# Large Scale Machine Learning for Information Retrieval

Bo Long, Liang Zhang
LinkedIn Applied Relevance Science
CIKM 2013, San Francisco

# Structure of This Tutorial

- Part I: Introduction
  - Challenges from Big Data
  - Introduction of Distributed Computing
  - Introduction of Map-Reduce
  - Web Recommender Systems -- Our Motivation Problem
- Part II: Large Scale Logistic Regression
- Part III: Parallel Matrix Factorization
- Part IV: Bag of Little Bootstraps
- Part V: Future of Distributed Computing Infrastructure

# Big Data becoming Ubiquitous

- Bioinformatics
- Astronomy
- Internet
- Telecommunications
- Climatology
- …

# Big Data: Some size estimates

- 1000 human genomes: > 100TB of data (1000 genomes project)
- Sloan Digital Sky Survey: 200GB data per night (>140TB aggregated)
- Facebook: A billion monthly active users
- LinkedIn:  238M members worldwide
- Twitter: 500 million tweets a day
- Over 6 billion mobile phones in the world generating data everyday

# Challenges: Machine Learning for Big Data

- Before:
  - A small/medium sized data set
  - Train models on a single machine
- Now:
  - Almost infinite supply of data
  - Does the problem become easier? (Asymptotic theory)
  - Not quite
    - More data comes with more heterogeneity
    - Need to change our thinking in terms of
      - Models
      - Model fitting approaches
      - Data analysis
      - …

# Some Challenges

- Exploratory Analysis (EDA), Visualization
  - Retrospective (on Terabytes)
  - More Real Time (streaming computations every few minutes/hours)
- Machine Learning Models
  - Scale (computational challenge)
  - Curse of dimensionality
    - Millions of predictors, heterogeneity
  - Temporal and Spatial correlations

# Some Challenges continued

- Experiments
  - To test new methods, test hypothesis from randomized experiments
  - Adaptive experiments
- Forecasting
  - Planning, advertising
- Many more we are not fully well versed in

# Introduction of Distributed Computing

# Distributed Computing for Big Data

- Distributed computing invaluable tool to scale computations for big data

- Some distributed computing models
  - Multi-threading
  - Graphics Processing Units (GPU)
  - Message Passing Interface (MPI)
  - Map-Reduce

# Evaluating a method for a problem

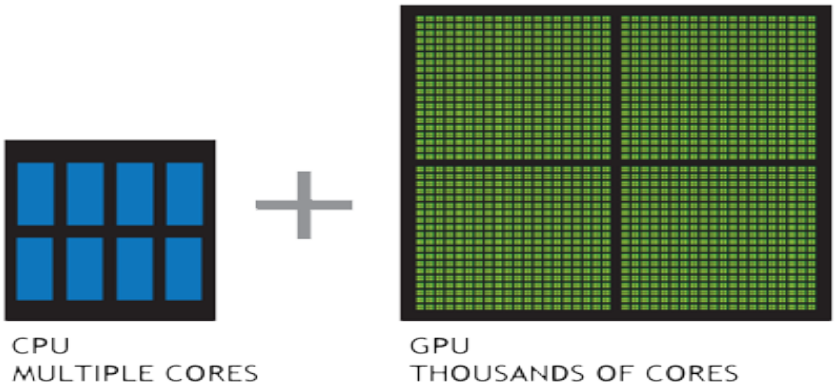
- Scalability
  - Process X GB in Y hours
- Ease of use
- Reliability (fault tolerance)
- Cost
  - Hardware and cost of maintenance
- Good for the computations required?
  - E.g., Iterative versus one pass
- Resource sharing

# Multithreading

- Multiple threads take advantage of multiple CPUs
- Shared memory
- Threads can execute independently and concurrently
- Can only handle Gigabytes of data
- Reliable

# Graphics Processing Units (GPU)

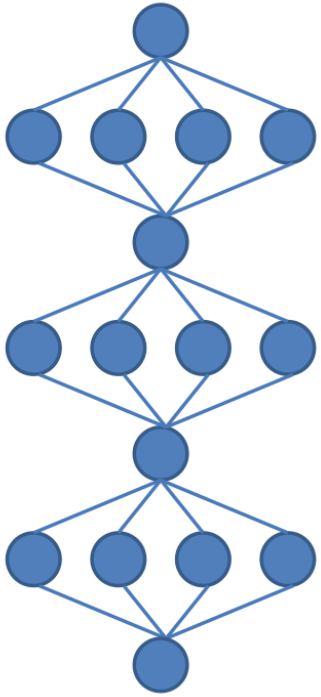
- Number of cores:
  - CPU: Order of 10
  - GPU: smaller cores
    - Order of 1000




