On the Algorithmic & System Interface of BIG LEARNING

Eric Xing
epxing@cs.cmu.edu
School of Computer Science
Carnegie Mellon University

Acknowledgement:
Qirong Ho, Seunghak Lee, Jin Kyu Kim, Wei Dai, Jinliang Wei, Abhimanu Kumar, Pengtao Xie, Xun Zheng and, Phil Gibbons, Greg Ganger, Garth Gibson
One of The First Encounterances of Science with Big Data
Trees Falling in the Forest

"If a tree falls in a forest and no one is around to hear it, does it make a sound?" — George Berkeley

Data ≠ Knowledge

- Nobody Knows What Is In All These Data Unless Having Them Processed and Analyzed
  - Needs a scalable way to automatically search, digest, index, and understand contents
Challenge #1 – Massive Data Scale

~1B nodes, do not fitting into the main memory of a single machine, a familiar problem!
Challenge #2 – Gigantic Model Size

> $10^{11}$ parameters, do not fitting into the main memory of a single machine either!
Challenge #3 – Huge Cognitive Space

1M ~ 1B categories are seen in modern extreme classification problems, kNN? SVM?
The Alg and Sys INTERFACE of Big ML

- Graphical Models
- Nonparametric Bayesian Models
- Regularized Bayesian Methods
- Sparse Structured I/O Regression
- Sparse Coding
- Spectral/Matrix Methods
- Others

Machine Learning Models/Algorithms

Hardware and infrastructure
- Network switches
- Network attached storage
- Infiniband
- Flash storage
- Server machines
- GPUs
- Desktops/Laptops
- NUMA machines
- Cloud compute (e.g. Amazon EC2)
- Virtual Machines
Existing Efforts on Scalable ML

**An Algorithmic Path:**
- Focus on correctness/faster convergence, but assuming an ideal system
- Examples:
  - Cyclic fixed-delay schemes (Langford et al., Agarwal & Duchi)
  - Single-machine asynchronous (Niu et al.)
  - Naively-parallel SGD (Zinkevich et al.)
  - Partitioned SGD (Gemulla et al.)

```
for (t = 1 to T) {
  doThings()
  parallelUpdate(x, θ)
  doOtherThings()
}
```
- May oversimplify systems issues
  - e.g. need machines to perform consistently
  - e.g. need lots of synchronization
  - e.g. or even try not to communicate at all

**A System Path:**
- Focus on high iteration throughput, but assuming correctness of algorithm
- Examples:
  - MapReduce: Hadoop and Mahout
  - Spark
  - Graph-based: GraphLab, Pregel

- May oversimplify ML issues
  - e.g. assume algorithms “just work” in the distributed setting, without proof
  - e.g. must convert programs into their model (MapReduce, graph programs), which can be a nontrivial effort
PETUUM: A General-Purpose Big-ML Framework

API, Tools, UI, Libraries
Practitioner (direct call),
ML Researcher (Matlab-style),
Power User (Low-level API)

Programming Models

BIG-ML Architecture

Resource Allocators

Fault Tolerance
Mathematics 101 for ML

$$\arg \max \limits_{\theta} \equiv \mathcal{L}(\{x_i, y_i\}_{i=1}^{N}; \theta) + \Omega(\theta)$$

Model  \hspace{1cm} Data  \hspace{1cm} Parameter

$$\theta^{t+1} = \theta^t + \Delta_f \theta(D)$$

This computation needs to be parallelized!
Toward Big ML

\[ \mathbf{\bar{\theta}}^{t+1} = \mathbf{\bar{\theta}}^t + \Delta_f \bar{\mathbf{\theta}}(\mathcal{D}) \]

\( \mathcal{D} \equiv \{ \mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n \} \)

\( \mathbf{\bar{\theta}} \equiv [\mathbf{\bar{\theta}}_1^T, \mathbf{\bar{\theta}}_2^T, \ldots, \mathbf{\bar{\theta}}_k^T]^T \)

