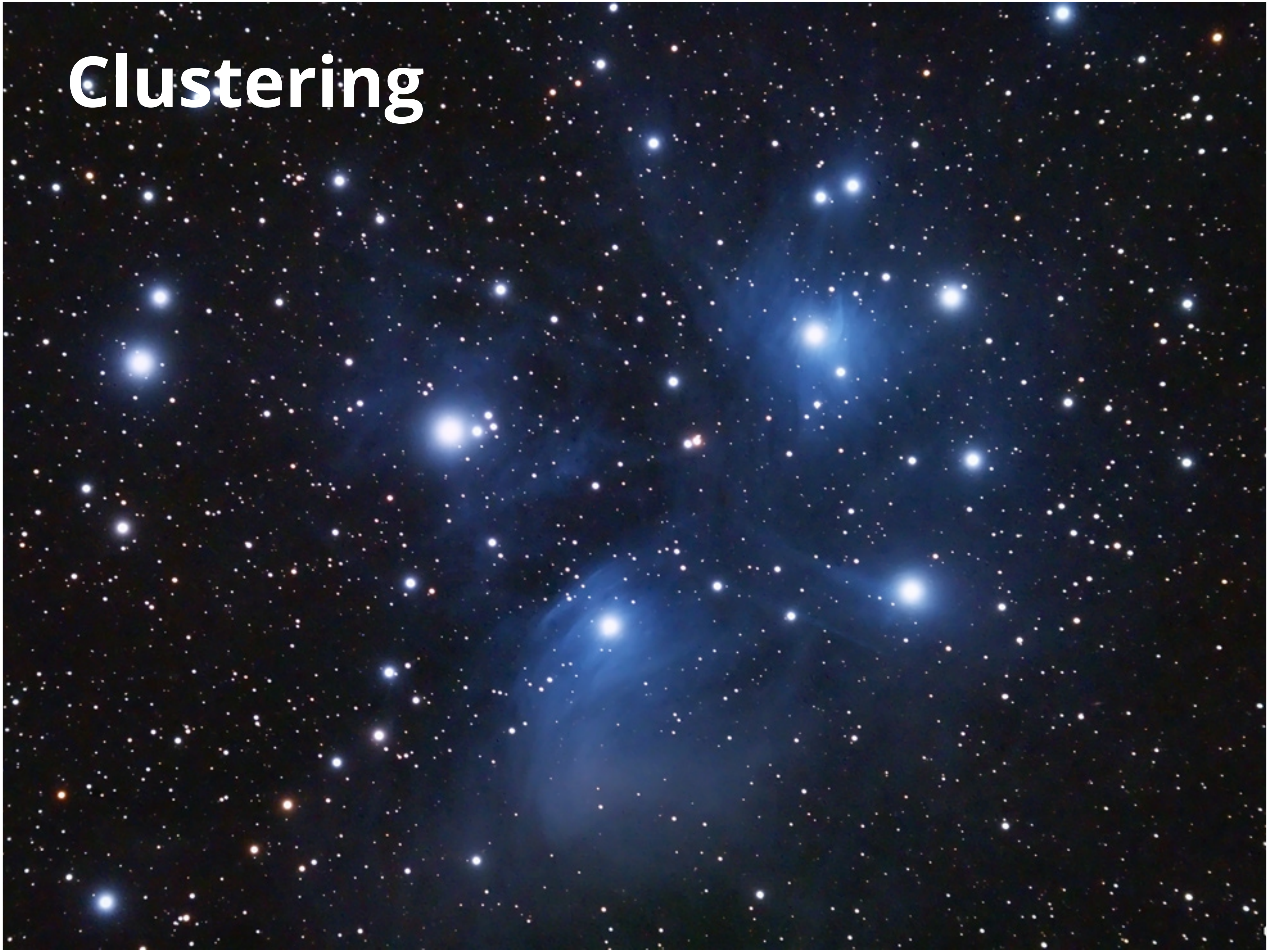
