GREAT 2011 Summer School

C3: How to analyze a petabyte

Matthew J. Graham (Caltech, VAO)

Overview

- Using MapReduce for analysis
- Parallelizing an algorithm
- Learning theory
- Making an algorithm online
- Making an algorithm stochastic

The world of MapReduce

- Two basic scenarios:
  - Traditional batch processing – text processing, data warehousing
  - Machine learning:
    - Training can be computationally hard
    - Can be necessary to send tons of data to each mapper
- The key is to have a summation form:

$$y = \sum f(x)$$

reduce map

How to sort a petabyte in 16.1 (or 6.8) hrs

- Algorithm (Hadoop terasort):
  - Divide data between mappers
  - Mappers produce a key for each data
  - The data is partitioned across R reducers using a partitioning function based on a two-level prefix tree (trie) that guarantees that all the keys in reduce N are after all of the keys in reduce N-1
  - Normal guarantee that within a given partition, key/value pairs are processed in increasing key order
  - Reducer is an identity function
- Launch on ~3800 nodes (2 x Quad Core 2.5 GHz Xeons, 4 x 160 GB SATA, 16 GB RAM per node), 80000 mappers, 20000 reducers (1.03 TB/minute)

MapReduce for data mining

	One iteration	Multiple iterations	Not good for MR
Clustering		k-means	
Classification	Naïve Bayes, kNN	Gaussian mixture	SVM, HMM
Graphs		PageRank, connected component	
Information retrieval	Inverted index		

- Single pass
- Keys uniformly distributed

- Small shared information synchronized across iterations
- Multiple passes
- Intermediate states are small