\( f \equiv \{ f_1, f_2, \ldots, f_m \} \)
Toward Big ML

**Theory:** Degree of parallelism, convergence analysis, sub-sample complexity …

**System:** Distributed architecture: DFS, parameter server, task scheduler…

**Hardware:** GPU, flash storage, cloud …

**Programming model & Interface:** High: Matlab/R Medium: C/JAVA Low: MPI

**Algorithm:** Parallelizable and stochastic MCMC, VI, Opt, Spectrum …

**Model:** Generic building blocks: loss functions, structures, constraints, priors …

**Representation:** Compact and informative features
On Data Parallelism:
Mixed Memberships of Actors in Massive Social Network

- Micro-inference vs. Macro-inference
- Multi-role of every node
- Context dependent role-instantiation
...Leading to Many Applications

1. Community Detection
2. User Personalization
3. Behavior Prediction
4. Info Propagation
Triangular Motifs: More Scalable and Informative Network Features

Adj. Matrix

<table>
<thead>
<tr>
<th>Dest. Node</th>
<th>Edge Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Length $O(N)$ vs. Length $O(Degree^2)$

Triangle features more compact for low node degree!

four edges, five non-edges

2/3-Triangles

<table>
<thead>
<tr>
<th>Dest. Nodes</th>
<th>Triangle Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2,3)</td>
<td></td>
</tr>
<tr>
<td>(2,4)</td>
<td></td>
</tr>
<tr>
<td>(2,9)</td>
<td></td>
</tr>
<tr>
<td>(3,4)</td>
<td></td>
</tr>
<tr>
<td>(3,9)</td>
<td></td>
</tr>
<tr>
<td>(4,9)</td>
<td></td>
</tr>
</tbody>
</table>

one 3-triangle, five 2-triangles
Latent Space Modeling of Network Motifs (e.g., triangles)

- Nodes lie in a **Latent K-dimensional “feature space”**
  - **Features analogous to “roles” or “communities”**
  - “Compatible” node features => high probability of being in a certain form of triangle
- Motif-modeling is more informative and compact than edge-modeling
The MMTM Model


\[ \theta_i \sim \text{Dirichlet}(\alpha) \]
\[ s_{i,jk} \sim \text{Multinomial}(\theta_i) \]
\[ B_{xyz} \sim \text{Dirichlet}(\lambda) \]
\[ E_{ijk} \sim \text{TriangleDistribution}(B, s_{i,jk}, s_{j,ik}, s_{k,ij}) \]

Rao-Blackwellized/Collapsed Gibbs Sampling for inference, with \( \theta \) and \( B \) integrated out
Mixed-Membership Inference Algorithms: MCMC and SVI

Markov Chain Monte Carlo:
Randomly sample each variable in sequence

Stochastic Variational Inference:
Gradient ascent on randomly-chosen variables
Scalable Algorithms

- **δ-subsampling**: down-sample neighborhood of high-degree nodes
  - Pick a constant $\delta$ and subsample $\delta(\delta - 1)/2$ motifs from every node with degree $> \delta$
  - A possible theory of projection invariance

- **Stochastic algorithms**: update small random subset of variables every iteration
  - In topic modeling: Pick a random subset of nodes, and update only their local variables $\theta$, $z$.
  - Fully update global parameters $\phi$ (triangle probs for each global topic) every iteration

- **Parsimonious model**: with linear $O(K)$ number of role parameters
  - **3-roles-same**: $B(a,a,a)$ for each of the $K$ choices of $a \rightarrow K$ parameters
  - **2-roles-same**: $B(a,a,\cdot)$ for each of the $K$ choices of $a$, where $\cdot \neq a \rightarrow K$ parameters
  - **All-roles-different**: $B(\cdot,\cdot,\cdot)$ where all three $\cdot$ are different $\rightarrow 1$ parameter

- **More recent advancements of stochastic inference**:
  - Adaptive learning rate [R. Ranganath, C. Wang, D. Blei and E. P. Xing, ICML 2013]
  - Variance Reduction [C. Wang, X. Chen, A. Smola and E. P. Xing, NIPS 2013]
Gibbs Sampling (with $\delta$-subsampling):

- Stanford web graph, $N \approx 280,000$
  - Converged in 500 Gibbs sampling iterations
  - Runtime: 18 hours using one processor core

Figure 5: $N = 281,903$ Stanford web graph, MMTM mixed-membership visualization.
SVI: Faster & More Accurate