- Can be >100x faster than CPU
  - Parallel computationally intensive tasks off-loaded to GPU
- Good for certain computationally-intensive tasks
- Can only handle Gigabytes of data
- Not trivial to use, requires good understanding of low-level architecture for efficient use

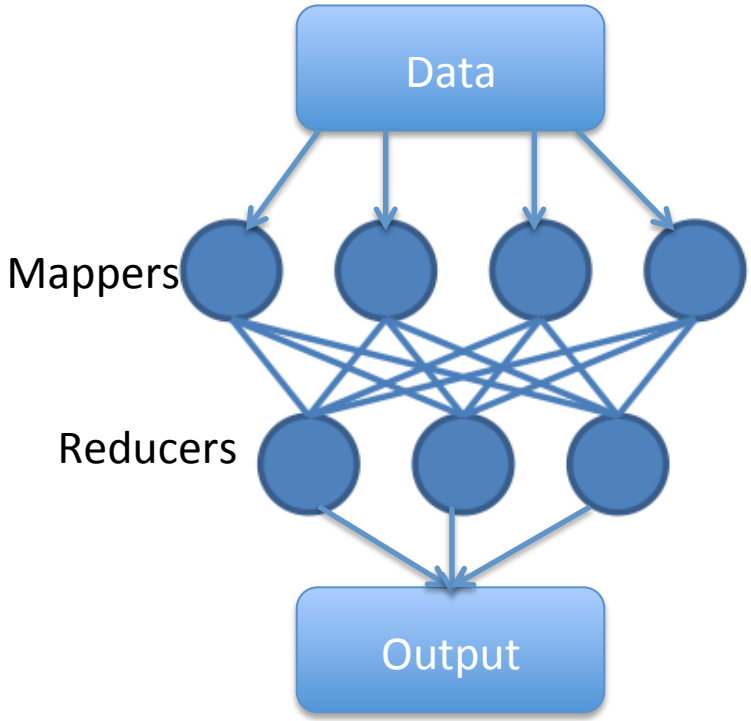
# Message Passing Interface (MPI)

- Language independent communication protocol among processes (e.g. computers)
- Most suitable for master/slave model
- Can handle Terabytes of data
- Good for iterative processing
- Fault tolerance is low



# Map-Reduce (Dean & Ghemawat, 2004)

- Computation split to Map (scatter) and Reduce (gather) stages
- Easy to Use:
  - User needs to implement two functions: Mapper and Reducer
- Easily handles Terabytes of data
- Very good fault tolerance (failed tasks automatically get restarted)



# Comparison of Distributed Computing Methods

| | Multithreading | GPU | MPI | Map-Reduce |
|---|---|---|---|---|
| Scalability (data size) | Gigabytes | Gigabytes | Terabytes | Terabytes |
| Fault Tolerance | High | High | Low | High |
| Maintenance Cost | Low | Medium | Medium | Medium-High |
| Iterative Process Complexity | Cheap | Cheap | Cheap | Usually expensive |
| Resource Sharing | Hard | Hard | Easy | Easy |
| Easy to Implement? | Easy | Needs understanding of low-level GPU architecture | Easy | Easy |

