Big Data Infrastructure

Session 6: MapReduce – Data Mining

Jimmy Lin
University of Maryland
Monday, March 9, 2015

Today's Agenda

- Clustering
- Classification

Clustering

Source: Wikipedia (Star cluster)

Problem Setup

- Arrange items into clusters
  - High similarity (low distance) between objects in the same cluster
  - Low similarity (high distance) between objects in different clusters
- Cluster labeling is a separate problem

Applications

- Exploratory analysis of large collections of objects
- Collection pre-processing for web search
- Image segmentation
- Recommender systems
- Cluster hypothesis in information retrieval
- Computational biology and bioinformatics
- Many more!

Distance Metrics

1. Non-negativity:

$$d(x, y) \geq 0$$

2. Identity:

$$d(x, y) = 0 \iff x = y$$

3. Symmetry:

$$d(x, y) = d(y, x)$$

4. Triangle Inequality

$$d(x, y) \leq d(x, z) + d(z, y)$$

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: 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:

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

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

Distance: Hamming

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

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

Minhash

Source: www.flickr.com/photos/rheinitz/6158837748/

Near-Duplicate Detection of Webpages

- What's the source of the problem?
  - Mirror pages (legit)
  - Spam farms (non-legit)
  - Additional complications (e.g., nav bars)
- Naïve algorithm:
  - Compute cryptographic hash for webpage (e.g., MD5)
  - Insert hash values into a big hash table
  - Compute hash for new webpage: collision implies duplicate
- What's the issue?
- Intuition:
  - Hash function needs to be tolerant of minor differences
  - High similarity implies higher probability of hash collision

Minhash

- Seminal algorithm for near-duplicate detection of webpages
  - Used by AltaVista
  - For details see Broder et al. (1997)
- Setup:
  - Documents (HTML pages) represented by shingles (*n*-grams)
  - Jaccard similarity: dups are pairs with high similarity

Preliminaries: Representation

- Sets:
  - A = $\{e_1, e_3, e_7\}$
  - B = $\{e_3, e_5, e_7\}$
- Can be equivalently expressed as matrices:

Element	A	B
e_1	1	0
e_2	0	0
e_3	1	1
e_4	0	0
e_5	0	1
e_6	0	0
e_7	1	1

Preliminaries: Jaccard

Element	A	B
e_1	1	0
e_2	0	0
e_3	1	1
e_4	0	0
e_5	0	1
e_6	0	0
e_7	1	1

Let:

M_{00} = # rows where both elements are 0

M_{11} = # rows where both elements are 1

M_{01} = # rows where A=0, B=1

M_{10} = # rows where A=1, B=0

$$\mathrm{J}(A,B) = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}$$

Minhash

- Computing minhash
  - Start with the matrix representation of the set
  - Randomly permute the rows of the matrix
  - minhash is the first row with a "one"
- Example:

$h(A) = e_3$ $h(B) = e_5$

Element	A	B
e_1	1	0
e_2	0	0
e_3	1	1
e_4	0	0
e_5	0	1
e_6	0	0
e_7	1	1

Element	A	B
e_6	0	0
e_2	0	0
e_5	0	1
e_3	1	1
e_7	1	1
e_4	0	0
e_1	1	0

Minhash and Jaccard

Element	A	B	
e_6	0	0	M_{00}
e_2	0	0	M_{00}
e_5	0	1	M_{01}
e_3	1	1	M_{11}
e_7	1	1	M_{11}
e_4	0	0	M_{00}
e_1	1	0	M_{10}

$$P[h(A) = h(B)] = \mathrm{J}(A, B)$$

$$\frac{M_{11}}{M_{01} + M_{10} + M_{11}} \qquad \frac{M_{11}}{M_{01} + M_{10} + M_{11}}$$

To Permute or Not to Permute?

- Permutations are expensive
- Interpret the hash value as the permutation
- Only need to keep track of the minimum hash value
  - Can keep track of multiple minhash values at once

Extracting Similar Pairs (LSH)

- We know: $P[h(A) = h(B)] = \mathrm{J}(A, B)$
- Task: discover all pairs with similarity greater than *s*
- Algorithm:
  - For each object, compute its minhash value
  - Group objects by their hash values
  - Output all pairs within each group
- Analysis:
  - Probability we will discovered all pairs is *s*
  - Probability that any pair is invalid is (1 – s)
- What's the fundamental issue?

Two Minhash Signatures

- Task: discover all pairs with similarity greater than s
- Algorithm:
  - For each object, compute two minhash values and concatenate together into a signature
  - Group objects by their signatures
  - Output all pairs within each group
- Analysis:
  - Probability we will discovered all pairs is s^2
  - Probability that any pair is invalid is $(1 - s)^2$

k Minhash Signatures

- Task: discover all pairs with similarity greater than s
- Algorithm:
  - For each object, compute k minhash values and concatenate together into a signature
  - Group objects by their signatures
  - Output all pairs within each group
- Analysis:
  - Probability we will discovered all pairs is s^k
  - Probability that any pair is invalid is $(1 - s)^k$
- What's the issue now?

n different *k* Minhash Signatures

- Task: discover all pairs with similarity greater than s
- Algorithm:
  - For each object, compute n sets k minhash values
  - For each set, concatenate k minhash values together
  - Within each set:
    - Group objects by their signatures
    - Output all pairs within each group
  - De-dup pairs
- Analysis:
  - Probability we will miss a pair is $(1 - s^k)^n$
  - Probability that any pair is invalid is $n(1 - s)^k$

Practical Notes

- In some cases, checking all candidate pairs may be possible
  - Time cost is small relative to everything else
  - Easy method to discard false positives
- Most common practical implementation:
  - Generate *M* minhash values, randomly select *k* of them *n* times
  - Reduces amount of hash computations needed
- Determining "authoritative" version is non-trivial

MapReduce Implementation

- Map over objects:
  - Generate *M* minhash values, randomly select *k* of them *n* times
  - Each draw yields a signature: emit as intermediate key, value is object id
- Shuffle/sort:
- Reduce:
  - Receive all object ids with same signature, emit clusters
- Second pass to de-dup and group clusters