- Large shared information
- Lot of fine-grained synchronization

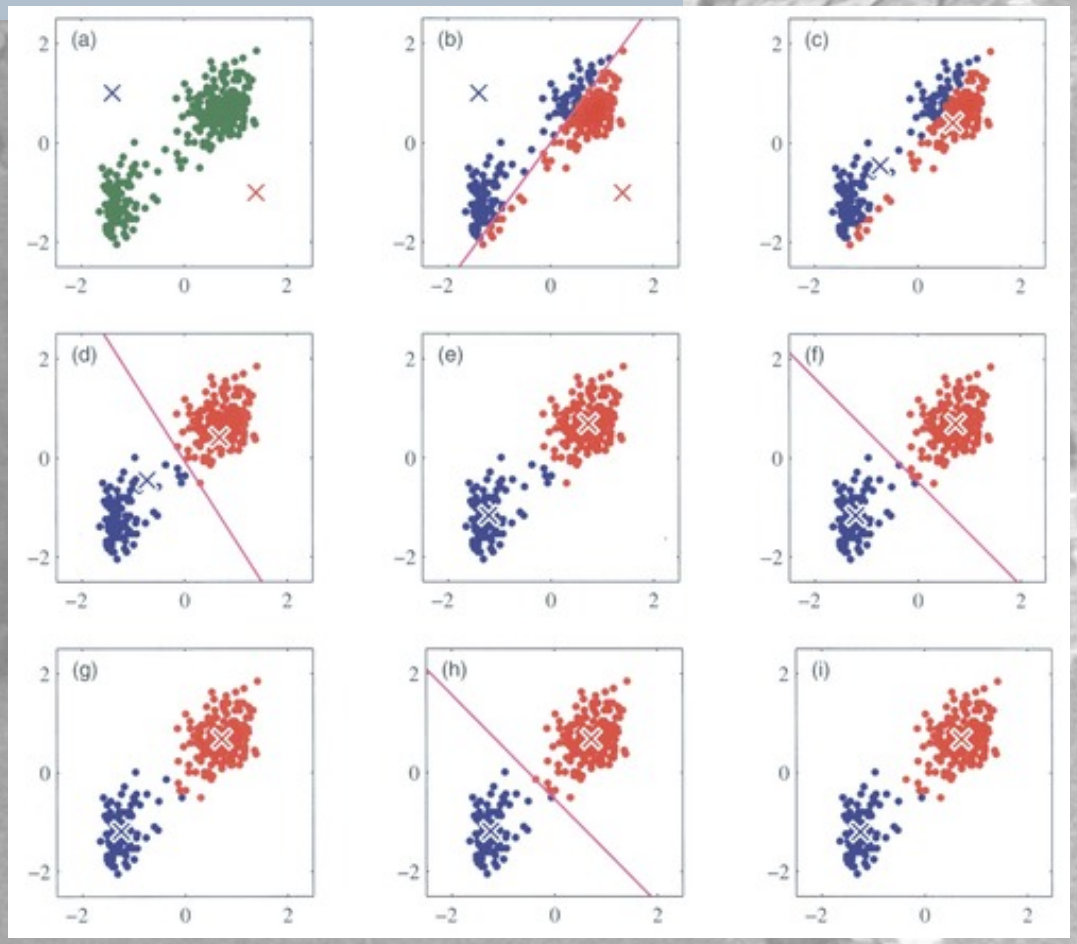
K-means clustering

1. Start with k cluster centers (chosen randomly or to some recipe)
2. Assign each data point to its nearest cluster center
3. Reevaluate the cluster centers as the "average" of the data in (2)
4. Repeat until cluster centers no longer change or some other stopping criterion is met

Aims to minimize squared loss function:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2$$

K-means illustration



Parallelizing k-means

- Analysis:
  - Large amounts of data that do not need to be sent around processors
  - Minimum processor intercommunication
  - Data set needs to be read for each iteration but each point only needs to be read by one processor
- Solution:
  - Divide data amongst processors
  - Each processor reads previous iteration's cluster centers and assigns its data to the clusters
  - Each processor then calculates new centers for its data
  - True cluster centers for this iteration are weighted average of new centers from each processor
- Local clustering, work with average quantities

MapReduce implementation

- Initialize centers for iteration 0 (prerun?)
- Map:
  - Get centers from last iteration
  - Read data and calculate distance to each center
  - Calculate average coordinates for each cluster
  - Emit (key = cluster id, value = size of cluster + average coordinates) for each cluster
- Reduce:
  - Calculate weighted average of its input
  - Emit (iteration #, cluster id, cluster center coordinates, size of cluster)
- Persist cluster details for each iteration

Visiting the data only once

- Data are too large to store on available resources or hold in memory
- Data are not persistent so no later processing possible
- Rough-and-ready results required for data exploration
- Time-dependent results to check convergence, data quality

Learning theory

- The goal of a learning system is to find the minimum of the expected loss function
- The ground truth function is unknown
- An approximation (empirical loss function) can be made using a finite training set of independent observations
- Types of training methods:
  - Batch-based – all training data at same time
  - Online – incremental updating
  - Decremental – handles concept drift
  - Stochastic – using random samples of data
- Established k-means MapReduce implementations are batch-based because output of mappers is completely written to file system before grouping by keys

Making k-means online

- Need an incremental version of the algorithm:

  Make initial guesses for the centers $w_1, w_2, \ldots, w_t$
  Set the counts $n_1, n_2, \ldots, n_t$ to zero
  Until interrupted:
  Acquire the next example, x
  If w_i is closest to x:
  Increment n_i
  Replace w_i by $w_i + (1/n_i)*(x - w_i)$
  end_if
  end_until

- Can this be parallelized? (Exercise for the student)

Making k-means stochastic

- k-means is prone to local minima and sensitive to initial clusters
- Normally repeat several times
- Stochastic algorithm can reach (global) minimum quicker:
  - (Nominally) works with subset of the data
  - Relative position of clusters found very quickly
  - Terminal convergence slowed down by stochastic noise implied by random choice of points
  - Great learning algorithm but hopeless optimization algorithm

Gradient descent learning

- In gradient descent, the parameter updates are proportional to the gradient of the partial loss:

$$w^{t+1} = w^t - \mu^t \nabla J^t(w^t)$$

where w^t is the value of the centers after updating from example t and μ is the learning rate.

- For k-means:

$$w^t = w^{t-1} - 2\mu_t(w^{t-1} - z_i)$$

Learning rate

- The convergence rate of gradient descent drastically improves:
  - replacing the scalar learning rate μ_t by a definite positive symmetric matrix Φ_t that approximates the inverse Hessian of the loss function:

$$\Phi_t \approx H^{-1}(w_t), H(w) = \nabla\nabla_w J(w)$$

- For k-means:
  - the Hessian of the loss function is a diagonal matrix whose coefficients are equal to the probability that an example x is associated with the center w_t
- This can be estimated by:
  - simply counting how many examples n_t have been associated with a cluster w_t

Stochastic implementation

Make initial guesses for the centers w_1, w_2, ..., w_t
Set the counts n_1, n_2, ..., n_t to zero
    Until interrupted:
        Acquire the next example, x
        If w_i is closest to x:
                Increment n_i
                Replace w_i by $w_i + (1/n_i)*(x - w_i)$
        end_if
    end_until

This is the same as the online version!