Search Engine Architecture

8. Clustering
Problem Setup

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

Applications

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

Three 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?
• 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

K-Means Algorithm

- Let $d$ be the distance between documents
- Define the centroid of a cluster to be:
  \[ \mu(c) = \frac{1}{|c|} \sum_{x \in c} x \]
- 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 $c_j$, $s_j = \mu(c_j)$
Basic MapReduce Implementation

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

$K$-Means Clustering Example

Pick seeds
Reassign clusters
Compute centroids
Reassign clusters
Compute centroids
Reassign clusters
Converged!

MapReduce Implementation w/ IMC

1: class Mapper
2:   method Configure()
3:     c ← LoadClusters()
4:     H ← InitAssociativeArray()
5:   method Map(id i, point p)
6:     n ← NearestClusterID(clusters c, point p)
7:     p ← ExtendPoint(point p)
8:     H{n} ← H{n} + p
9:   method Close()
10:   for all clusterid n ∈ H  do
11:     Emit(clusterid n, point H{n})
1: class Reducer
2:   method Reduce(clusterid n, points [p1, p2, ...])
3:     s ← InitPointSum()
4:     for all point p ∈ points  do
5:       s ← s + p
6:     m ← ComputeCentroid(point s)
7:     Emit(clusterid n, centroid m)

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

Clustering w/ Gaussian Mixture Models

- Model data as a mixture of Gaussians
- Given data, recover 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:**
  \[
p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp \left( -\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu) \right)
  \]
  A vector-value random variable with such a distribution we write as:
  \[
x \sim \mathcal{N}(\mu, \Sigma)
  \]

Univariate Gaussian

Multivariate Gaussians

Figure 2:
The figure on the left shows a heatmap indicating values of the density function for an axis-aligned multivariate Gaussian with mean $\mu = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$ and diagonal covariance matrix $\Sigma = \begin{bmatrix} 25 & 0 \\ 0 & 9 \end{bmatrix}$. Notice that the Gaussian is centered at $(3, 2)$, and that the isocontours are all elliptically shaped with major/minor axis lengths in a 5:3 ratio. The figure on the right shows a heatmap indicating values of the density function for a non-axis-aligned multivariate Gaussian with mean $\mu = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$ and covariance matrix $\Sigma = \begin{bmatrix} 10 & 5 \\ 5 & 5 \end{bmatrix}$. Here, the ellipses are again centered at $(3, 2)$, but now the major and minor axes have been rotated via a linear transformation.

Gaussian Mixture Models

- Model parameters
  - Number of components: $K$
  - “Mixing” weight vector: $\pi$
  - 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

Learning for Simple Univariate Case

- Problem setup:
  - Given number of components: $K$
  - Given points: $x_{1:N}$
  - Learn parameters: $\pi, \mu_{1:K}, \sigma_{1:K}^2$
- 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:
  $$p(x_{1:N}, z_{1:N,1:K} | \mu_{1:K}, \sigma_{1:K}^2, \pi)$$

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

EM for Univariate GMMs

- Initialize: $\pi, \mu_{1:K}, \sigma_{1:K}^2$
- 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

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

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

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

### K-Means vs. GMMs

<table>
<thead>
<tr>
<th></th>
<th>K-Means</th>
<th>GMM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Map</strong></td>
<td>Compute distance of points to centroids</td>
<td>E-step: compute expectation of z indicator variables</td>
</tr>
<tr>
<td><strong>Reduce</strong></td>
<td>Recompute new centroids</td>
<td>M-step: update values of model parameters</td>
</tr>
</tbody>
</table>

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”