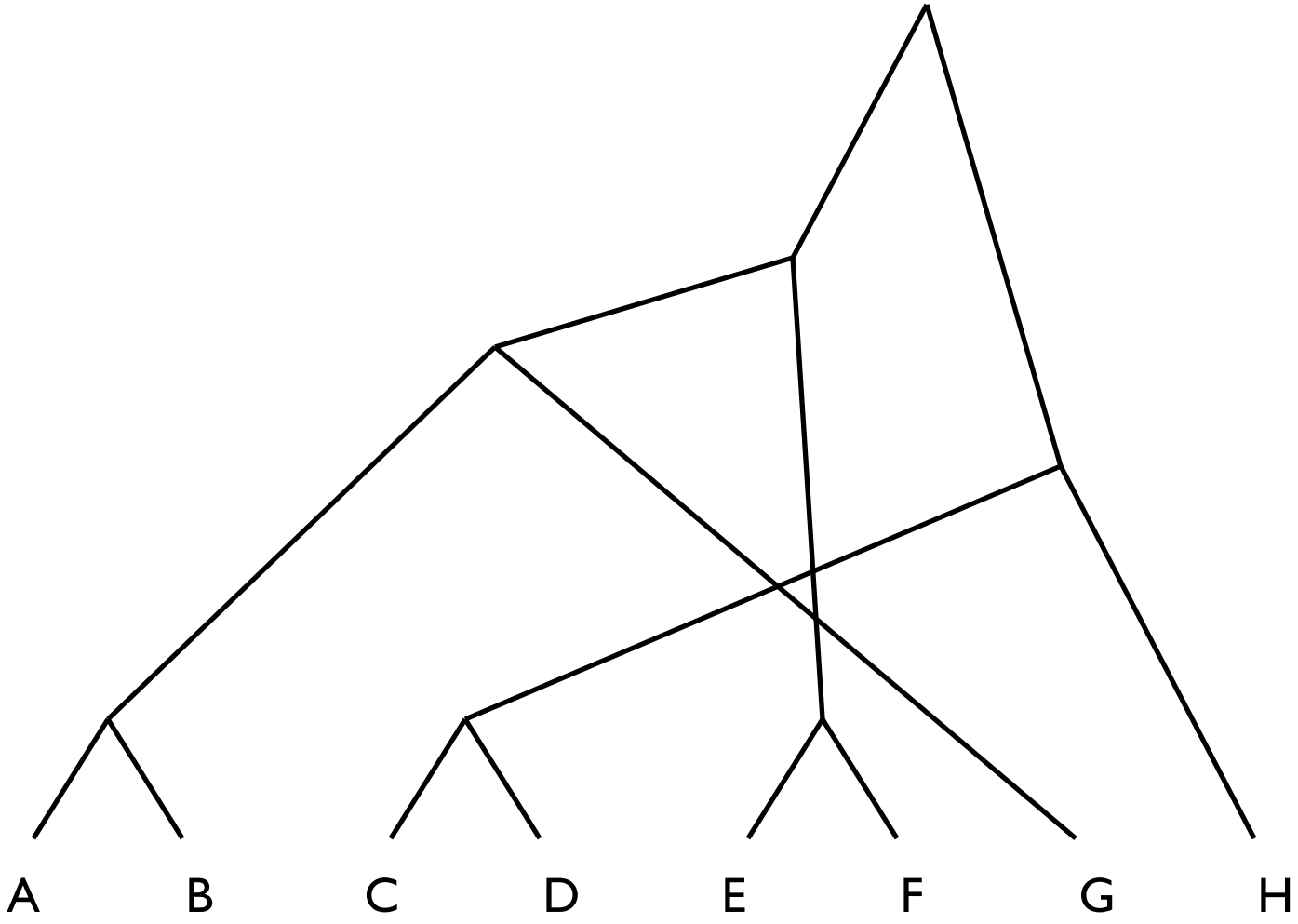
General Clustering Approaches

- 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?

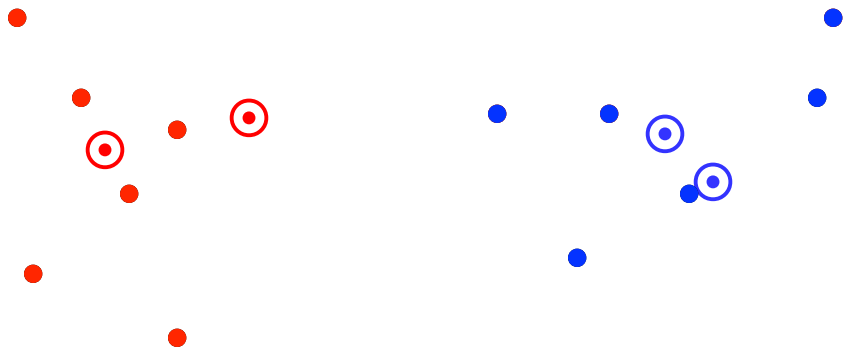
K-Means Algorithm

- 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

```
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)
```

(Just a clever way to keep track of denominator)

