Data-Intensive Computing with MapReduce

Session 7: Clustering and Classification

Jimmy Lin
University of Maryland
Thursday, March 7, 2013
Today’s Agenda

- Personalized PageRank
- Clustering
- Classification
Problem Setup

- Arrange items into clusters
  - High similarity between objects in the same cluster
  - Low similarity between objects in different clusters
- Cluster labeling is a separate problem
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 LSH 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 $c_j$, $s_j = \mu(c_j)$
K-Means Clustering Example

Pick seeds
Compute centroids
Reassign clusters
Reassign clusters
Compute centroids
Reassign clusters
Converged!
Basic MapReduce Implementation

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)
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 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:**

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

  - A vector-value random variable with such a distribution we write as:
    \[ x \sim \mathcal{N}(\mu, \Sigma) \]
Univariate Gaussian

\[ \phi_{\mu,\sigma^2}(x) \]

Multivariate Gaussians

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.

Source: Lecture notes by Chuong B. Do (IIT Delhi)
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: $\pi, \mu_{1:K}, \sigma^2_{1:K}$

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

  - M-step: compute new model parameters
    
    $$\pi_k = \frac{1}{N} \sum_n z_{n,k}$$
    
    $$\mu_k = \frac{1}{\sum_n z_{n,k}} \sum_n z_{n,k} \cdot x_n$$
    
    $$\sigma^2_k = \frac{1}{\sum_n z_{n,k}} \sum_n z_{n,k} ||x_n - \mu_k||^2$$
MapReduce Implementation

Map

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

Reduce

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

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

$$\sigma^2_k = \frac{1}{\sum_n z_{n,k} \sum_n z_{n,k} \|x_n - \mu_k\|^2}$$
# K-Means vs. GMMs

<table>
<thead>
<tr>
<th>Map</th>
<th>K-Means</th>
<th>GMM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Compute distance of</td>
<td>E-step: compute expectation of z indicator variables</td>
</tr>
<tr>
<td></td>
<td>points to centroids</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reduce</th>
<th>Recompute new centroids</th>
<th>M-step: update values of model parameters</th>
</tr>
</thead>
</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”
Supervised Machine Learning

- The generic problem of function induction given sample instances of input and output
  - Classification: output draws from finite discrete labels
  - Regression: output is a continuous value

- Focus here on supervised classification
  - Suffices to illustrate large-scale machine learning

This is not meant to be an exhaustive treatment of machine learning!
Applications

- Spam detection
- Content (e.g., movie) classification
- POS tagging
- Friendship recommendation
- Document ranking
- Many, many more!
Supervised Binary Classification

- Restrict output label to be *binary*
  - Yes/No
  - 1/0

- Binary classifiers form a primitive building block for multi-class problems
  - One vs. rest classifier ensembles
  - Classifier cascades
Limits of Supervised Classification?

- Why is this a big data problem?
  - Isn’t gathering labels a serious bottleneck?

- Solution: user behavior logs
  - Learning to rank
  - Computational advertising
  - Link recommendation

- The virtuous cycle of data-driven products
The Task

○ Given \( D = \{(x_i, y_i)\}_{i=1}^n \)

\[
x_i = [x_1, x_2, x_3, \ldots, x_d] \\
y \in \{0, 1\}
\]

○ Induce \( f : X \rightarrow Y \)

  • Such that loss is minimized

\[
\frac{1}{n} \sum_{i=0}^{n} \ell(f(x_i), y_i)
\]

○ Typically, consider functions of a parametric form:

\[
\arg\min_{\theta} \frac{1}{n} \sum_{i=0}^{n} \ell(f(x_i; \theta), y_i)
\]
Key insight: machine learning as an optimization problem!
(closed form solutions generally not possible)
Gradient Descent: Preliminaries

- Rewrite:
  \[
  \arg \min_{\theta} \frac{1}{n} \sum_{i=0}^{n} \ell(f(x_i; \theta), y_i) \quad \rightarrow \quad \arg \min_{\theta} L(\theta)
  \]

