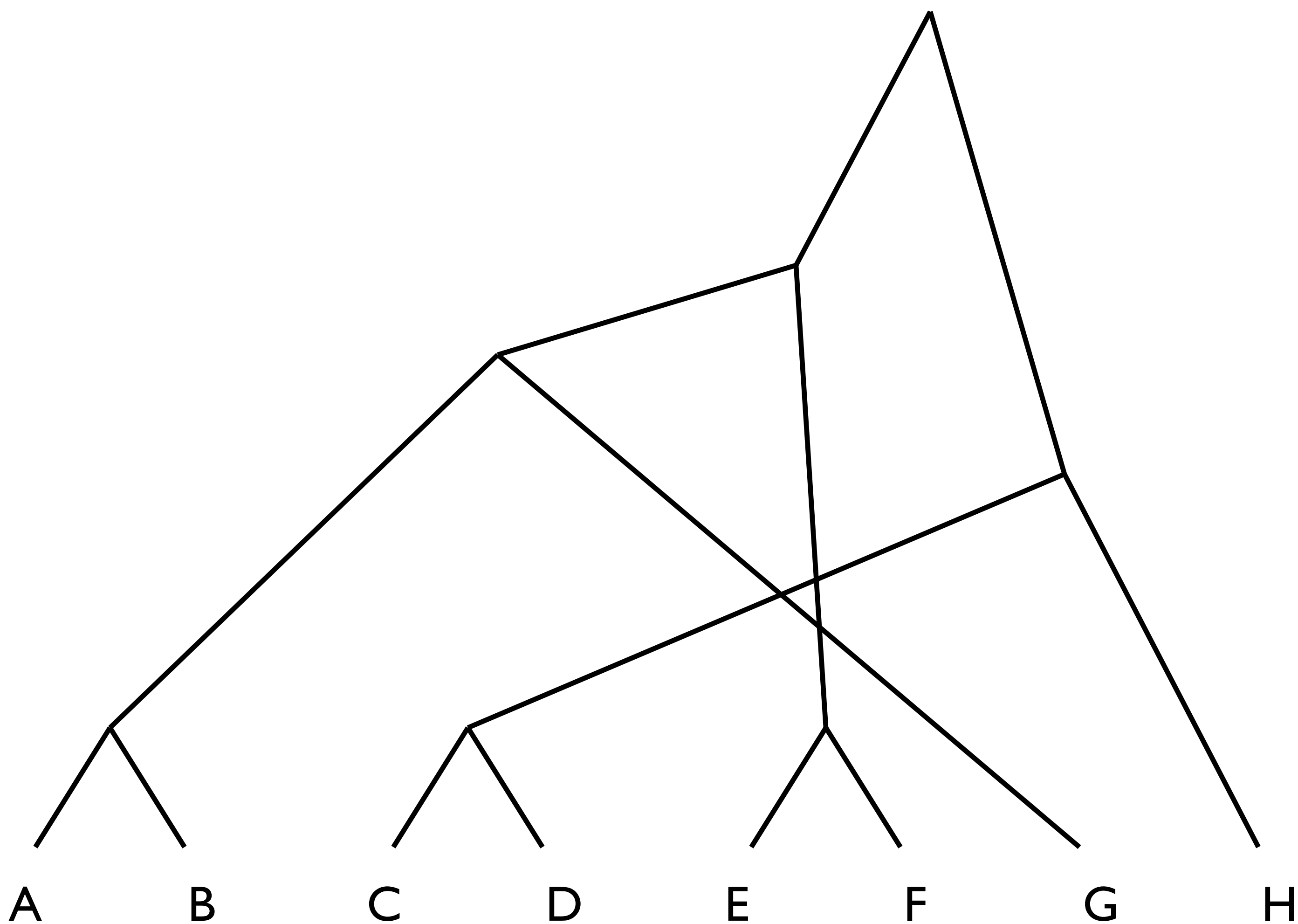
General Clustering Approaches

- Hierarchical
- *K*-Means
- Gaussian Mixture Models

Hierarchical Agglomerative Clustering

- Start with each document 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



Cluster Merging

- Which two clusters do we merge?
- What's the similarity between two clusters?
  - Single Link: similarity of two most similar members
  - Complete Link: similarity of two least similar members
  - Group Average: average similarity between members

Link Functions

- Single link:
  - Uses maximum similarity of pairs:

$$\text{sim}(c_i, c_j) = \max_{x \in c_i, y \in c_j} \text{sim}(x, y)$$

  - Can result in "straggly" (long and thin) clusters due to *chaining effect*
- Complete link:
  - Use minimum similarity of pairs:

$$\text{sim}(c_i, c_j) = \min_{x \in c_i, y \in c_j} \text{sim}(x, y)$$

  - Makes more "tight" spherical clusters

MapReduce Implementation

- What's the inherent challenge?