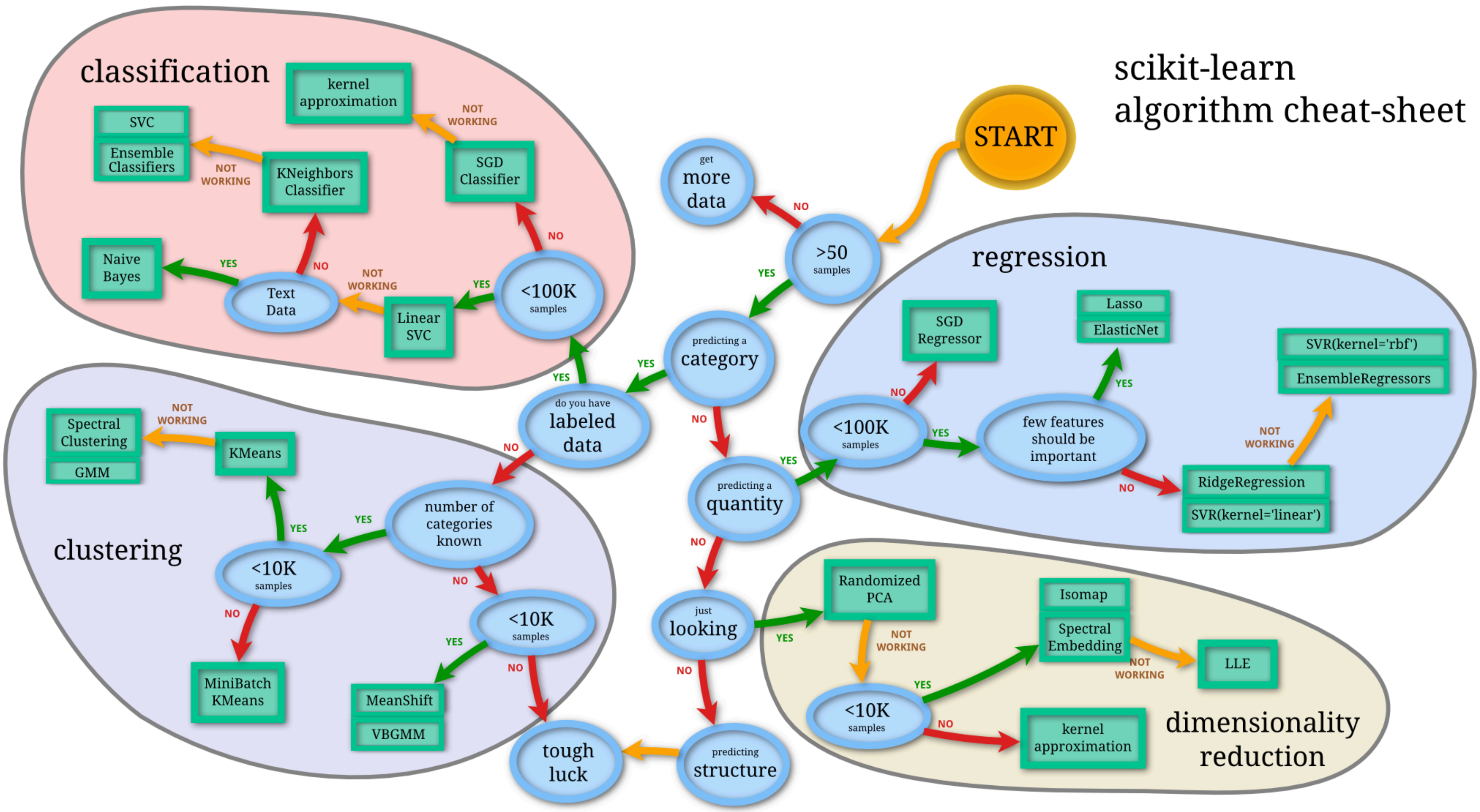
Search Engine Architecture