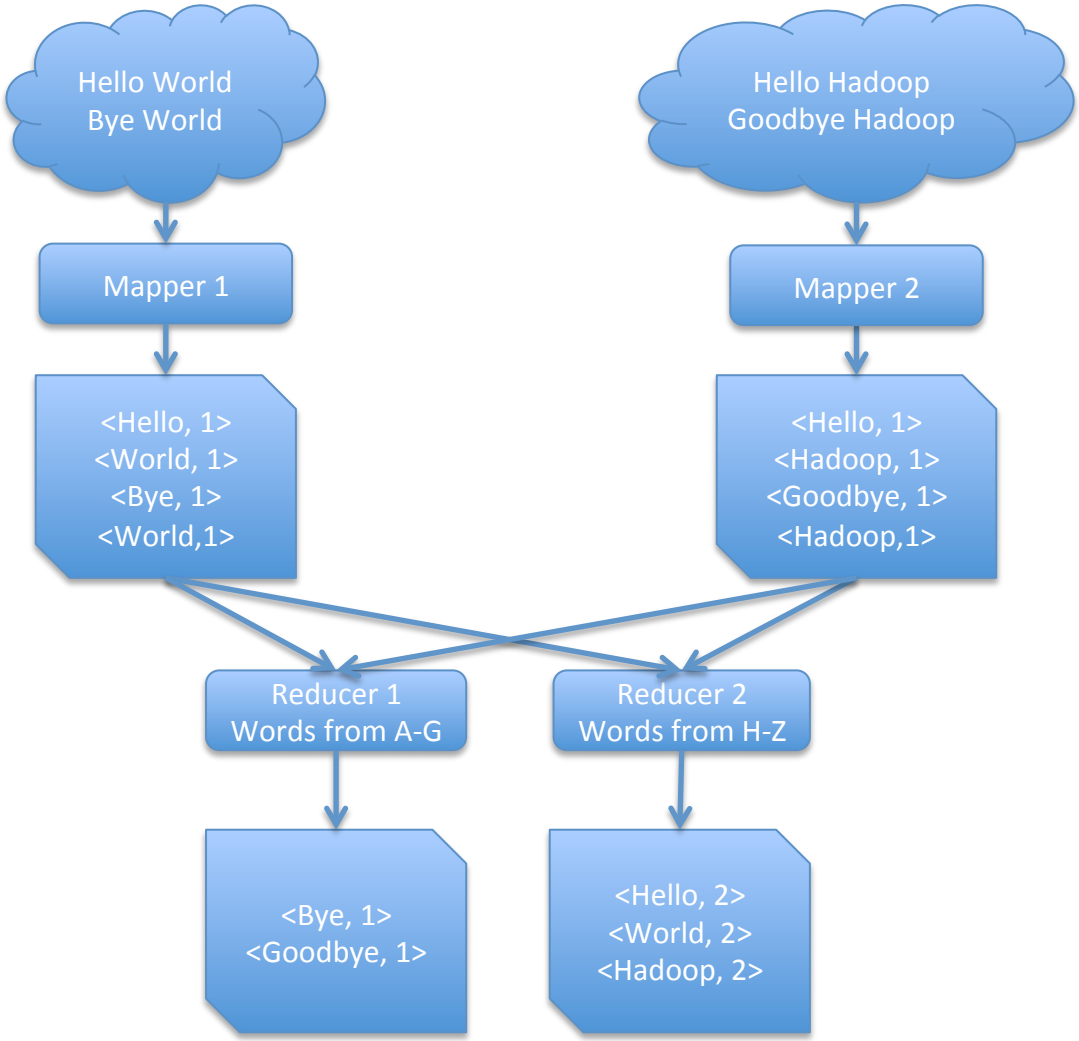
# Introduction of Map-Reduce

# Example Problem

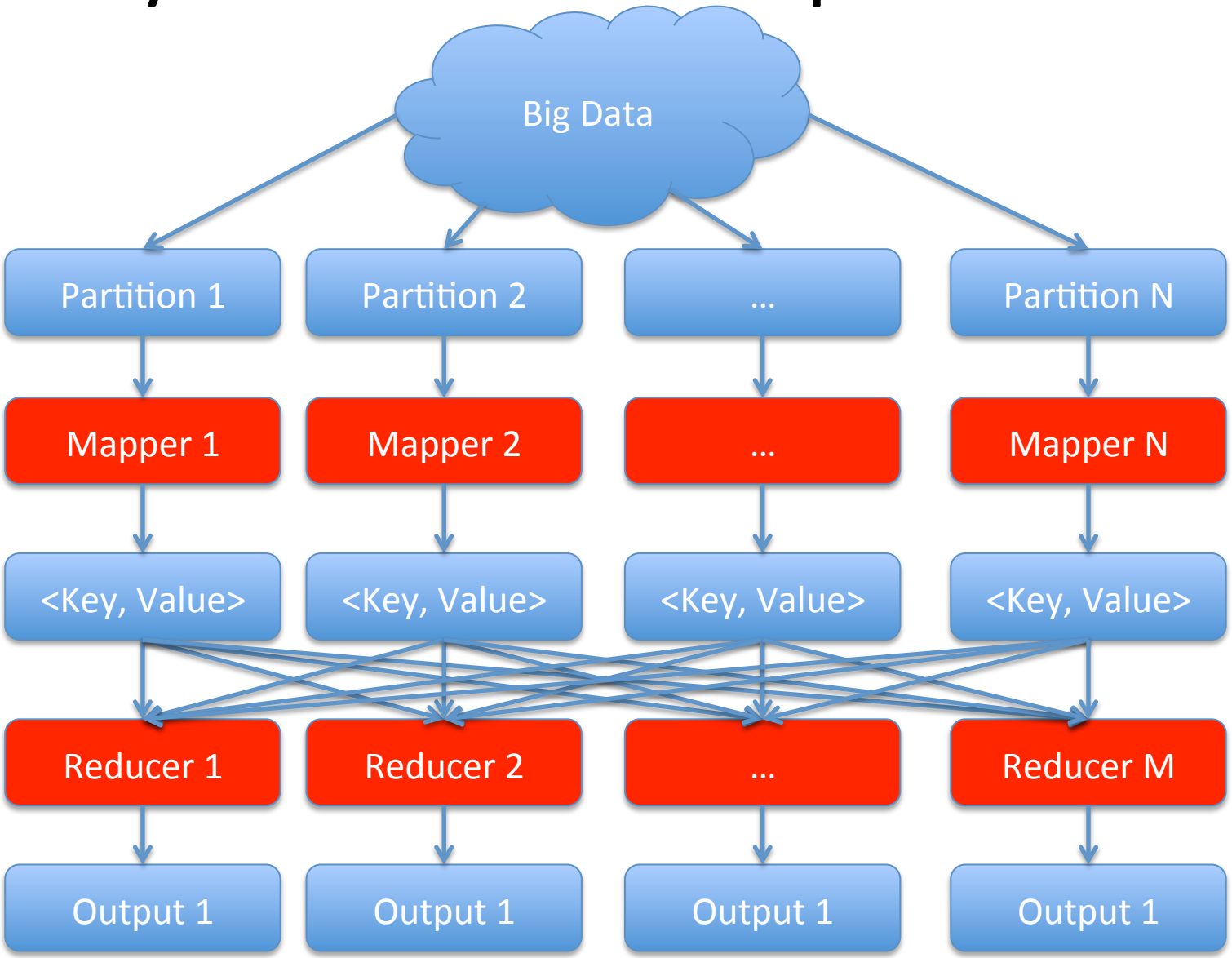
- Tabulating word counts in corpus of documents

- Output: <word, count>

# Word Count Through Map-Reduce



# Key Ideas about Map-Reduce



# Key Ideas about Map-Reduce

- Data are split into partitions and stored in many different machines on disk (distributed storage)
- Mappers process data chunks independently and emit <Key, Value> pairs
- Data with the same key are sent to the same reducer. One reducer can receive multiple keys
- Every reducer sorts its data by key
- For each key, the reducer processes the values corresponding to the key according to the customized *reducer* function and output

# More on Map-Reduce

- Depends on distributed file systems
- Typically mappers are the data storage nodes
- Map/Reduce tasks automatically get restarted when they fail (good fault tolerance)
- Map and Reduce I/O are all on disk
  - Data transmission from mappers to reducers are through disk copy
- Iterative process through Map-Reduce
  - Each iteration becomes a map-reduce job
  - Can be expensive since map-reduce overhead is high

# Web Recommender Systems – Our Motivation Problem

# Web Recommender Systems at LinkedIn

# Similar problem:
# Content recommendation on Yahoo!


Recommend content links (out of 30-40, editorially programmed)

4 slots exposed, F1 has maximum exposure

Routes traffic to other Y! properties

# Web Recommender Systems

Algorithm selects item $j$ from a set of candidates with item features
(e.g. keywords, categories…)

User $i$ visits → $(i, j)$ : response $y_{ij}$ (click or not)

with
user features
(e.g., industry,
behavioral features,
Demographic features,
……)

**Which item should we recommend to the user?**

- The item with highest expected gain of utility
- Sample utilities:
  - CTR
  - Revenue (e.g. CTR * bid)
  - Time spent …