<table>
<thead>
<tr>
<th>Name</th>
<th>Nodes</th>
<th>Edges</th>
<th>δ</th>
<th>2,3-Tris (for δ)</th>
<th>Frac. 3-Tris</th>
<th>Roles K</th>
<th>Threads</th>
<th>Runtime (10 data passes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brightkite</td>
<td>58K</td>
<td>214K</td>
<td>50</td>
<td>3.5M</td>
<td>0.11</td>
<td>64</td>
<td>4</td>
<td>34 min</td>
</tr>
<tr>
<td>Brightkite</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slashdot Feb 2009</td>
<td>82K</td>
<td>504K</td>
<td>50</td>
<td>9.0M</td>
<td>0.030</td>
<td>100</td>
<td>4</td>
<td>2.4 h</td>
</tr>
<tr>
<td>Slashdot Feb 2009</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stanford Web</td>
<td>282K</td>
<td>2.0M</td>
<td>20</td>
<td>11.4M</td>
<td>0.57</td>
<td>5</td>
<td>4</td>
<td>10 min</td>
</tr>
<tr>
<td>Stanford Web</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Berkeley-Stanford Web</td>
<td>685K</td>
<td>6.6M</td>
<td>30</td>
<td>57.6M</td>
<td>0.55</td>
<td>100</td>
<td>8</td>
<td>15.2 h</td>
</tr>
<tr>
<td>Youtube</td>
<td>1.1M</td>
<td>3.0M</td>
<td>50</td>
<td>36.0M</td>
<td>0.053</td>
<td>100</td>
<td>8</td>
<td>9.1 h</td>
</tr>
</tbody>
</table>

Stochastic VI MMSB (Gopalan et al, NIPS 2012) took 8 days using 4 threads

340x speedup!

Gibbs MMTM (Ho et al, NIPS 2012) took 18.5 hours using 1 thread

110x speedup!
The Need for Distributed Computation

- Triangular model SVI can handle 1M node networks with 100 roles in a few hours, on just one machine

- What if we want to analyze 10K roles in a 100M-node network?

  - Memory:
    - $100M \times 10K = 1$ trillion latent states = 4TB of RAM

  - Computation:
    - SVI algorithm analyzes 1M nodes and 100 roles in a few hrs on one machine
    - 100M nodes and 10K roles would require 10K+ hrs on one machine, i.e. yrs!

- Need many machines to satisfy memory and computational requirements!
Data-Distributed Sampling Cycle

Sample $\Omega_t$
Requires a reduction step

Parallelize over worker threads
Share global model parameters via RAM
There Is No Ideal Distributed System!

- Not quite that easy…
- **Two distributed challenges:**
  - Networks are slow
  - “Identical” machines rarely perform equally

**Unequal performance**

**Low bandwidth, High delay**

**Compute vs Network**
LDA 32 machines (256 cores)

- Network waiting time
- Compute time
How to Data-Parallel

- Existing ways are either safe by slow, or fast but risky

- “Partial” synchronicity
  - Spread network comms evenly (don’t sync unless needed)
  - Threads usually shouldn’t wait – but mustn’t drift too far apart!

- Straggler tolerance
  - Slow threads must somehow catch up
Stale Synchronous Parallelism (SSP)

Non-degradation staleness
- Allow threads to run at their own pace, without synchronization
- Fastest/slowest threads not allowed to drift >S iterations apart
- Threads cache local (stale) versions of the parameters, to reduce network syncing

Consequence: Fast threads must check network every iteration, Slow threads check once every S iterations – fewer network accesses, so catch up!
SSP + Parameter Server = Easy Data-Distributed ML

- We implement SSP as a “parameter server” (PS)†, called SSPTable, or, LazyTable
  - Provides all machines with convenient access to global model parameter
  - Can be run on multiple machines – reduces load per machine

- SSPTable allows easy conversion of single-machine parallel ML algorithms
  - “Distributed shared memory” programming style
  - No need for complicated message passing
  - Replace local memory access with PS access

† Ahmed et al. (WSDM 2012), Power and Li (OSDI 2010)
SSPTable Programming

- Just put global parameters in SSPTable! Examples:

  - **Topic Modeling (MCMC)**
    - Topic-word table

  - **Matrix Factorization (SGD)**
    - Factor matrices L, R

  - **Lasso Regression (CD)**
    - Coefficients $\beta$

- SSPTable supports *generic classes* of algorithms
  - With these models as examples
Network Bottlenecks in ML Mitigated