8. Clustering

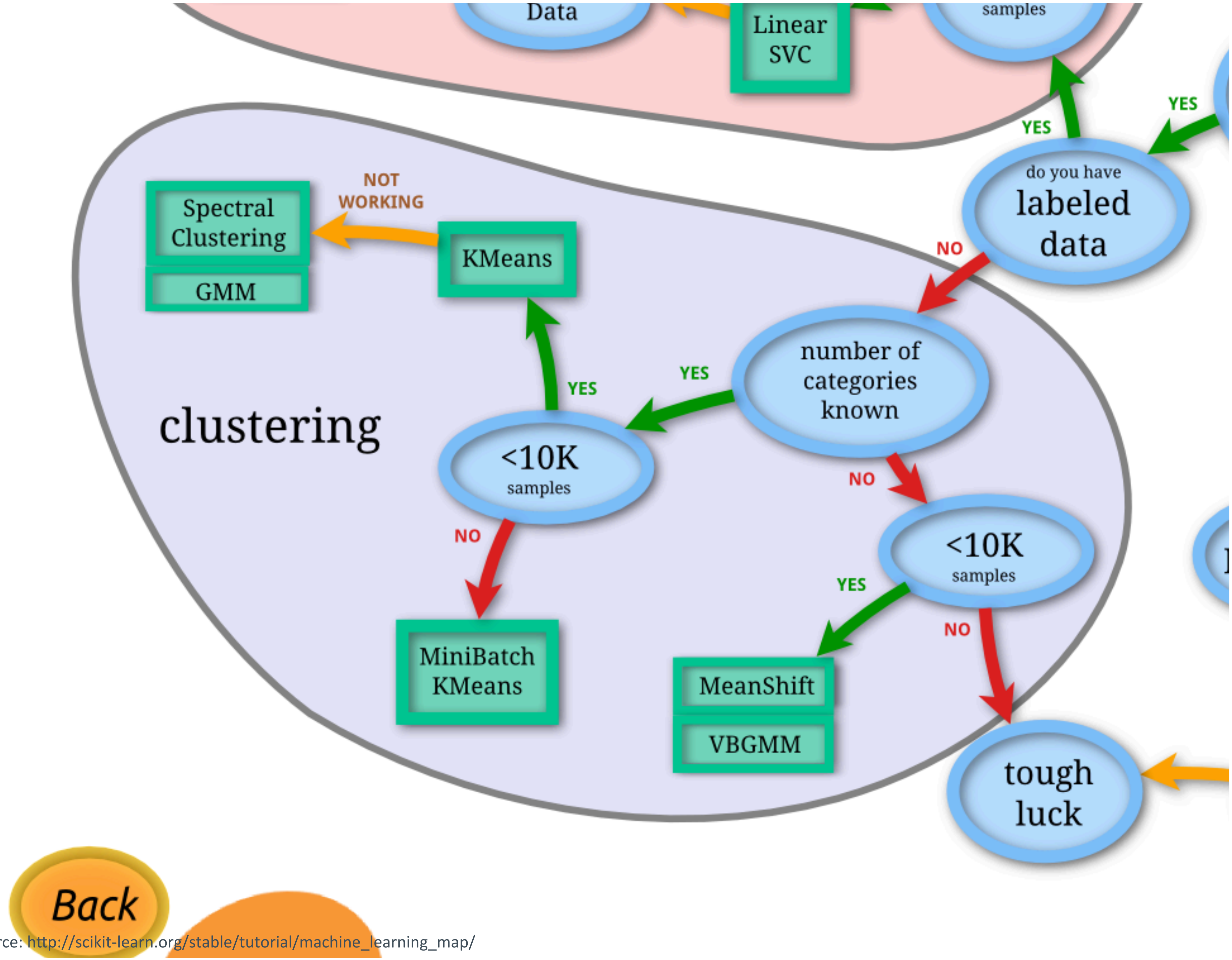
Noted slides adapted from: Lin et al.'s Big Data Infrastructure, UMD Spring 2015 with cosmetic changes.