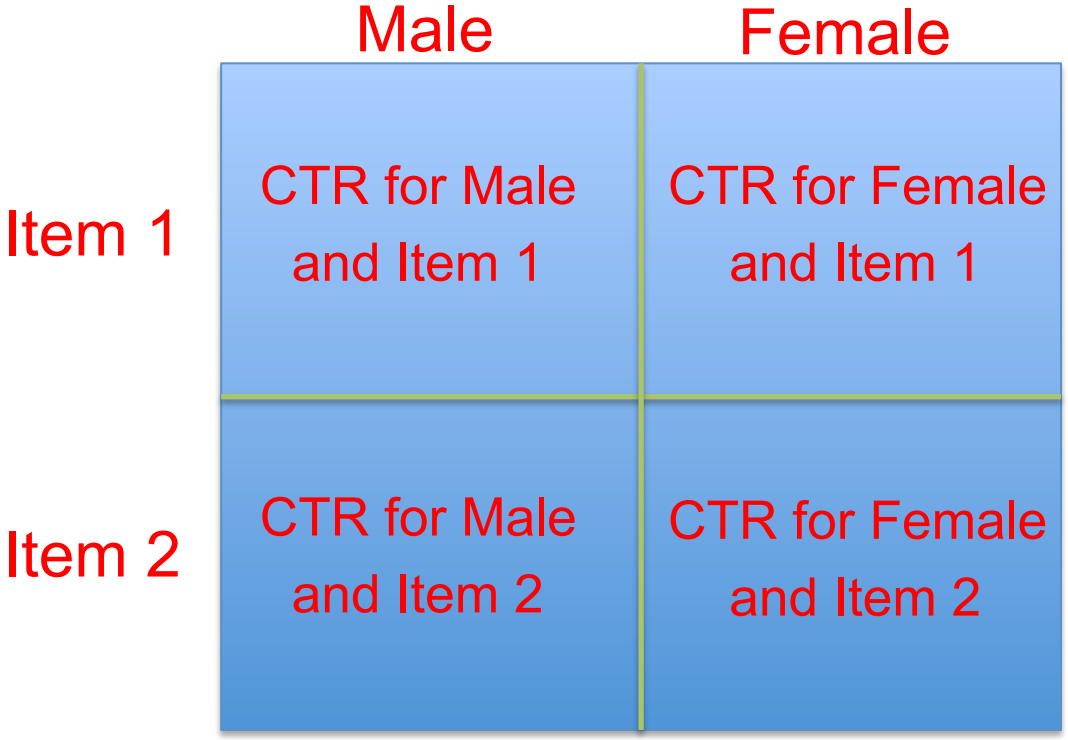
# Challenges of Web Recommender Systems

- Large scale data
- Low latency real-time ranking
- Low positive rates (sparsity problem)
- Properties of the item set to rank per user
  - Dynamic: new items may get added any time
  - Personalized:
    - Ads: advertisers specify targeted user to show their ads
    - Update stream in social network: only from connections
  - May have temporal patterns, e.g. novelty effect
- New users come to the site every second

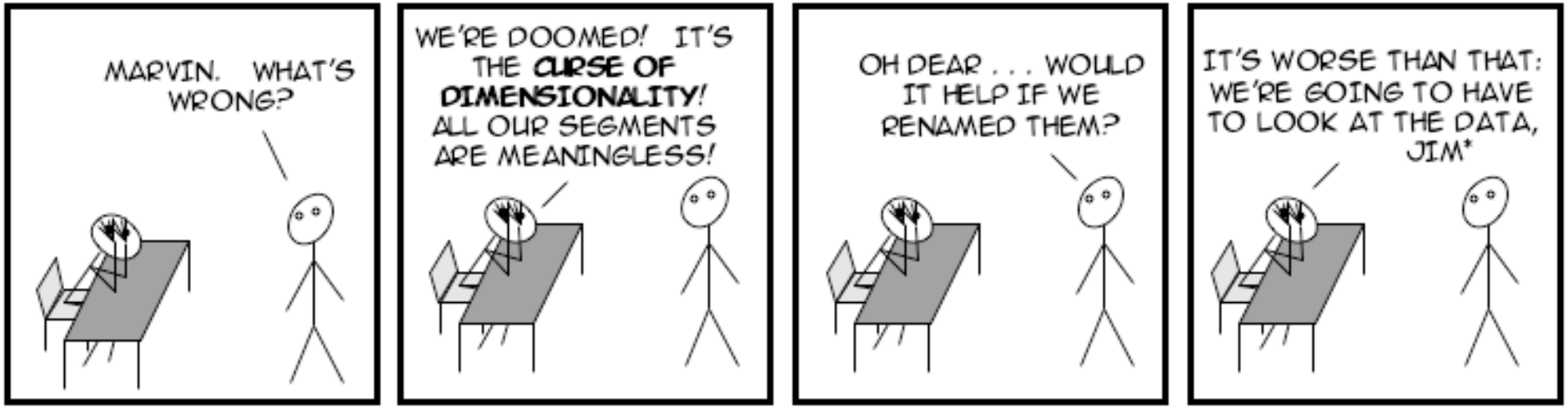
# A Naïve Model (Most-Popular)

- Use empirical CTR for each item to rank
- Item j: $CTR_j$ = $\#Clicks_j$ / # $Views_j$
- Advantage:
  - Quick update of CTR: e.g. every 5-minutes
- Problem:
  - Different users might like different items
  - Cold start & sparsity: many items may have very few clicks

# Segmented Most Popular




- However: Sparsity dramatically increases as the number of features grows
- Curse of dimensionality




HTTP://SCIENTIFICMARKETER.COM



* WITH APOLOGIES TO MR SPOCK & STAR TREK.

