Big Data Learning

Ideas on dealing with a lot of data

Data Instance Size

- One million instances
  - Can be crowdsourced
  - Can be processed by a quadratic algorithm (once parallelized)
  - Preprocessing/data probing is crucial
- One billion instances
  - Web-scale
  - Data likely in different formats (ASCII, pictures, JSON, PDF)
  - Likely to have (near) duplicates
  - Likely has badly preprocessing
  - Storage is an issue
- One trillion instances
  - Beyond reach of modern technology
  - Data privacy/inconsistence an issue since can’t store all in one place

Crowdsourcing labeled data

- Crowdsourcing is a tough business
  - People are not machines
- Any worker who can game the system WILL game it
- Validation framework + qualification tests are a must
- Labeling a lot of data can be fairly expensive

How to label 1M Instances

- Budge a month of work + ~ $50,000
  - Offer someone you trust $10,000 for one month of work
  - Construct a qualification test for your job
  - Hire 100 workers who pass it (and keep their worker IDs)
  - Explain to them the task to make sure they get it
  - Offer them 4 cents per data instance that they do correctly
  - The person you trust will label 500 data instances each day and use this to validate worker results
  - Each worker gets a daily task of 1000 data instances
    - Some of these (500) are already labeled by the contractor
  - Check every worker's result on that validation set
    - Fire the worst 50 workers (and disregard their results)
  - Hire 50 new ones
  - Repeat for a month (20 working days)
    - 50 workers x 20 days x 1000 data points/day x 4 cents

Now What?

- Train your model!
- Rule of thumb: more training data produces better results
- Parallelization is likely unavoidable

MapReduce

- MapReduce is a programming model and an associated implementation (library) for processing and generating large data sets (on large clusters)
- A new abstraction allows to express the simple computations that hides the messy details of parallelization, fault-tolerance, data distribution, and load balancing in a library

Motivation

- Large-Scale Data Processing
  - Want to use 1000s of CPUs
  - But don’t want hassle of managing things
- MapReduce provides
  - Automatic parallelization & distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring & status updates

Map/Reduce in Lisp/Scheme

- (map f list [list2 list3 …])
  - Applies function f to each element in the lists
- Example:
  - (map square ‘(1 2 3 4))
  - ‘(1 4 9 16)
- (reduce f list)
  - Recursively applies the result of applying f to each member of list
- Example:
  - (reduce + ‘(1 4 9 16))
  - 30

Map/Reduce

- We want to think about
  - What parts of the algorithms are independent and thus can be parallelized (Map)
  - At what parts of the algorithm must we combine results (Reduce) before moving forward

Locally Weighted Linear Regression (LWLR)

- Solve $\theta = A^{-1}b$ where
  - $A = \sum_{i=1}^{m} w_i(x^{(i)}x^{(i)^T})$
  - $b = \sum_{i=1}^{m} w_i(x^{(i)}y^{(i)})$
  - When w_i=1, this is global linear regression
- Mappers:
  - One set parallelizes the computation of A
  - Another set computes b
- Two reducers:
  - Combine results from parallelizing A and b (sums results)
- Finally compute the solution

Naïve Bayes (NB)

- Goal: estimate $\mathrm{P}\left(y = 1 \middle| x^{(i)}\right)$ *and* $P\left(y = 0 \middle| x^{(i)}\right)$
- Computation necessary:
  - Count the occurrences of y=1 and y=0
  - For all features j
    - Count the occurrence of $(\mathrm{x_j} = x_j^{(i)} | y = 1)$, $(\mathrm{x_j} = x_j^{(i)} | y = 0)$ and $x_j^{(i)}$
  - Compute divisions to get $P\left(x_j^{(i)} \middle| Y = y\right)$
- Mappers:
  - Count a subgroup of training samples
- Reducers:
  - Aggregate the intermediate counts and calculate the final results

Neural Network (NN)

- Back-propagation, 3-layer network
  - Input, middle, 2 output nodes
- Goal: compute the weights in the NN by back propagation
- What can be done independently?
  - Forward propagation for a given node
  - Computing loss for output nodes and updating links to them
  - Loss at hidden node needs sum of weighted lost at next stage
- Map/Reduce:
  - Compute output loss and update weights
  - For each hidden layer node
    - Map product of weight and output loss
    - Reduce (sum)
    - Compute hidden layer loss