Time Breakdown: Compute vs Network
LDA 32 machines (256 cores), 10% data per iter

Network waiting time
Compute time

BSP is SSP with stale = 0

Network communication is already a huge bottleneck with tens of machines
SSP balances network and compute time
Enjoys Async Speed, But BSP Guarantee

- Massive Data Parallelism
- Effective across different algorithms
Scales With # Machines

LDA on NYtimes dataset
(staleness = 10, 1k docs per core per iteration)

Double # machines:
→ 78% speedup
→ converge in 56% time

SSP computational model enables linear scaling with # machines (up to at least 32)
SSP is Theoretically Sound

Instead of $x_{true}$, SSP sees $x_{stale} = x_{true} + error$

The error caused by staleness is bounded
Over many iterations, average error goes to zero
Why Does SSP Converge?

When a thread reads a parameter, the number of “missing updates” is bounded. Essentially a partial, but bounded, loss of serializability.

Hence the numeric error in the parameter is also bounded.

\[ \epsilon \leq C(2s - 1) \]
Convergence Theorem

- **Want:** minimize convex \( f(x) = \frac{1}{T} \sum_{t=1}^{T} f_t(x) \) (Example: Stochastic Gradient)
  - \( L \)-Lipschitz, problem diameter bounded by \( F^2 \)
  - Staleness \( s \), using \( P \) threads across all machines
  - Use step size \( \eta_t = \frac{\sigma}{\sqrt{t}} \) with \( \sigma = \frac{F}{L \sqrt{2(s+1)P}} \)

- **SSP converges according to**
  - Where \( T \) is the number of iterations

\[
R[X] := \left[ \frac{1}{T} \sum_{t=1}^{T} f_t(\bar{x}_t) \right] - f(x^*) \leq 4FL \sqrt{\frac{2(s+1)P}{T}}
\]

- Note the RHS interrelation between \((L, F)\) and \((s, P)\)
  - An interaction between theory and systems parameters
More Generally ...

- We can prove that for a wide class of iteration- or value- bounded async parallel (BAP) system:

  - BAP converges in expectation with distance from the optimal point going to zero.

  - BAP converges in variance which is bounded from above.

  \[
  V_T[X] := \frac{\sum_{t=1}^{T} [f_t(\tilde{x}_t) - f(x^*)]^2}{T} \leq \frac{L^2F^2}{T} + \frac{4\delta^2L^2P^2K}{T} + \frac{4\delta K^{1/2}P\sqrt{LF}}{\sqrt{T}}
  \]

  - BAP follows a better convergence path with higher quality optima and fewer oscillations.

  \[
  V_T[X] - V_{T-1}[X] \leq \frac{L^2}{T} (L\Omega^*-1 + 2\delta\sqrt{K}(P - 1))^2
  \]
MMTM on SSP-PS: State-of-the-Art Speed and Size

- Livejournal details: 4M nodes, 36M edges, 100 roles
- Friendster sample details: 39M nodes, 180M edges, 50 roles
On Model Parallelism:
Genome-Wide Association Mapping via Structured Sparse Regression

\[ \theta \equiv [\theta_1^T, \theta_2^T, \ldots, \theta_k^T]^T \]

\[
\arg \max_{\beta} \equiv \mathcal{L}(\{x_i, y_i\}; \beta) + \Omega(\beta)
\]
A Massive Optimization Problem

- **E.g., Whole-Genome regression!**
  - And many high-D prediction problems in economics and engineering
    \[ \arg \max_{\beta} L(\{x_i, y_i\}; \beta) + \Omega(\beta) \]

- **Main Difficulties:**
  - Complex loss \( L(\{x_i, y_i\}; \beta) \), (e.g., GMs with intractable factors or loopy): Intractable inference
  - Complex shrinkage \( \Omega(\beta) \), (e.g., overlapping group penalties): Non-differentiable, non-separable

- **Classical approaches are slow**
  - Each iteration is typically quadratic to size of model and and size of data
  - Many iterations