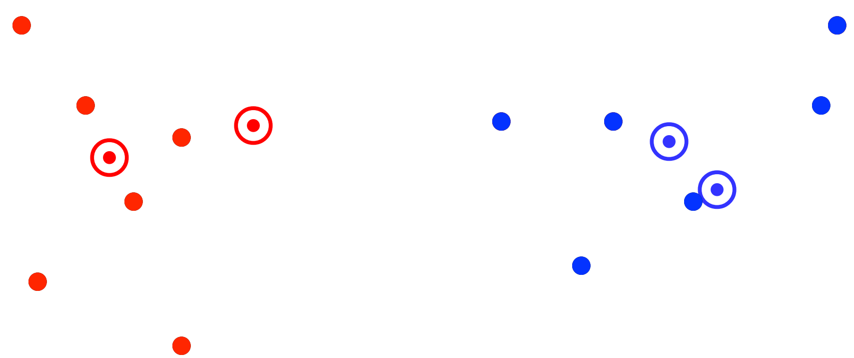
K-Means Algorithm

- Let d be the distance between documents
- Define the centroid of a cluster to be:

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

- Select k random instances $\{s_1, s_2, \ldots s_k\}$ as seeds.
- Until clusters converge:
    - Assign each instance x_i to the cluster c_j such that $d(x_i, s_j)$ is minimal
    - Update the seeds to the centroid of each cluster
    - For each cluster c_j, $s_j = \mu(c_j)$

K-Means Clustering Example



Pick seeds

Reassign clusters

Compute centroids

Reassign clusters

Compute centroids

Reassign clusters

Converged!

Basic MapReduce Implementation

```
class Mapper
    method Configure()
        c ← LoadClusters()
    method Map(id i, point p)
        n ← NearestClusterID(clusters c, point p)
        p ← ExtendPoint(point p)
        Emit(clusterid n, point p)
class Reducer
    method Reduce(clusterid n, points [p1, p2, ...])
        s ← InitPointSum()
        for all point p ∈ points do
            s ← s + p
        m ← ComputeCentroid(point s)
        Emit(clusterid n, centroid m)
```

(Just a clever way to keep track of denominator)

MapReduce Implementation w/ IMC

```
class Mapper
    method Configure()
        c ← LoadClusters()
        H ← InitAssociativeArray()
    method Map(id i, point p)
        n ← NearestClusterID(clusters c, point p)
        p ← ExtendPoint(point p)
        H{n} ← H{n} + p
    method Close()
        for all clusterid n ∈ H do
            Emit(clusterid n, point H{n})
class Reducer
    method Reduce(clusterid n, points [p1, p2, ...])
        s ← InitPointSum()
        for all point p ∈ points do
            s ← s + p
        m ← ComputeCentroid(point s)
        Emit(clusterid n, centroid m)
```