MapReduce Implementation w/ IMC

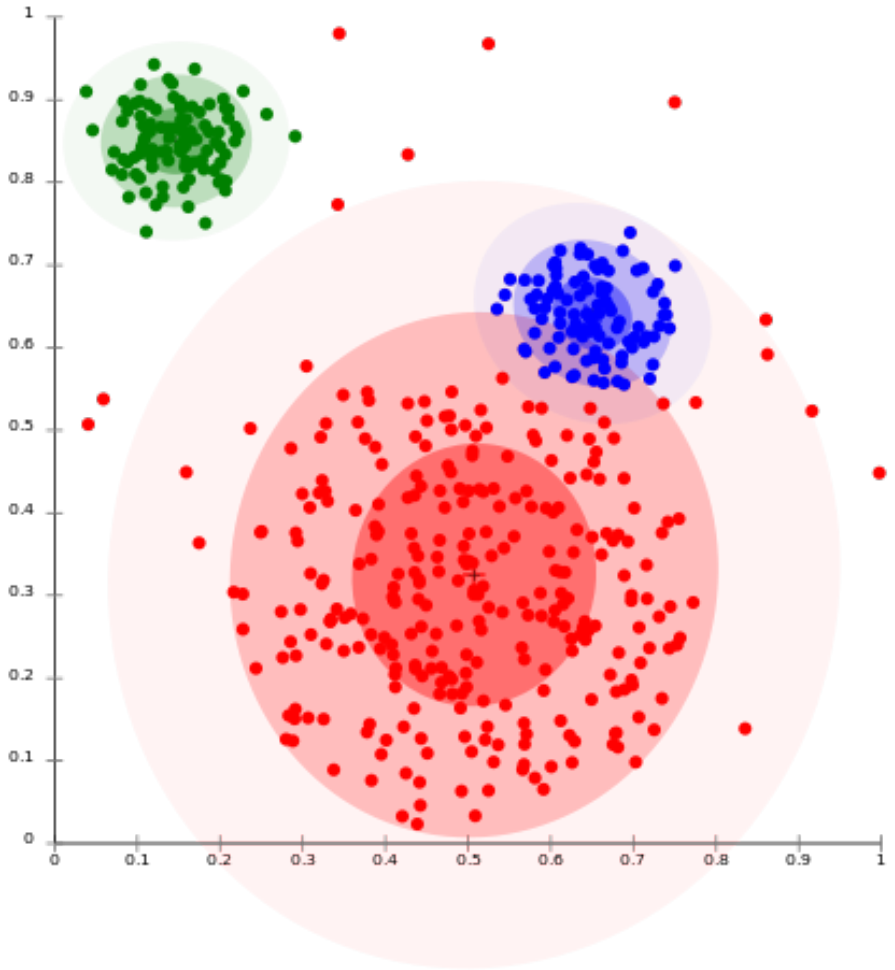
```
 1: class Mapper
 2:     method Configure()
 3:         c ← LoadClusters()
 4:         H ← InitAssociativeArray()
 5:     method Map(id i, point p)
 6:         n ← NearestClusterID(clusters c, point p)
 7:         p ← ExtendPoint(point p)
 8:         H{n} ← H{n} + p
 9:     method Close()
10:         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:

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

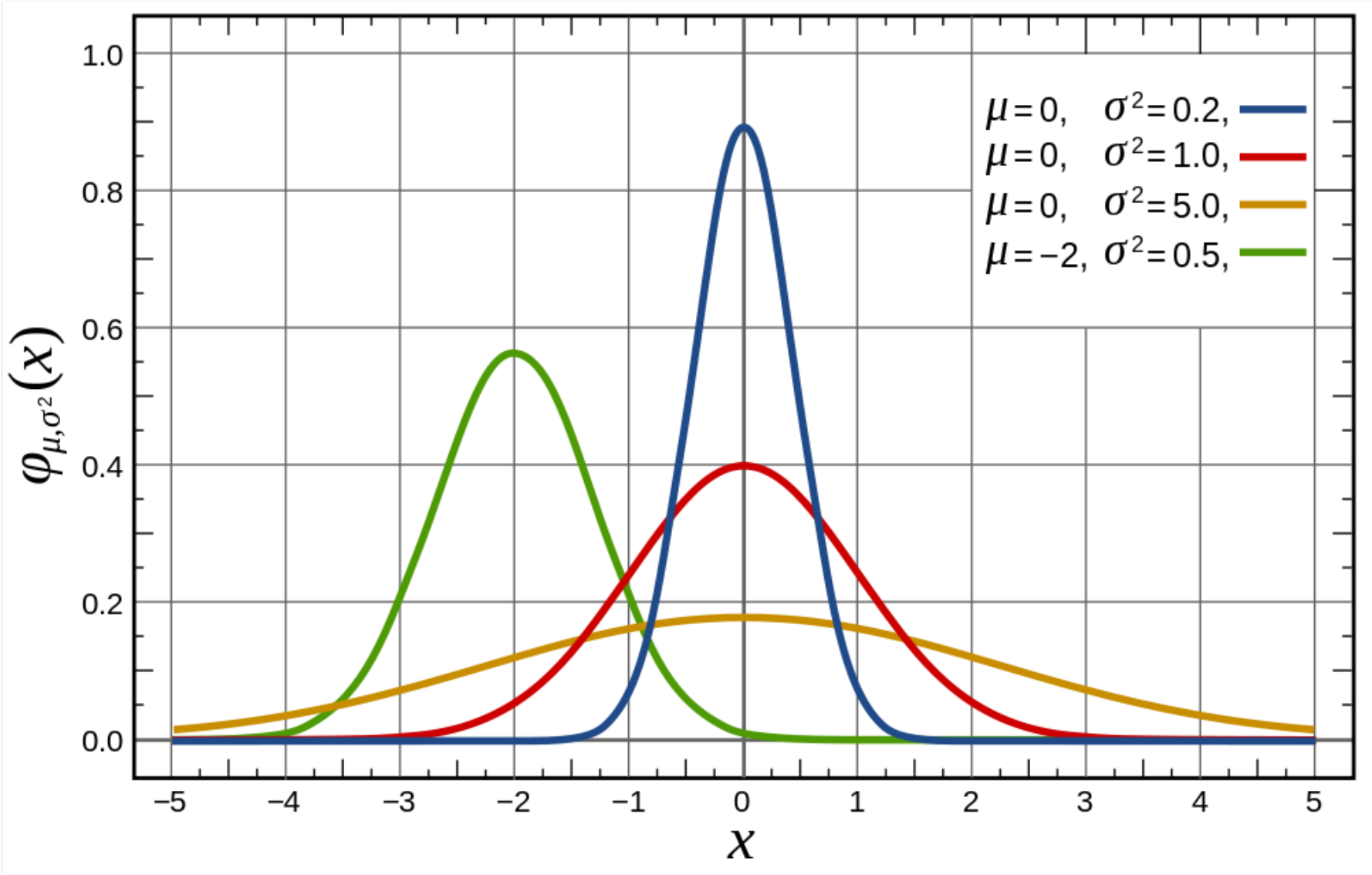
- Multivariate Gaussian:

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

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

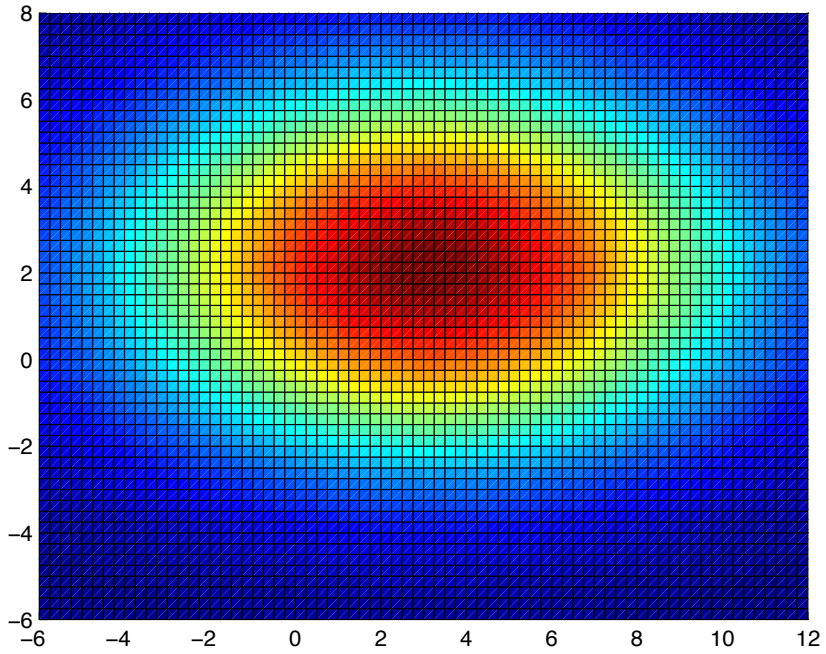
$$\mathrm{x} \sim \mathcal{N}(\mu, \Sigma)$$

Univariate Gaussian

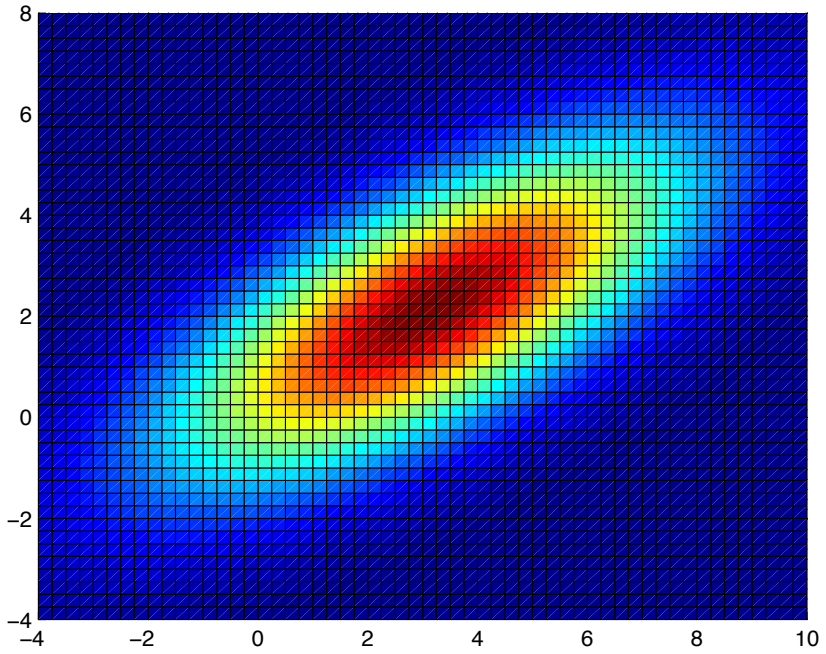


Source: Wikipedia (Normal Distribution)

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

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:

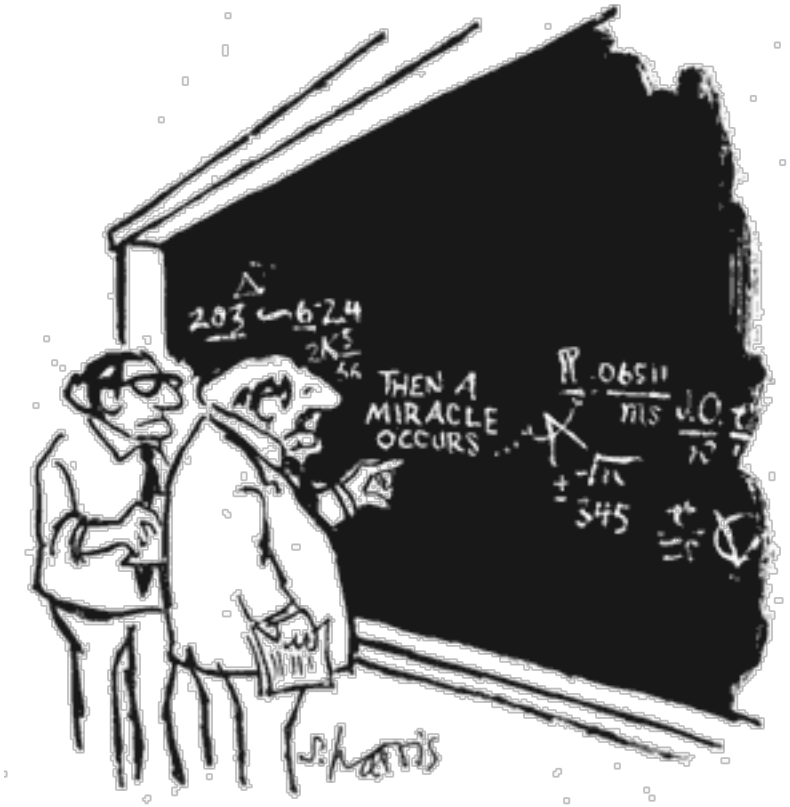
$$p(x_{1:N}, z_{1:N,1:K} | \mu_{1:K}, \sigma^2_{1:K}, \pi)$$

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

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

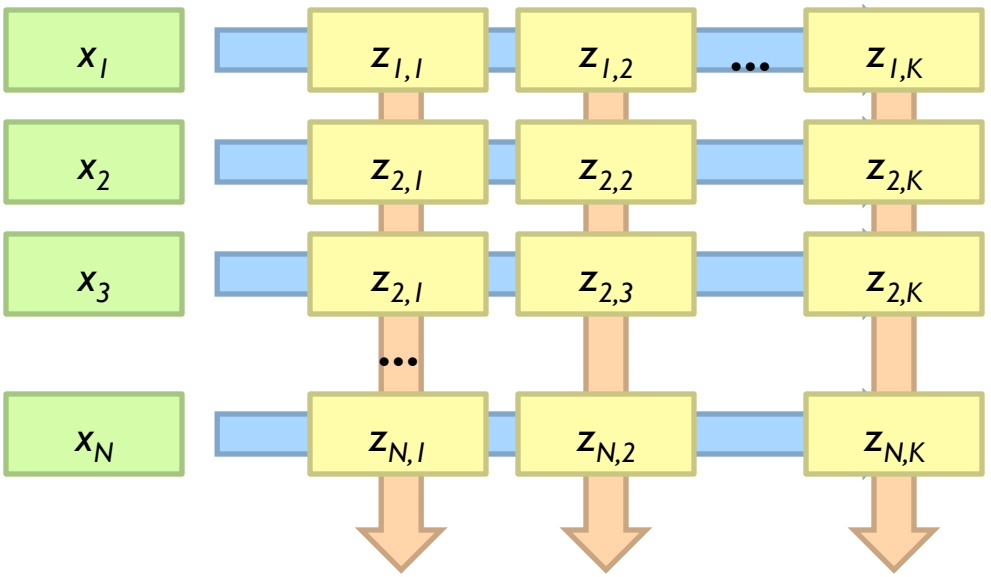
$$\mu_k = \frac{1}{\sum_n z_{n,k}} \sum_n z_{n,k} \cdot x_n$$