- **Scalable solutions through algorithms:**
  - Alternating Direction Dual Decomposition (AD\(^3\)) [Martins et al, ICML 2011]
  - Smooth Proximal Gradient (SPG) [Chen et al, AOAS 2012]
  - Hierarchical Group Thresholding (HGH) [Lee and Xing, 2012, submitted]

- **What if this is still not faster enough?**
Lasso

$$\min_{\beta} \| y - X\beta \|_2^2 + \lambda \sum_{j} |\beta_j|$$

A huge number of parameters (e.g.) $J = 100M$
Model Parallelism

- Determine the degree of parallelization according to system resources
- Non-uniform execution/update policies
  - Within group – synchronous (i.e., sequential) update
  - Inter group – asynchronous update

Intra-Group domain
Synchronous Execution/Update domain

\[ G_0 = \{b_0, b_1, b_2, b_3, b_4, b_5\} \]
\[ G_1 = \{b_6, b_7, b_8, b_9, b_{10}, b_{11}\} \]

Inter-Group domain
Asynchronous Execution/Update domain
Whole data = \{G_0, G_1\}
But ... there are model-dependencies!

- Concurrent updates of $\beta$ may induce errors

Sequential updates
\[
\beta_1 \\
\beta_2
\]

Concurrent updates
\[
\beta_1 \\
\beta_1 \\
\beta_2 \\
\beta_2
\]

Sync

Induces parallelization error

\[
\beta_1^{(t)} \leftarrow S(x_1^T y - x_1^T x_2 \beta_2^{(t-1)}, \lambda)
\]

Need to check $x_1^T x_2$ before updating parameters
How to Model-Parallel

• Again, existing ways are either safe by slow, or fast but risky

• Need to avoid processing the whole data just for optimal distribution
  • i.e., build expensive data representation on the whole data
  • Compute all variable dependencies

• Dynamic load balance

Graph Partition  ???  Random Partition
Structure-Aware Parallelization (SAP)

Structure-aware partitioning
Load-balancing
Dispatching

[Lee, Kim, Xun, Ho, Gibson and Xing, submitted 2014]
A General Framework

Partitioning data and/or models

Maintaining model consistency

Scheduling updates of variables/params

Data partition

Model partition

Worker

Key-value store

Scheduler
Structure-aware Dynamic Scheduler (STRADS)

Blocks of variables are dispatched to workers taking into account dynamic structures of the problems.
STRADS Lasso Scheduling

• Selector (bootstrapped):
  \[ U = \{ \beta_j \} \sim (\delta \beta_j^{(t-1)})^2 + \eta \]

• Model dependency checker:
  \[ |x_j x_k| < \rho \text{ for all } j \neq k \in U \]
  Discard parameters that violate the above condition

• Dispatcher:
  Send a subset of \( \{ \beta_j \} \) that pass the dependence checker to workers
STRADS versus Shotgun

100M features
9 machines

\[ p(j) \sim \left( \delta \beta_j^{(t-1)} \right)^2 + \eta \]

\[ p(j) \sim \text{uniform} \]

Objective

STRADS Initialization

STRADS
Lasso–RR

Sharp drop
Faster and Better Convergence

- STRADS achieves better speed and objective
Bigger Models Now Manageable

- Massive Model Parallelism
- Effective across different models

**Lasso**

- 9 machines
- 10M features: STRADS, Lasso-RR
- 50M features: STRADS, GraphLab
- 100M features: STRADS, GraphLab

**MF**

- 9 machines
- Ranks: STRADS, GraphLab
  - 80: 6620, 34194
  - 160: 6620, 34194
  - 320: 6620, 34194
  - 1000: 6620, 34194
  - 2000: 6620, 34194

**LDA**

- 64 machines
- Vocab/Topics: STRADS, YahooLDA
  - 2.5M/5k: 6620, 34194
  - 2.5M/10k: 6620, 34194
  - 21.8M/5k: 6620, 34194
  - 21.8M/10k: 6620, 34194

**Notes**

- LDA: YahooLDA failed for large topic and vocabulary
- MF:
  - GraphLab did not converge close to STRADS’s objective at k=80, 160
  - GraphLab failed at k >= 320
Scales With # Machines

As we increase the number of machines, the time to reach a fixed log-likelihood decreases.