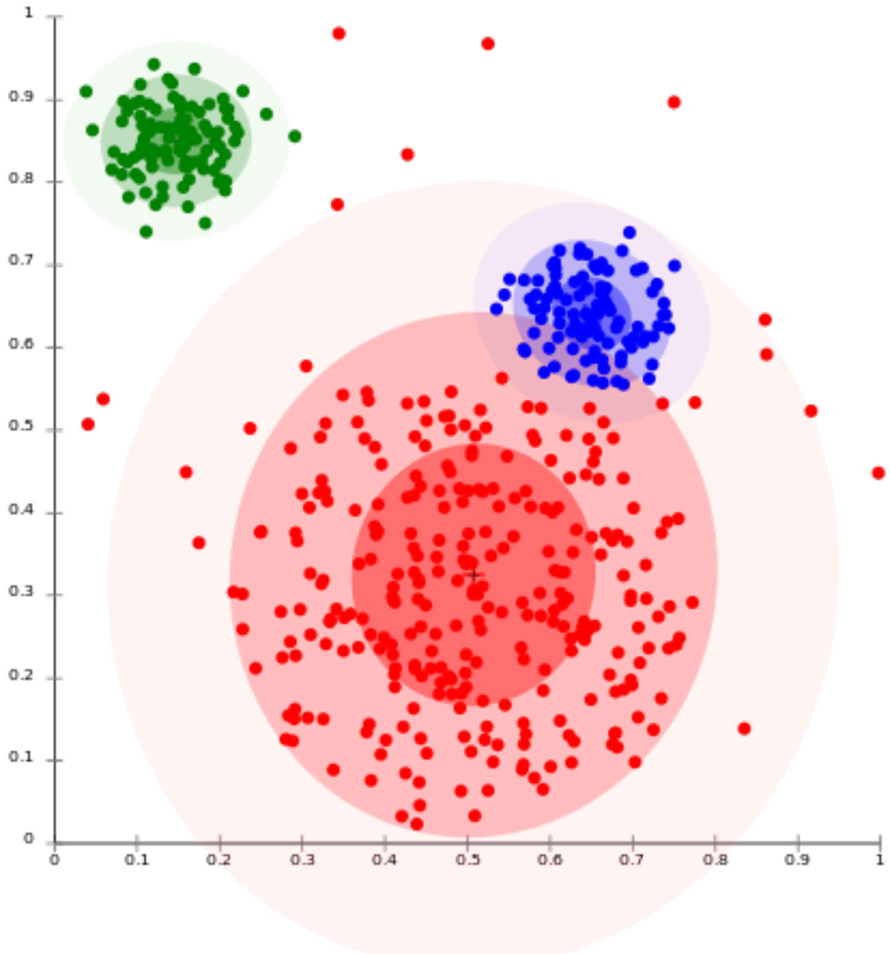
What about Spark?

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

Clustering w/ Gaussian Mixture Models

- Model data as a mixture of Gaussians
- Given data, recover model parameters



Source: Wikipedia (Cluster analysis)

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:

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

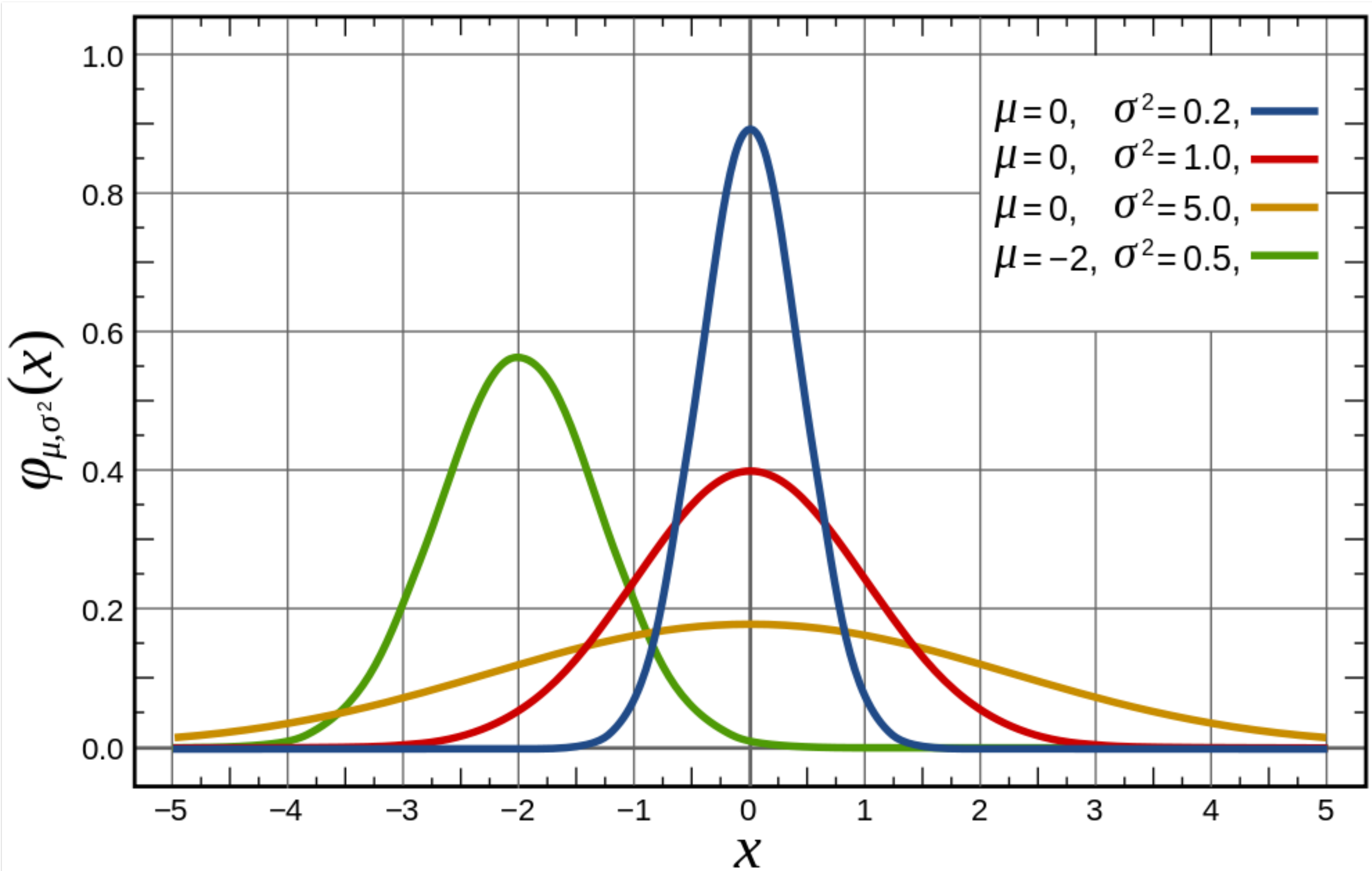
- Multivariate Gaussian:

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

  - A vector-value random variable with such a distribution we write as:

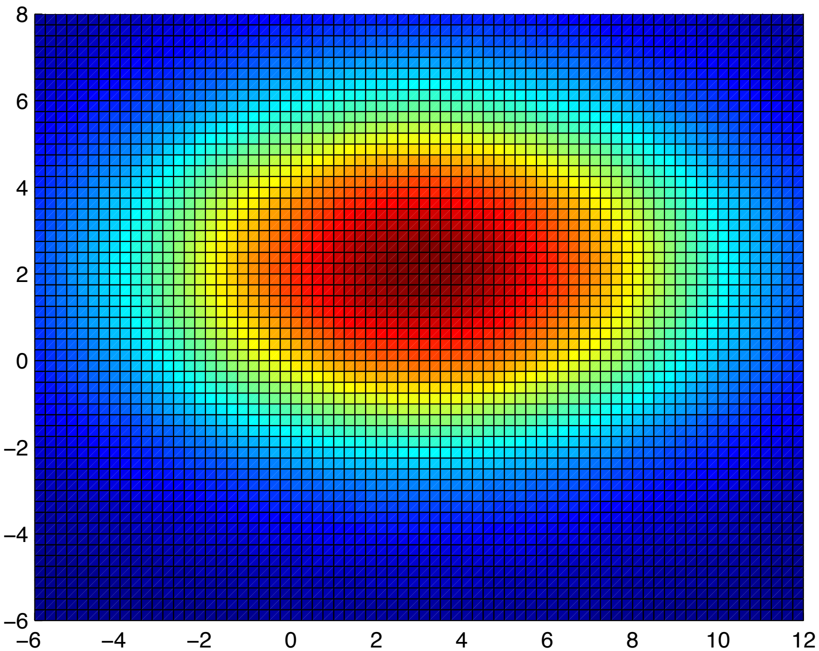
    $\mathrm{x} \sim \mathcal{N}(\mu, \Sigma)$