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

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

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

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

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

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

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"

Classification

Source: Wikipedia (Sorting)

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 $D = \{(\mathrm{x}_i, y_i)\}_i^n$

  label

  (sparse) feature vector

  $\mathrm{x}_i = [x_1, x_2, x_3, \ldots, x_d]$

  $y \in \{0, 1\}$

- Induce $f : X \rightarrow Y$
  - Such that loss is minimized

$$\frac{1}{n} \sum_{i=0}^{n} \ell(f(\mathrm{x}_i), y_i)$$

loss function

- Typically, consider functions of a parametric form:

$$\arg\min_{\theta} \frac{1}{n} \sum_{i=0}^{n} \ell(f(x_i; \theta), y_i)$$

model parameters

Key insight: machine learning as an optimization problem!

(closed form solutions generally not possible)

Gradient Descent: Preliminaries

- Rewrite:

$$\arg\min_{\theta} \frac{1}{n} \sum_{i=0}^{n} \ell(f(\mathrm{x}_i; \theta), y_i) \quad \Rightarrow \quad \arg\min_{\theta} L(\theta)$$

- 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: *

$$\mathrm{b} = \mathrm{a} - \gamma \nabla L(\mathrm{a})$$

$$L(\mathrm{a}) \geq L(\mathrm{b})$$

* 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:

$$L(\theta^{(0)}) \geq L(\theta^{(1)}) \geq L(\theta^{(2)}) \dots$$

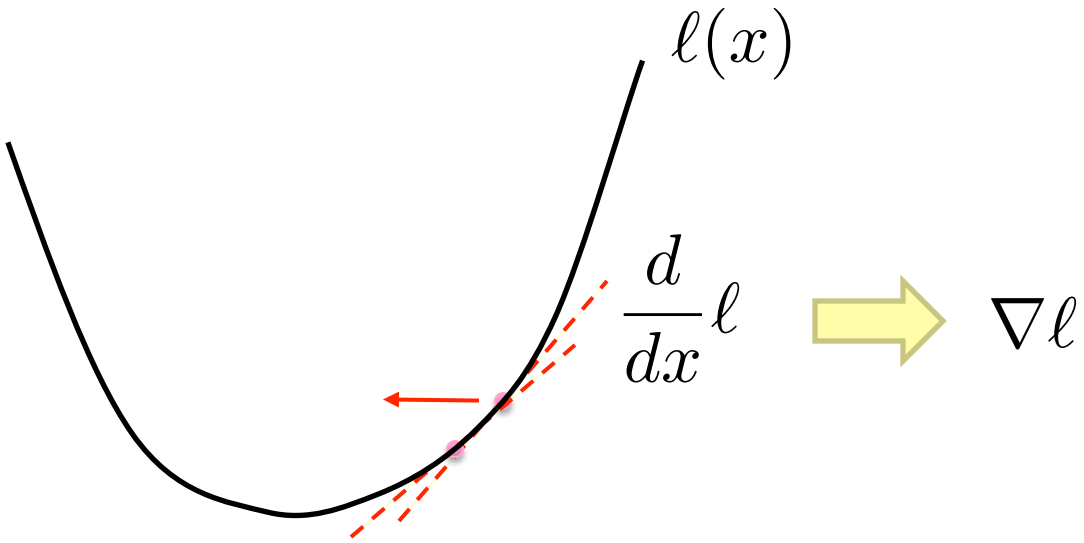
- Lots of details:
  - Figuring out the step size
  - Getting stuck in local minima
  - Convergence rate
  - …

Gradient Descent

Repeat until convergence:

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

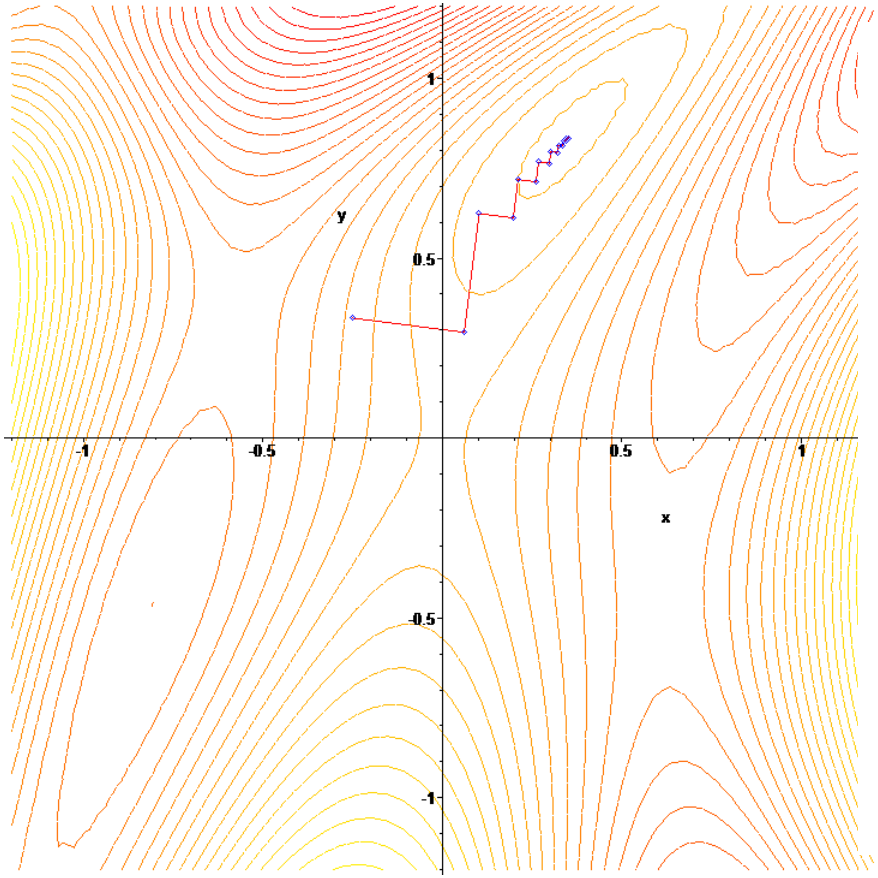
Intuition behind the math...

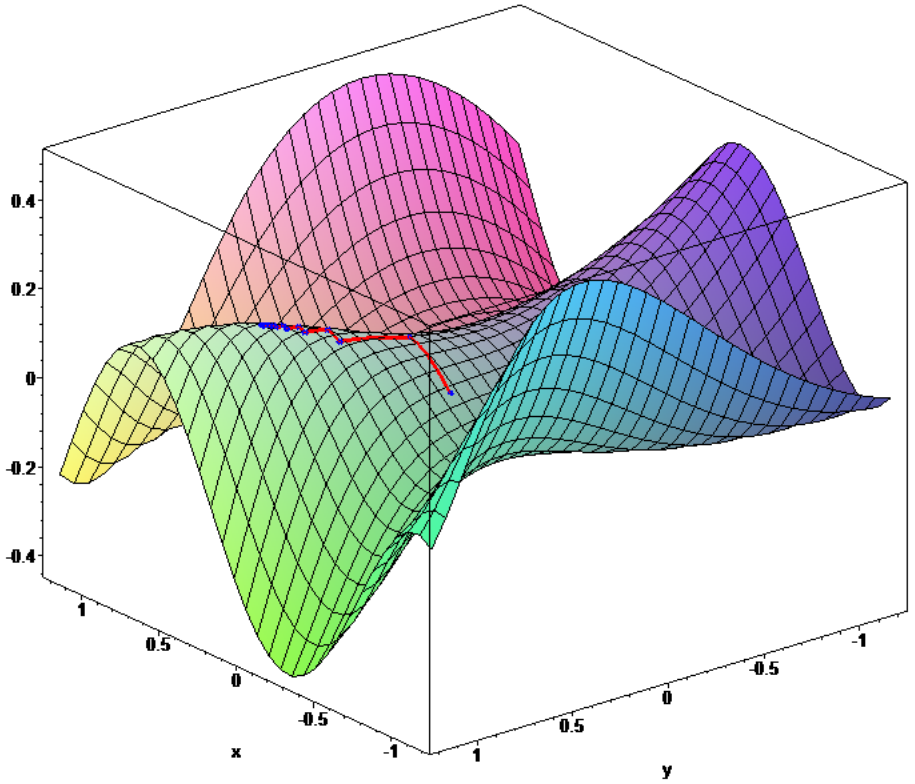


$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(\mathrm{x}_i; \theta^{(t)}), y_i)$$

New weights Old weights

Update based on gradient


y
x
1
0.5
-0.5
-1
-1
-0.5
0.5
1


0.4
0.2
0
-0.2
-0.4
x
1
0.5
0
-0.5
-1
y
1
0.5
0
-0.5
-1

Gradient Descent

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

Source: Wikipedia (Hills)

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

Source: Wikipedia (Hammer)

Logistic Regression: Preliminaries

- Given $D = \{(\mathrm{x}_i, y_i)\}_i^n$

$$\mathrm{x}_i = [x_1, x_2, x_3, \ldots, x_d]$$
$$y \in \{0, 1\}$$

- Let's define:

$$f(\mathrm{x}; \mathrm{w}) : \mathbb{R}^d \rightarrow \{0, 1\}$$
$$f(\mathrm{x}; \mathrm{w}) = \begin{cases} 1 \text{ if } \mathrm{w} \cdot \mathrm{x} \geq t \\ 0 \text{ if } \mathrm{w} \cdot \mathrm{x} < t \end{cases}$$