- Compute gradient:
  - “Points” to fastest increasing “direction”
  \[
  \nabla L(\theta) = \left[ \frac{\partial L(\theta)}{\partial w_0}, \frac{\partial L(\theta)}{\partial w_1}, \ldots, \frac{\partial L(\theta)}{\partial w_d} \right]
  \]

- So, at any point: *
  \[
  b = a - \gamma \nabla L(a)
  \]
  \[
  L(a) \geq L(b)
  \]

* caveats
Gradient Descent: Iterative Update

- Start at an arbitrary point, iteratively update:
  \[ \theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \nabla L(\theta^{(t)}) \]

- We have:
  \[ L(\theta^{(0)}) \geq L(\theta^{(1)}) \geq L(\theta^{(2)}) \ldots \]

- Lots of details:
  - Figuring out the step size
  - Getting stuck in local minima
  - Convergence rate
  - …
Gradient Descent

Repeat until convergence:

\[ \theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(x_i; \theta^{(t)}), y_i) \]
**Intuition behind the math...**

\[
\ell(x) = \frac{d}{dx} \ell \quad \nabla \ell
\]

\[
\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(x_i; \theta^{(t)}), y_i)
\]

New weights  Old weights  Update based on gradient
Gradient Descent

\[ \theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(x_i; \theta^{(t)}), y_i) \]
Lots More Details...

- Gradient descent is a “first order” optimization technique
  - Often, slow convergence
  - Conjugate techniques accelerate convergence

- Newton and quasi-Newton methods:
  - Intuition: Taylor expansion
    \[ f(x + \Delta x) = f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2 \]
  - Requires the Hessian (square matrix of second order partial derivatives): impractical to fully compute
Logistic Regression
Logistic Regression: Preliminaries

- **Given**
  \[ D = \{(x_i, y_i)\}_{i=1}^n \]
  \[ x_i = [x_1, x_2, x_3, \ldots, x_d] \]
  \[ y \in \{0, 1\} \]

- **Let’s define:**
  \[ f(x; w): \mathbb{R}^d \rightarrow \{0, 1\} \]
  \[ f(x; w) = \begin{cases} 1 & \text{if } w \cdot x \geq t \\ 0 & \text{if } w \cdot x < t \end{cases} \]

- **Interpretation:**
  \[ \ln \left[ \frac{\Pr (y = 1|x)}{\Pr (y = 0|x)} \right] = w \cdot x \]
  \[ \ln \left[ \frac{\Pr (y = 1|x)}{1 - \Pr (y = 1|x)} \right] = w \cdot x \]
Relation to the Logistic Function

- After some algebra:

\[
\Pr(y = 1 | x) = \frac{e^{w \cdot x}}{1 + e^{w \cdot x}}
\]

\[
\Pr(y = 0 | x) = \frac{1}{1 + e^{w \cdot x}}
\]

- The logistic function:

\[
f(z) = \frac{e^z}{e^z + 1}
\]
Training an LR Classifier

- Maximize the conditional likelihood:
  \[ \arg \max_w \prod_{i=1}^n \Pr(y_i|x_i, w) \]

- Define the objective in terms of conditional log likelihood:
  \[ L(w) = \sum_{i=1}^n \ln \Pr(y_i|x_i, w) \]

  - We know \( y \in \{0, 1\} \) so:
    \[ \Pr(y|x, w) = \Pr(y = 1|x, w)^y [1 - \Pr(y = 0|x, w)]^{(1-y)} \]

  - Substituting:
    \[ L(w) = \sum_{i=1}^n \left( y_i \ln \Pr(y_i = 1|x_i, w) + (1 - y_i) \ln \Pr(y_i = 0|x_i, w) \right) \]
LR Classifier Update Rule

- Take the derivative:
  \[
  L(w) = \sum_{i=1}^{n} \left( y_i \ln \Pr(y_i = 1|x_i, w) + (1 - y_i) \ln \Pr(y_i = 0|x_i, w) \right)
  \]
  \[
  \frac{\partial}{\partial w} L(w) = \sum_{i=0}^{n} x_i \left( y_i - \Pr(y_i = 1|x_i, w) \right)
  \]