Log-Likelihood

Seconds

2.5M vocab, 5K topics

STRADS LDA
Memory Bottleneck in ML Mitigated

STRADS effectively partitions models and data

- STRADS’s memory usage per machine decreases as the number of machines increases
- YahooLDA uses the fixed amount of memory per machine, as each machine stores a duplicated copy of word-topic table
Theorem 1 Suppose $P = \{v_i\}_{i=1}^\ell$ is the set of indices of coefficients updated in parallel at the $t$-th iteration, and $\rho$ is sufficiently small such that $\rho \delta \beta_i^{(t)} \delta \beta_j^{(t)} < \epsilon$, for all $i \neq j \in P$, where $\epsilon$ is a small positive constant. Then, the distribution

$$p(j) \propto \left( \delta \beta_j^{(t)} \right)^2$$

approximately maximizes a lower bound $\mathcal{L}$ to the expected decrease in the objective function $F(\beta^{(t)})$ after updating coefficients indexed by $P$, where $\mathcal{L}$ is defined as

$$\mathcal{L} \leq E_P \left[ F(\beta^{(t)}) - F(\beta^{(t)} + \Delta \beta^{(t)}) \right].$$

(1)
DNN on STRADS: (preliminary)

- Application: phoneme classification in speech recognition.
- Dataset: TIMIT dataset with 1M samples.
- Network configuration: input layer with 440 units, output layer with 1993 units, six hidden layers with 2048 units in each layer

<table>
<thead>
<tr>
<th>Methods</th>
<th>PER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditional Random Field [1]</td>
<td>34.8%</td>
</tr>
<tr>
<td>Large-Margin GMM [2]</td>
<td>33%</td>
</tr>
<tr>
<td>CD-HMM [3]</td>
<td>27.3%</td>
</tr>
<tr>
<td>Recurrent Neural Nets [4]</td>
<td>26.1%</td>
</tr>
<tr>
<td>Deep Belief Network [5]</td>
<td>23.0%</td>
</tr>
<tr>
<td>Petuum DNN (Data Partition)</td>
<td>24.95%</td>
</tr>
<tr>
<td>Petuum DNN (Model Partition)</td>
<td>25.12%</td>
</tr>
</tbody>
</table>

- Results from a 2-Billion parameter DNN available soon
On Task Parallelism: Extreme Classification

- Now we have dealt with high feature dimension
  - Sparsity

- and we have know how to leverage structural knowledge
  - Structured shrinkage

- What about massive concept space?

$f \equiv \{f_1, f_2, \ldots, f_m\}$
Current Practices

- Most popular approaches
  - Nearest neighbor
  - One-vs-all multi-way classification

- Fancier ML: structure regularization
  - i.e. Tree lasso  [Kim and Xing, AOAS 2011]
  - Optimization

- Fancier ML: transfer learning
  - Optimization

\[
\arg \min_{\beta \in \mathbb{R}^J} f(\beta) = \frac{1}{2} \|y - X\beta\|^2_2 + \Omega(\beta)
\]

\[y \in \{1, \ldots, C\}\]

or \[y \in \{0, 1\}^C, \quad \sum_i y_i = 1\]

Is this still a good paradigm?
Output Coding

Every class is now represented by a bit-string
- **Coding**: a codeword is assigned to each class
- **Decoding**: given test data, look for most similar class codeword

Predict bit by bit through binary or ternary classifier – this is much easier than the 1 vs C-1 classifier

Decoding the bit-string – error correcting

Probabilistic decoding

[Zhao and Xing, CVPR 2013]
Classification Accuracy

• Data sets:

<table>
<thead>
<tr>
<th>DATA SET</th>
<th>#CLASS</th>
<th>#TRAIN</th>
<th>#TEST</th>
<th>#FEATURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMOZ-SMALL</td>
<td>1139</td>
<td>6323</td>
<td>1858</td>
<td>1199848</td>
</tr>
<tr>
<td>DMOZ-LARGE</td>
<td>12294</td>
<td>93805</td>
<td>34905</td>
<td>1199856</td>
</tr>
<tr>
<td>FOOD</td>
<td>1308</td>
<td>467374</td>
<td>467374</td>
<td>170006</td>
</tr>
<tr>
<td>IMAGENET</td>
<td>15284</td>
<td>2.5M</td>
<td>0.8M</td>
<td>338163</td>
</tr>
</tbody>
</table>