Clustering





Source: http://scikit-learn.org/stable/tutorial/machine_learning_map/



Source: http://scikit-learn.org/stable/tutorial/machine_learning_map/

Problem Setup

- Arrange items into clusters
  - High similarity between objects in the same cluster
  - Low similarity between objects in different clusters

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Applications

- Exploratory analysis of large collections of objects
- Image segmentation
- Recommender systems
- Cluster hypothesis in information retrieval
- Computational biology and bioinformatics
- Anomaly detection
- Pre-processing for many other algorithms

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Three Approaches

- Hierarchical clustering
- *K*-means clustering
- Gaussian mixture models

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

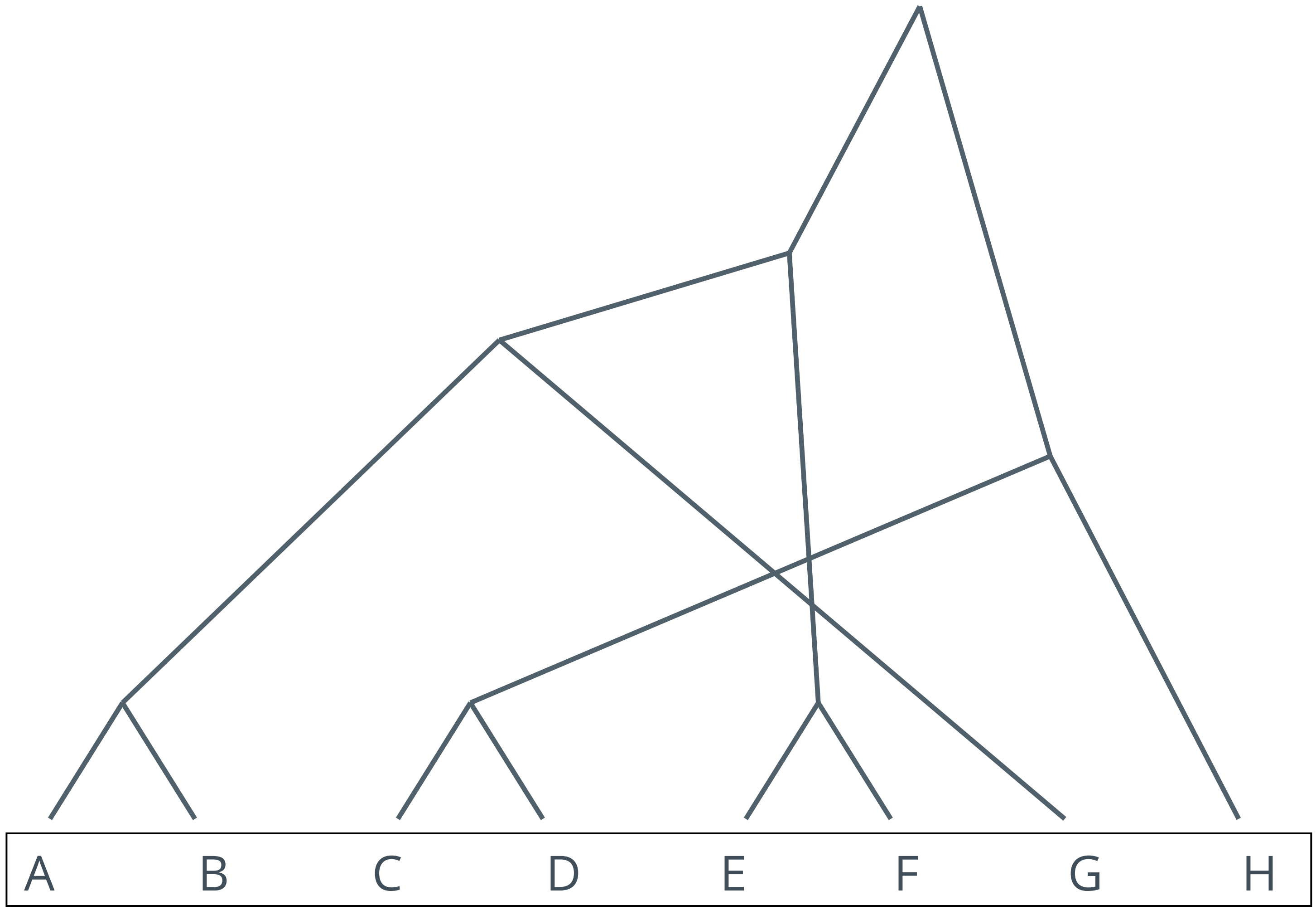
Hierarchical Clustering

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

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

HAC in Action



Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