- General form for update rule:
  \[
  w^{(t+1)} \leftarrow w^{(t)} + \gamma^{(t)} \nabla_w L(w^{(t)})
  \]
  \[
  \nabla L(w) = \left[ \frac{\partial L(w)}{\partial w_0}, \frac{\partial L(w)}{\partial w_1}, \ldots, \frac{\partial L(w)}{\partial w_d} \right]
  \]

- Final update rule:
  \[
  w_i^{(t+1)} \leftarrow w_i^{(t)} + \gamma^{(t)} \sum_{j=0}^{n} x_{j,i} \left( y_j - \Pr(y_j = 1|x_j, w^{(t)}) \right)
  \]
Lots more details...

- Regularization
- Different loss functions
- ...

Want more details?
Take a real machine-learning course!
MapReduce Implementation

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(x_i; \theta^{(t)}), y_i)$$

iterate until convergence

compute partial gradient

mappers

single reducer

update model
Shortcomings

- Hadoop is bad at iterative algorithms
  - High job startup costs
  - Awkward to retain state across iterations
- High sensitivity to skew
  - Iteration speed bounded by slowest task
- Potentially poor cluster utilization
  - Must shuffle all data to a single reducer
- Some possible tradeoffs
  - Number of iterations vs. complexity of computation per iteration
  - E.g., L-BFGS: faster convergence, but more to compute
Stochastic Gradient Descent
Batch vs. Online

Gradient Descent

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(x_i; \theta^{(t)}), y_i)$$

“batch” learning: update model after considering all training instances

Stochastic Gradient Descent (SGD)

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \nabla \ell(f(x; \theta^{(t)}), y)$$

“online” learning: update model after considering each (randomly-selected) training instance

In practice... just as good!
Practical Notes

- Most common implementation:
  - Randomly shuffle training instances
  - Stream instances through learner

- Single vs. multi-pass approaches

- “Mini-batching” as a middle ground between batch and stochastic gradient descent

We’ve solved the iteration problem!

What about the single reducer problem?
Ensembles
Ensemble Learning

- Learn multiple models, combine results from different models to make prediction

- Why does it work?
  - If errors uncorrelated, multiple classifiers being wrong is less likely
  - Reduces the variance component of error

- A variety of different techniques:
  - Majority voting
  - Simple weighted voting:
    
    \[
    y = \arg \max_{y \in Y} \sum_{k=1}^{n} \alpha_k p_k(y|x)
    \]
    
    - Model averaging
    - …
Practical Notes

- Common implementation:
  - Train classifiers on different input partitions of the data
  - Embarassingly parallel!

- Contrast with bagging

- Contrast with boosting
MapReduce Implementation

$$
\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \nabla \ell(f(x; \theta^{(t)}), y)
$$

- training data
- mapper
- model
- training data
- mapper
- model
- training data
- mapper
- model
- training data
- mapper
- model
MapReduce Implementation: Details

- Shuffling/resort training instances before learning
- Two possible implementations:
  - Mappers write model out as “side data”
  - Mappers emit model as intermediate output
Sentiment Analysis Case Study
Lin and Kolcz, SIGMOD 2012

- Binary polarity classification: \{positive, negative\} sentiment
  - Independently interesting task
  - Illustrates end-to-end flow
  - Use the “emoticon trick” to gather data

- Data
  - Test: 500k positive/500k negative tweets from 9/1/2011
  - Training: \{1m, 10m, 100m\} instances from before (50/50 split)

- Features: Sliding window byte-4grams

- Models:
  - Logistic regression with SGD (L2 regularization)
  - Ensembles of various sizes (simple weighted voting)
Ensembles with 10m examples better than 100m single classifier!

“for free”

Diminishing returns…
Takeaway Lesson

- Big data “recipe” for problem solving
  - Simple technique
  - Simple features
  - Lots of data

- Usually works very well!
Today’s Agenda

- Personalized PageRank
- Clustering
- Classification
Questions?