- Interpretation:

$$\ln \left[\frac{\Pr(y = 1|\mathrm{x})}{\Pr(y = 0|\mathrm{x})} \right] = \mathrm{w} \cdot \mathrm{x}$$

$$\ln \left[\frac{\Pr(y = 1|\mathrm{x})}{1 - \Pr(y = 1|\mathrm{x})} \right] = \mathrm{w} \cdot \mathrm{x}$$

Relation to the Logistic Function

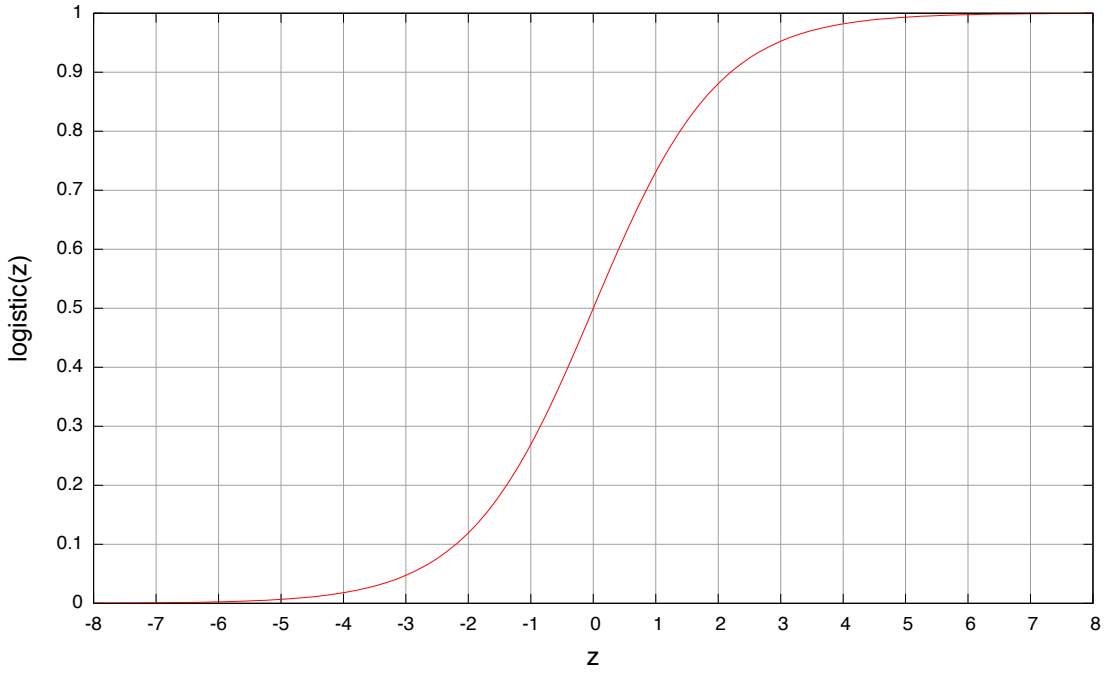
- After some algebra:

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

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

- The logistic function:

$$f(z) = \frac{e^z}{e^z + 1}$$



Training an LR Classifier

- Maximize the conditional likelihood:

$$\arg\max_{\mathrm{w}} \prod_{i=1}^{n} \Pr(y_i|\mathrm{x}_i, \mathrm{w})$$

- Define the objective in terms of conditional log likelihood:

$$L(\mathrm{w}) = \sum_{i=1}^{n} \ln \Pr(y_i|\mathrm{x}_i, \mathrm{w})$$

  - We know $y \in \{0, 1\}$ so:

$$\Pr(y|\mathrm{x}, \mathrm{w}) = \Pr(y = 1|\mathrm{x}, \mathrm{w})^{y} \left[1 - \Pr(y = 0|\mathrm{x}, \mathrm{w})\right]^{(1-y)}$$

  - Substituting:

$$L(\mathrm{w}) = \sum_{i=1}^{n} \Big(y_i \ln \Pr(y_i = 1|\mathrm{x}_i, \mathrm{w}) + (1 - y_i) \ln \Pr(y_i = 0|\mathrm{x}_i, \mathrm{w}) \Big)$$

LR Classifier Update Rule

- Take the derivative:

$$L(\mathrm{w}) = \sum_{i=1}^{n} \Big(y_i \ln \Pr(y_i = 1 | \mathrm{x}_i, \mathrm{w}) + (1 - y_i) \ln \Pr(y_i = 0 | \mathrm{x}_i, \mathrm{w}) \Big)$$