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

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

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

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

MapReduce Implementation

- What's the inherent challenge?
- One possible approach:
  - Iteratively use fast heuristic to group together similar items
  - When dataset is small enough, run HAC in memory on a single machine
  - Observation: structure at the leaves is not very important

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

K-Means Clustering

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, $s_j = \mu(c_j)$

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

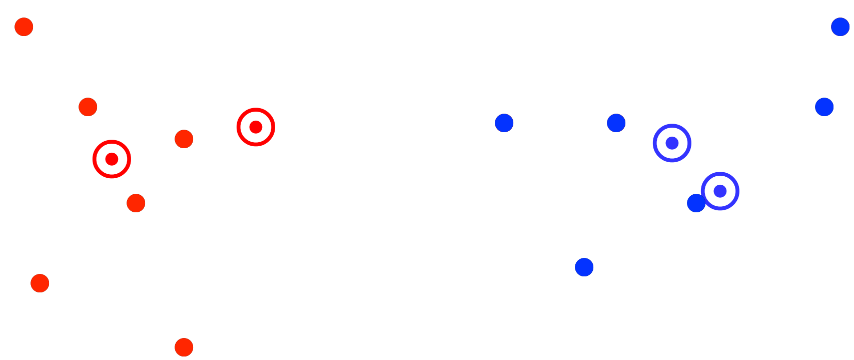
Basic MapReduce Implementation

```
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)
```

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

K-Means Clustering Example



Pick seeds

Reassign clusters

Compute centroids

Reassign clusters

Compute centroids

Reassign clusters

Converged!