• Performance:

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>DMOZ-SMALL Top 1</th>
<th>DMOZ-LARGE Top 1</th>
<th>FOOD (%) Top 1</th>
<th>ImageNet Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVR</td>
<td>49.09</td>
<td>62.76</td>
<td>75.02</td>
<td>46.59</td>
</tr>
<tr>
<td>RDOC</td>
<td>58.23</td>
<td>94.87</td>
<td>88.93</td>
<td>67.47</td>
</tr>
<tr>
<td>RSOC</td>
<td>57.70</td>
<td>94.53</td>
<td>86.52</td>
<td>66.88</td>
</tr>
<tr>
<td>SPECOC</td>
<td>55.17</td>
<td>77.90</td>
<td>81.94</td>
<td>58.12</td>
</tr>
<tr>
<td>HieSVM-1</td>
<td>53.16</td>
<td>63.13</td>
<td>82.76</td>
<td></td>
</tr>
<tr>
<td>ECT</td>
<td>52.25</td>
<td>62.92</td>
<td>74.18</td>
<td>45.03</td>
</tr>
<tr>
<td>LET</td>
<td>45.91</td>
<td>59.22</td>
<td>72.91</td>
<td>43.56</td>
</tr>
<tr>
<td>SSOC</td>
<td>31.59</td>
<td>52.47</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Embarrassingly Parallel?

Problem Data Set

Bit-Predic 1  Bit-Predic 2  Bit-Predic 3  Bit-Predic K
Conclusion: ML Programs and Systems
Opportunities

• **A program with an objective or loss function** that measures solution quality, can be optimized via principled algorithmic computations
  - Example Losses: least-squared loss in regression, negative log-likelihood in probabilistic models
  - Example Algorithm: MCMC, VI, CO, MF, Stochastic Inference

• Key Signatures of an ML program
  - ML algorithms are **iterative-convergent**
    - Iterative-convergent algorithms are resilient to errors in updates
  - ML programs are **blocky**
    - Opportunities for efficient parallelization under bounded error
  - ...

• **A general Big ML Framework shall leverage these properties**
An Algorithmic and System interface of A General-Purpose Big-ML Framework

Machine Learning Families
- Graphical Models
- Nonparametric Models
- Regularized Methods
- Large-Margin
- Sparse Structured I/O Regression
- Sparse Coding
- Spectral/Matrix Methods
- Others

Algorithmic Building Blocks
- Distributed MC
- Graph Propagation
- Convex Optimization
- Spectral Algorithms

System Building Blocks
- Big Model System
  - Dynamic Scheduling
  - Adaptive Load-Balancing
  - Client Autonomicity
- Programming Interface
  - For ML Practitioners
  - For ML Scientists
  - APIs for Power Users
- Big Data System
  - Data Partitioning
  - Parameter Server
  - Thread-Level Caching

Hardware and infrastructure
- Network switches
- Infiniband
- Network attached storage
- Flash storage
- Server machines
- GPUs
- Desksops/Laptops
- NUMA machines
- Cloud computing (e.g. Amazon EC2)
- Virtual Machines
A New Framework for Large Scale Parallel Machine Learning
(Petuum.org)
Extreme Scales

Yahoo!

2.5M words
Yahoo LDA
5000 topics

Petuum

21.8M words
Petuum LDA
10000 topics

Largest topic model to date

20x larger than existing topic model
Extreme Scales

Collaborative Filtering on Netflix data
- 0.5M users (Big Data)
- 200K movies (Big Data)
- 40~2000 ranks (Big Model)

GraphLab
- maximum rank = 40

Petuum
- Easily handle rank = 2000

50x larger on the same cluster
Superior Performance

Gene Analysis (LASSO)
- Shotgun (in GraphLab) (144 hrs)
- Petuum (2.7 hrs) 5.2x faster @ 100M dimensions

Collaborative Filtering
- GraphLab (5 mins)
- Petuum (2.8 mins) 2x faster @ 40 ranks

News Analysis (topic model)
- Yahoo LDA (13 mins)
- Petuum (6.5 mins) 2x faster @ 5000 topics
Our Goal: Democratizing ML on Big Data