$$\frac{\partial}{\partial \mathrm{w}} L(\mathrm{w}) = \sum_{i=0}^{n} \mathrm{x}_i \Big(y_i - \Pr(y_i = 1 | \mathrm{x}_i, \mathrm{w}) \Big)$$

- General form for update rule:

$$\mathrm{w}^{(t+1)} \leftarrow \mathrm{w}^{(t)} + \gamma^{(t)} \nabla_{\mathrm{w}} L(\mathrm{w}^{(t)})$$

$$\nabla L(\mathrm{w}) = \left[\frac{\partial L(\mathrm{w})}{\partial w_0}, \frac{\partial L(\mathrm{w})}{\partial w_1}, \ldots \frac{\partial L(\mathrm{w})}{\partial w_d} \right]$$

- Final update rule:

$$\mathrm{w}_i^{(t+1)} \leftarrow \mathrm{w}_i^{(t)} + \gamma^{(t)} \sum_{j=0}^{n} x_{j,i} \Big(y_j - \Pr(y_j = 1 | \mathrm{x}_j, \mathrm{w}^{(t)}) \Big)$$

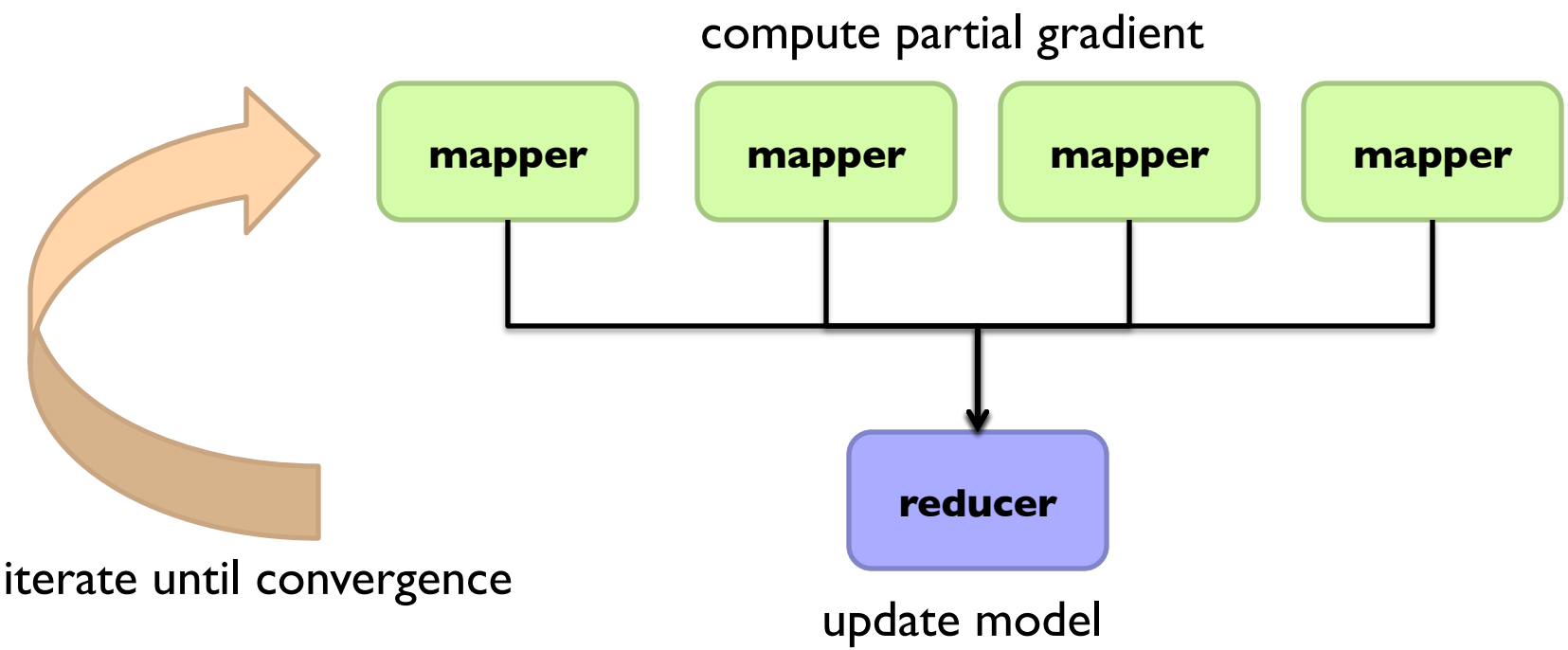
Lots more details…

- Regularization
- Different loss functions
- …

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

MapReduce Implementation

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \underbrace{\underbrace{\frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(\mathrm{x}_i; \theta^{(t)}), y_i)}_{\text{mappers}}}_{\text{single reducer}}$$



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

Source: Wikipedia (Hills)

Source: Wikipedia (Water Slide)

Batch vs. Online

Gradient Descent

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