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

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 [p_1, p_2, ...])
        s ← InitPointSum()
        for all point p ∈ points do
            s ← s + p
        m ← ComputeCentroid(point s)
        Emit(clusterid n, centroid m)
```

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Implementation Notes

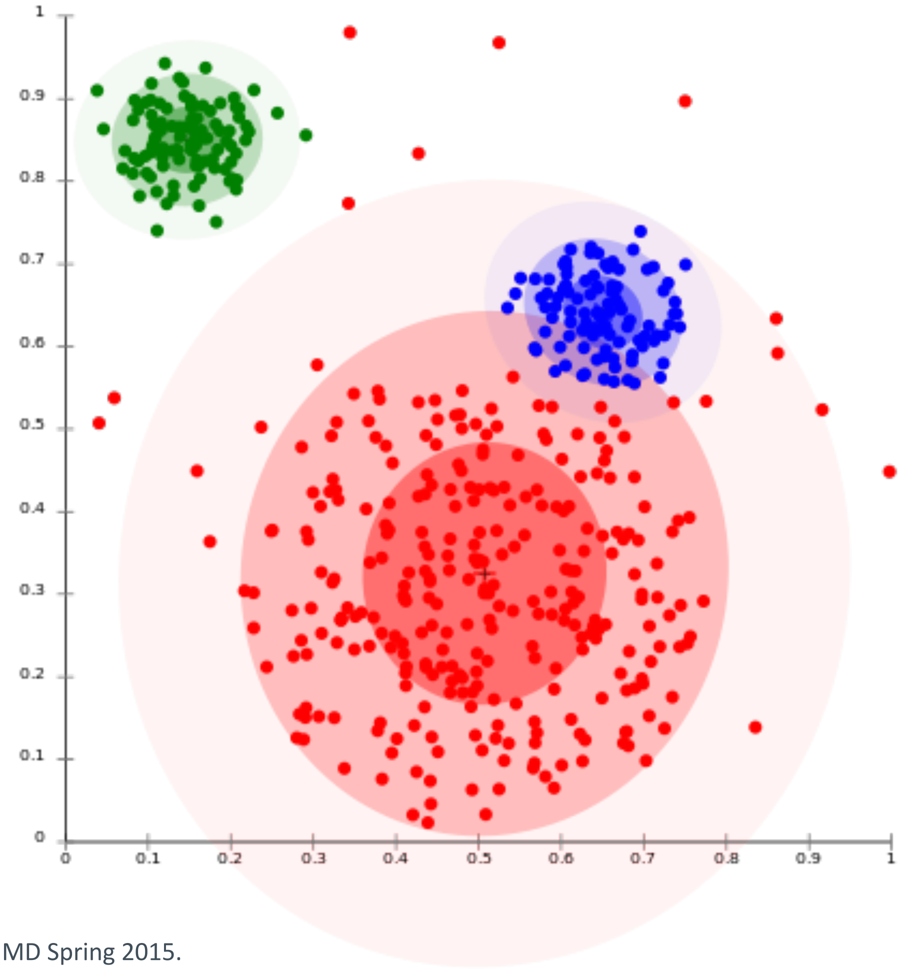
- Standard setup of iterative MapReduce algorithms
  - Driver program sets up MapReduce job
  - Waits for completion
  - Checks for convergence
  - Repeats if necessary
- Must be able to keep cluster centroids in memory
  - With large *k*, large feature spaces, potentially an issue
  - Memory requirements of centroids grow over time!
- Variant: *k*-medoids

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Gaussian Mixture Models

Clustering w/ Gaussian Mixture Models

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



Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