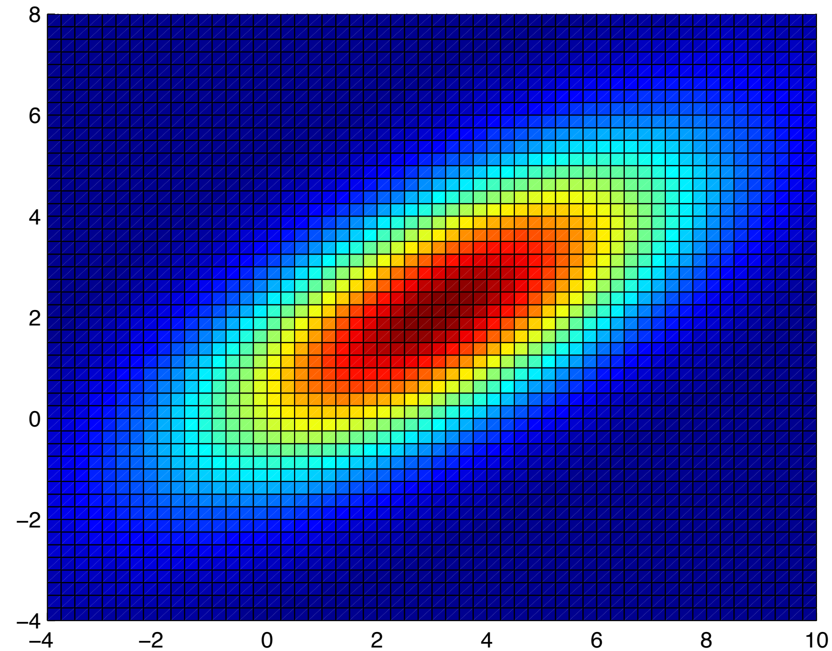
Univariate Gaussian



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: $\mu_{1:K}$ $\Sigma_{1:K}$
- Varying constraints on co-variance matrices
  - Spherical vs. diagonal vs. full
  - Tied vs. untied

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

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:

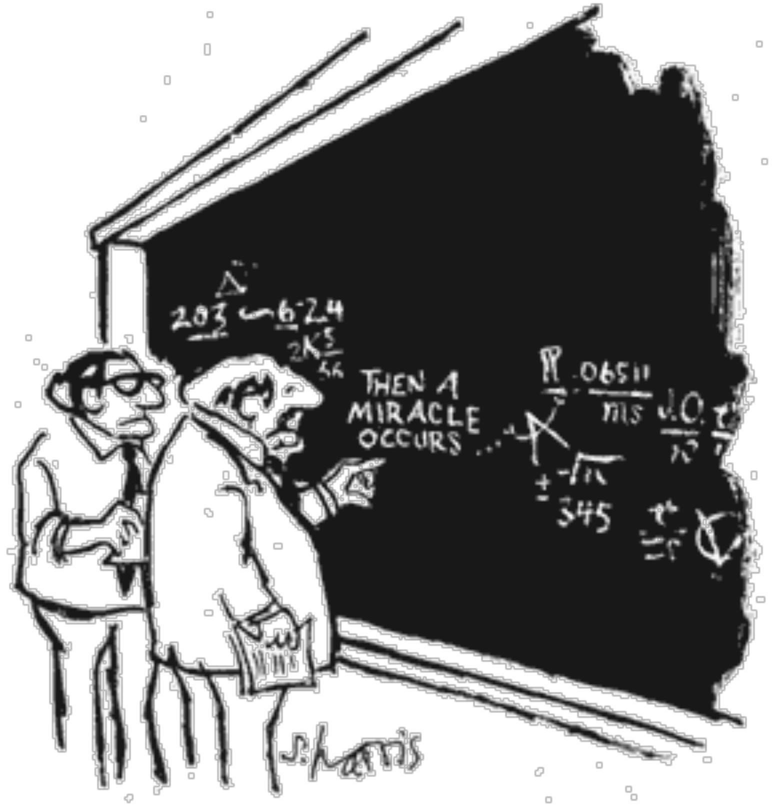
$$p(x_{1:N}, z_{1:N,1:K} | \mu_{1:K}, \sigma^2_{1:K}, \pi)$$

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

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

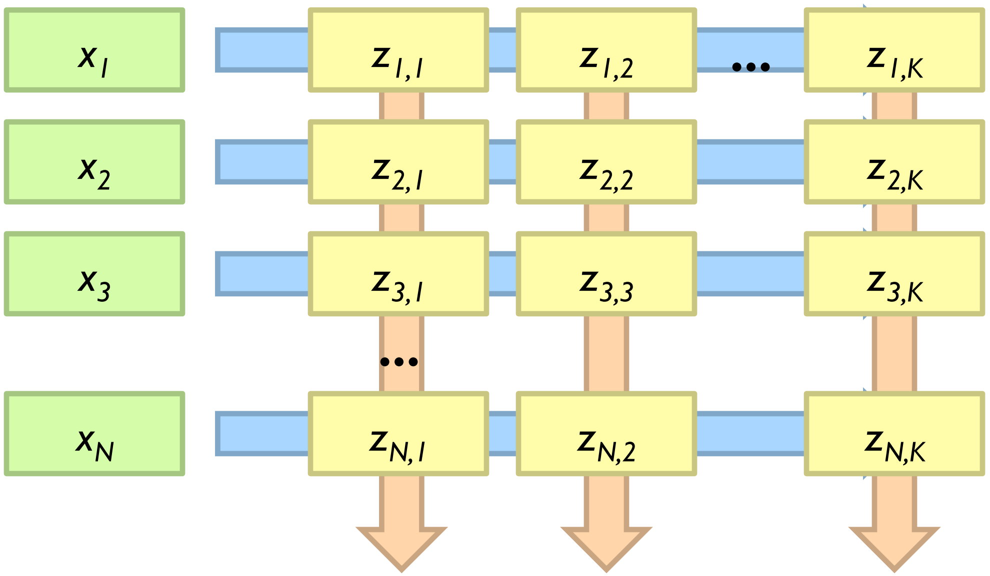
$$\mu_k = \frac{1}{\sum_n z_{n,k}} \sum_n z_{n,k} \cdot x_n$$