"batch" learning: update model after considering all training instances

Stochastic Gradient Descent (SGD)

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

"online" learning: update model after considering *each* (randomly-selected) training instance

In practice… just as good!

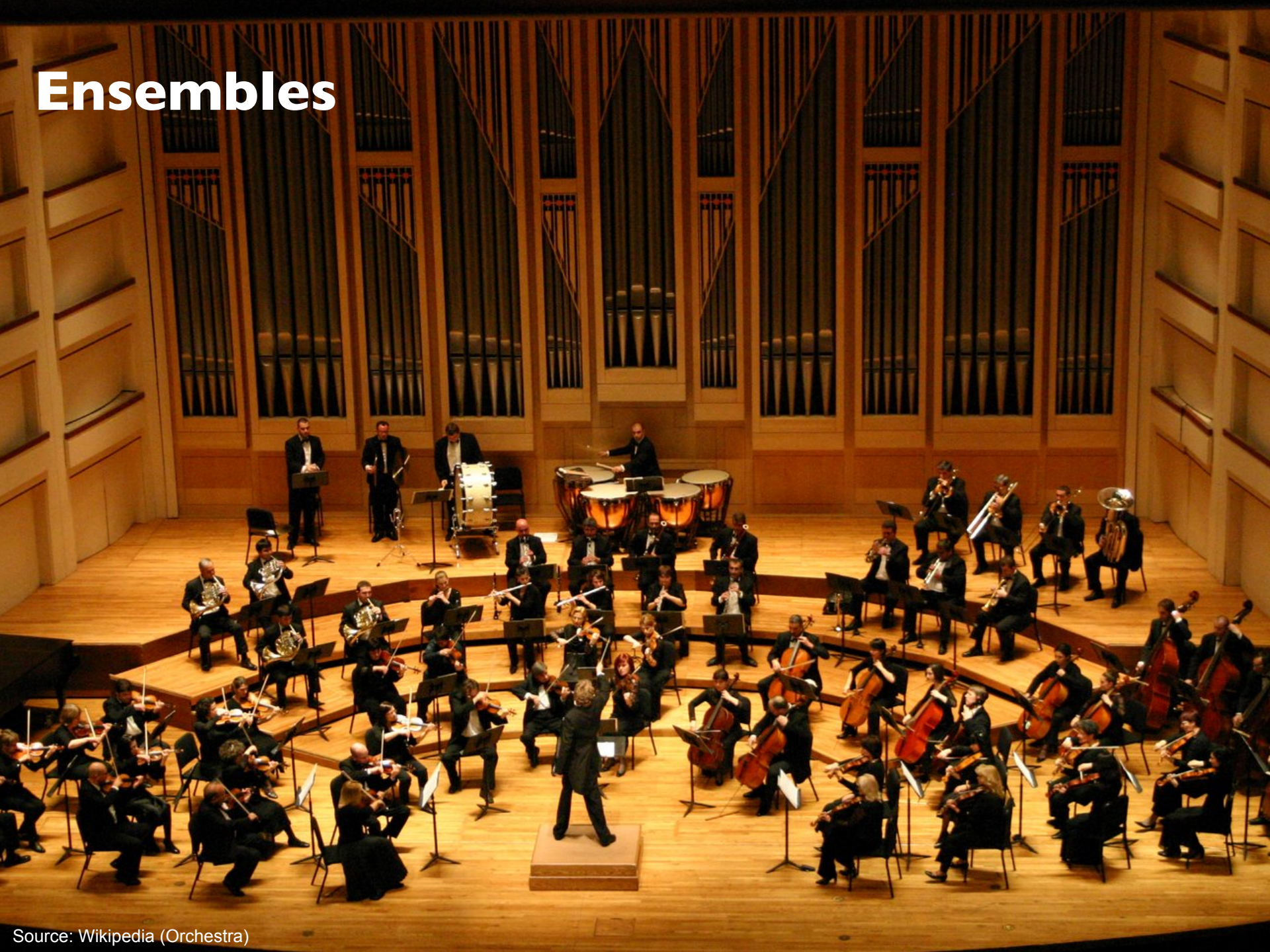
Practical Notes

- 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



Source: Wikipedia (Orchestra)

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:

$$y = \arg\max_{y \in Y} \sum_{k=1}^{n} \alpha_k p_k(y|\mathrm{x})$$

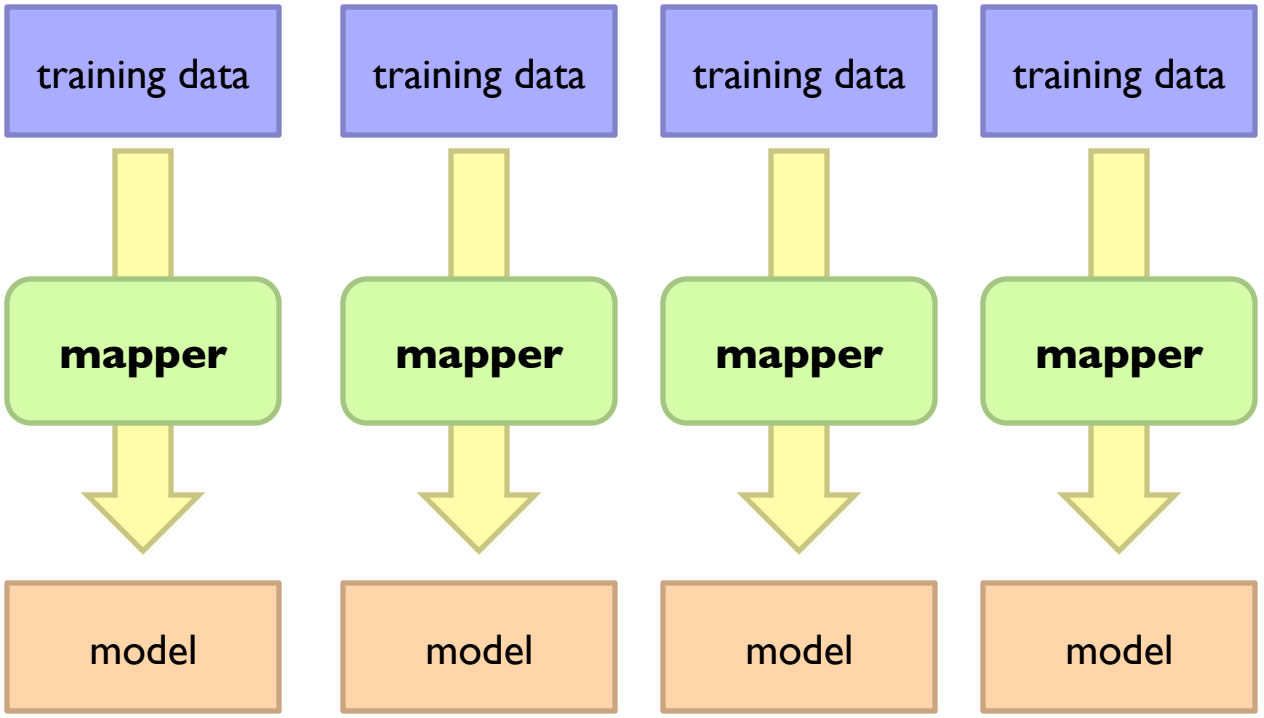
  - Model averaging
  - …

Practical Notes

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

MapReduce Implementation

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



MapReduce Implementation: Details

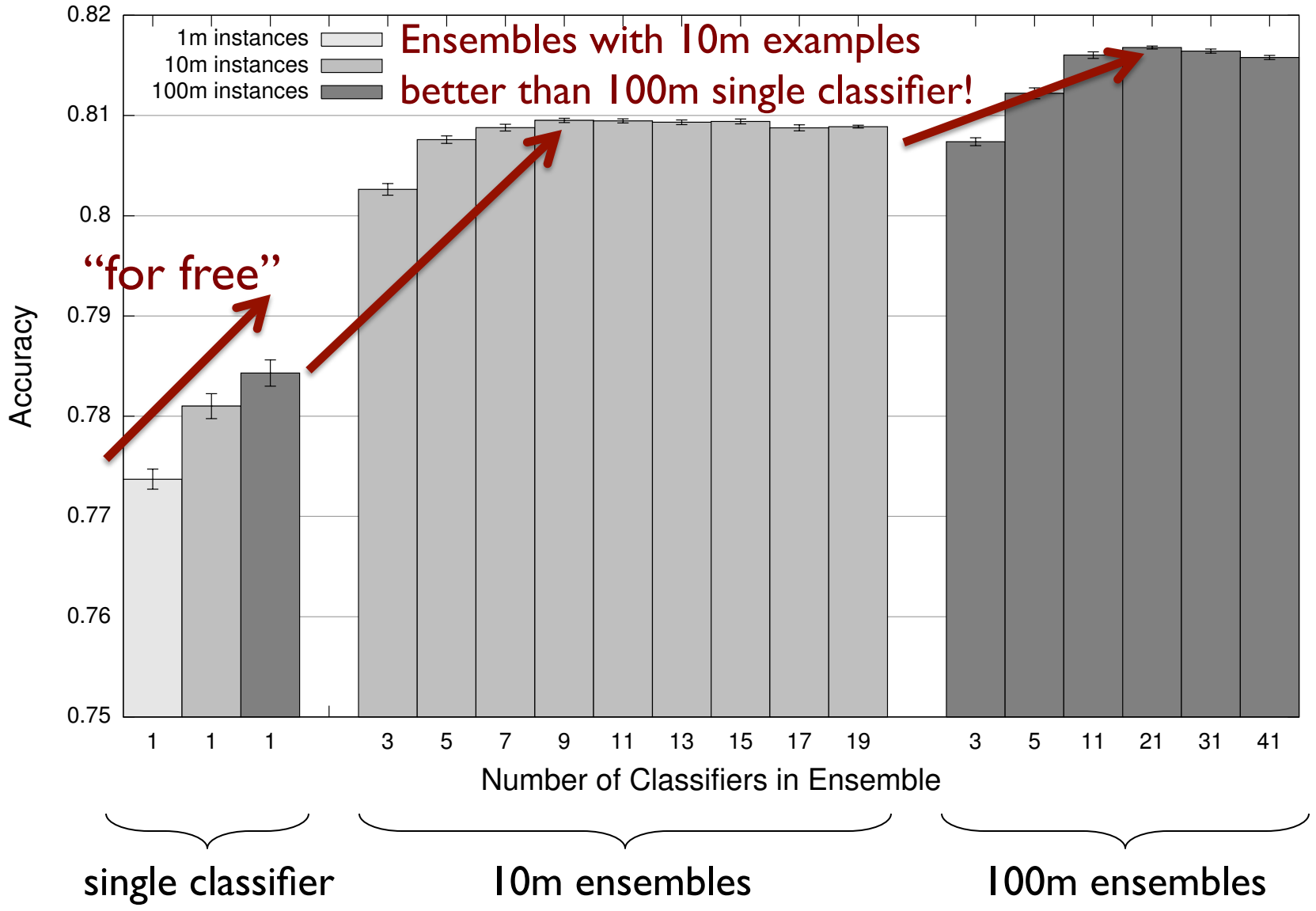
- Shuffling/resort training instances before learning
- Two possible implementations:
  - Mappers write model out as "side data"
  - Mappers emit model as intermediate output

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)

Diminishing returns…



Takeaway Lesson

- Big data "recipe" for problem solving
  - Simple technique
  - Simple features
  - Lots of data
- Usually works very well!

Today's Agenda

- Clustering
- Classification

Source: Wikipedia (Japanese rock garden)