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:

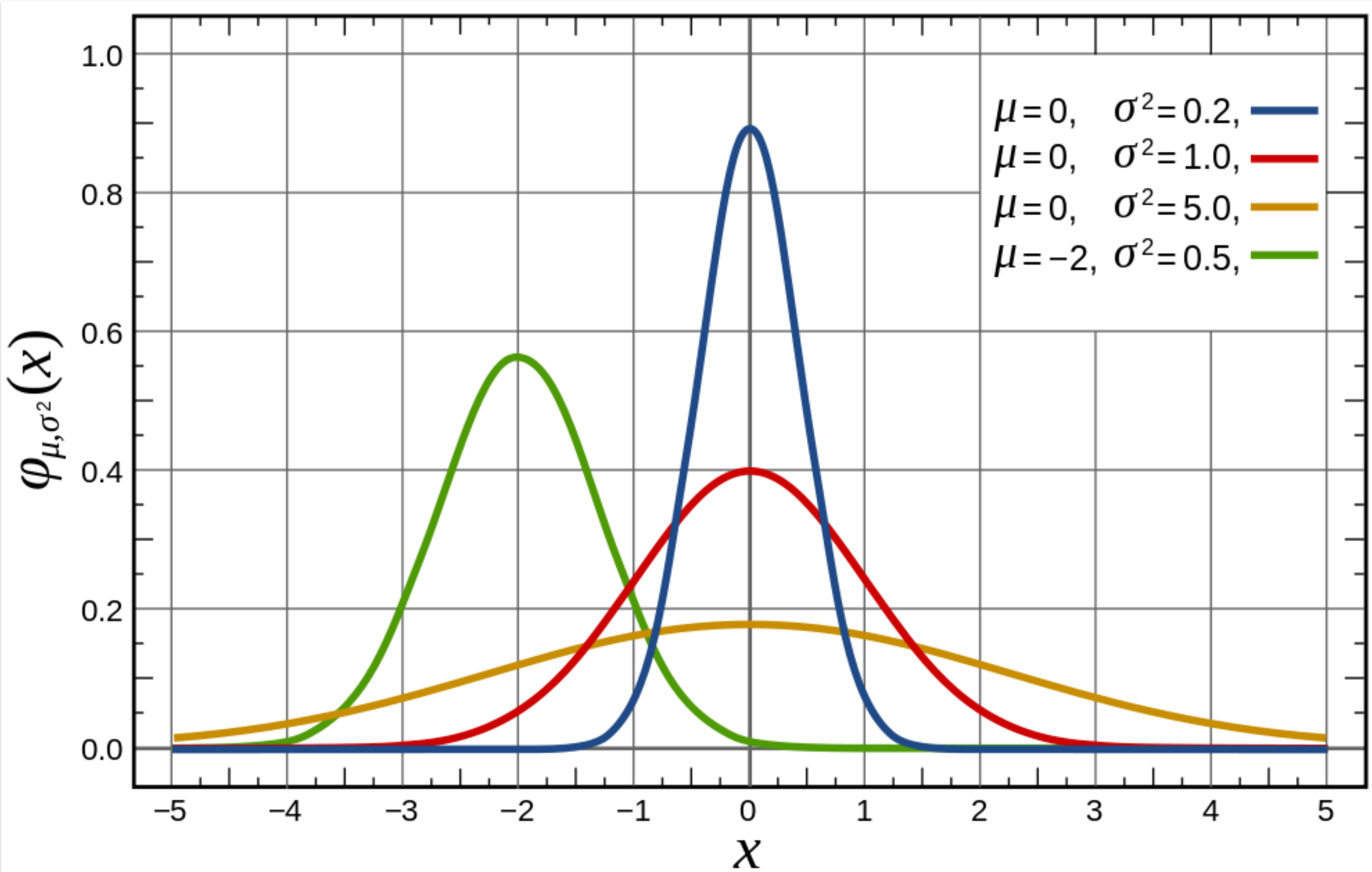
$$p(\mathrm{x}; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathrm{x}-\mu)^T \Sigma^{-1}(\mathrm{x}-\mu)\right)$$

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

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

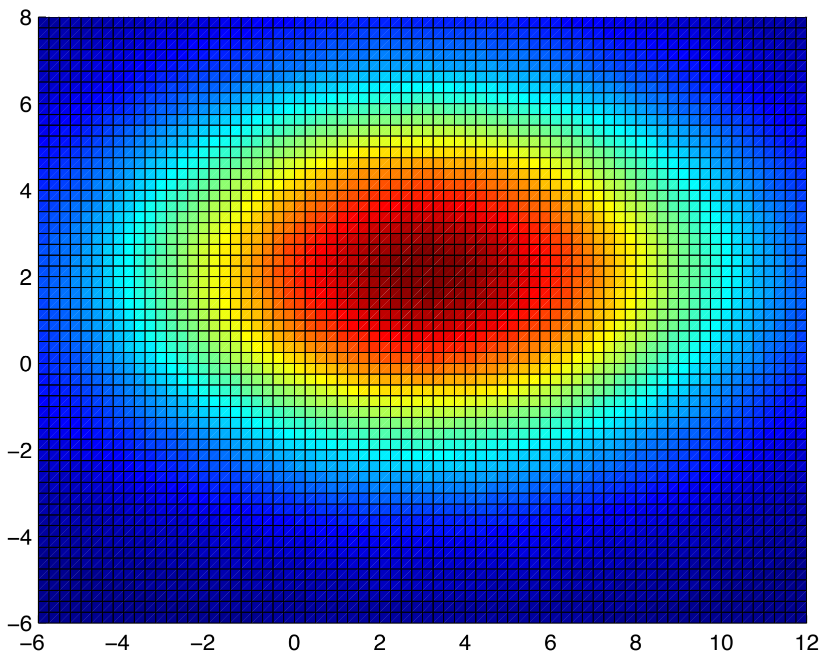
Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

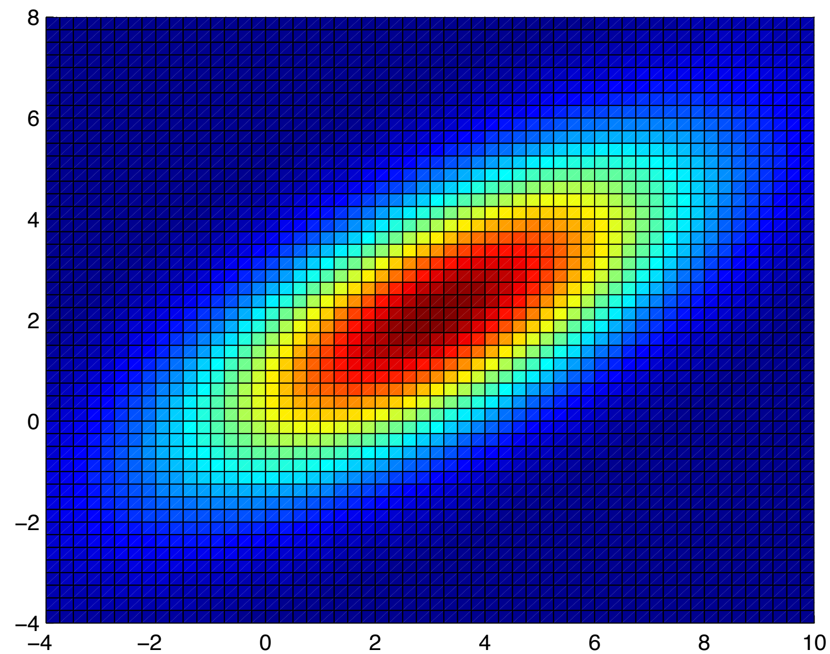
Univariate Gaussian



Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

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: Lin et al. Big Data Infrastructure, UMD Spring 2015.

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

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

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:

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

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

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

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