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

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

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

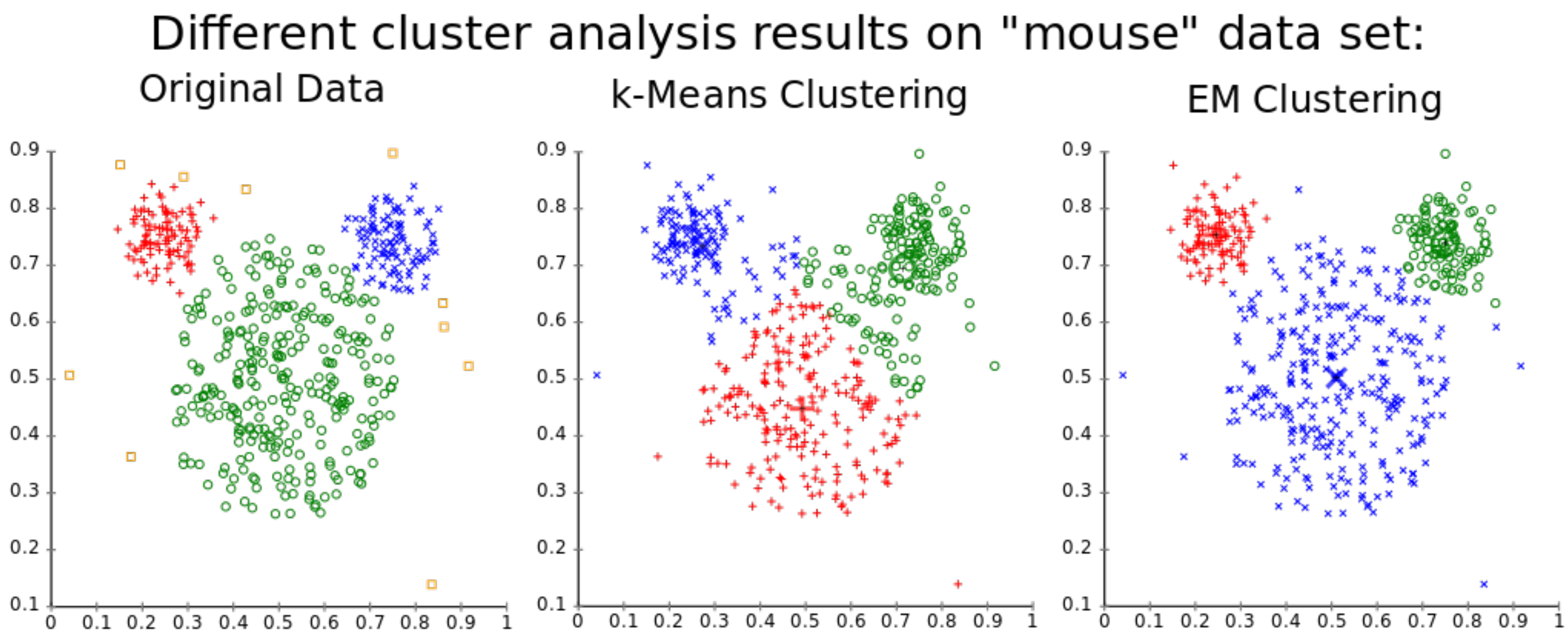
$$\mu_k = \frac{1}{\sum_n z_{n,k}} \sum_n z_{n,k} \cdot x_n$$

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

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
Reduce	Recompute new centroids	M-step: update values of model parameters



Source: Wikipedia (*k*-means clustering)

Source: Wikipedia (Japanese rock garden)