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

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

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

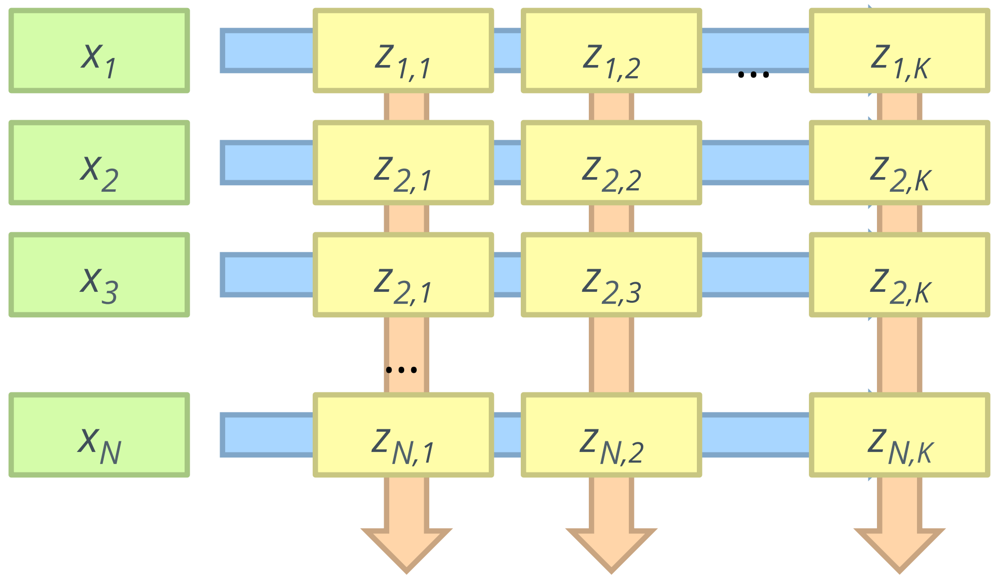
$$\sigma_k^2 = \frac{1}{\sum_n z_{n,k}} \sum_n z_{n,k} ||x_n - \mu_k||^2$$

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

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

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

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

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

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

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: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Summary

- Hierarchical clustering
  - Difficult to implement in MapReduce
- *K*-Means
  - Straightforward implementation in MapReduce
- Gaussian Mixture Models
  - Implementation conceptually similar to *k*-means, more “bookkeeping”

Source: Lin et al. Big Data Infrastructure, UMD Spring 2015.

Questions?