# Another Challenge: Data Collection

- Assume most-popular model for all traffic
- Item A: true CTR 5%, 500 clicks, 10000 views
- Item B: true CTR 10%, 0 click, 10 views
- We ends up always serving item A!
- Multi-armed bandit problem (Explore-Exploit):
  - A naïve approach: ε-greedy -- Use x% bucket to serve random traffic
  - Better approaches later in the tutorial

# Challenges from Data Analysis

- Example: How to generate confidence intervals of model performance like AUC?
- Single machine:
  - Bootstrap the test data 100 times
  - Compute the std(AUC) from AUC in the 100 bootstrapped data
- How do we do it for large scale data?
  - Bootstrap non-trivial!

# Large Scale Logistic Regression for Response Prediction in Web Recommender Systems

# Web Recommender Systems at LinkedIn

# Recap: Response Prediction in Web Recommender Systems

Algorithm selects item *j* from a set of candidates with item features
(e.g. keywords, categories…)

User *i* visits → $(i, j)$ : response $y_{ij}$ (click or not)

with
user features
(e.g., industry,
behavioral features,
Demographic features,
……)

**Which item should we recommend to the user?**

- The item with highest expected gain of utility
- Sample utilities:
  - CTR
  - Revenue (e.g. CTR * bid)
  - Time spent …

# Possible Approaches

- Logistic Regression
- Naïve Bayes
- Support Vector Machines (SVM)
- Decision Trees
- Neural Networks
- …

# Challenges of Response Prediction

- Abundance of features
  - Context: time, item placement, IP geo, other content on page
  - Member: gender, age, location, industry, title, function
  - Item: topic, keywords, standardized features
- Need models that can handle sparse data and many features
  - Logistic regression, Decision trees/GBDTs, etc.
- Has to perform at scale
- Logistic regression strikes a good balance
  - Well understood, industry proven
  - Can be scaled for both training and inference
  - Straightforward to reason about model

# Response Prediction for Ads

- Items to rank: Ad campaigns created by advertisers
- Ranking function:
  - predicted CTR X campaign bid
- System shows the top-ranked items to user
- Charge advertisers if clicked
- Second-price auction

# CTR Prediction Model for Ads

- Feature vectors
  - Member feature vector: $x_i$
  - Campaign feature vector: $c_j$
  - Context feature vector: $z_t$
- Model:

$$y_{ijt} \sim Bernoulli(p_{ijt}),$$

$$p_{ijt} = \frac{1}{1+\exp(-s_{ijt})}.$$

$$\begin{aligned} s_{ijt} &= \omega + s_{ijt}^{1,c} + s_{ijt}^{2,c} + s_{ijt}^{2,w} \\ s_{ijt}^{1,c} &= x_i'\alpha + c_j'\beta + z_t'\gamma \\ s_{ijt}^{2,c} &= x_i' A c_j + x_i' C z_t + z_t' B c_j \\ s_{ijt}^{2,w} &= \delta_j + x_i'\eta_j + z_t'\xi_j \end{aligned}$$

# CTR Prediction Model for Ads

- Feature vectors
  - Member feature vector: $x_i$
  - Campaign feature vector: $c_j$
  - Context feature vector: $z_k$
- Model:

$$y_{ijt} \sim Bernoulli(p_{ijt}),$$

$$p_{ijt} = \frac{1}{1+\exp(-s_{ijt})}.$$

$$s_{ijt} = \omega + s^{1,c}_{ijt} + s^{2,c}_{ijt} + s^{2,w}_{ijt}$$

$$s^{1,c}_{ijt} = x'_i\alpha + c'_j\beta + z'_t\gamma$$

$$s^{2,c}_{ijt} = x'_i A c_j + x'_i C z_t + z'_t B c_j$$

$$s^{2,w}_{ijt} = \delta_j + x'_i\eta_j + z'_t\xi_j$$

Cold-start component

Warm-start per-campaign component

# CTR Prediction Model for Ads

- Feature vectors
  - Member feature vector: $x_i$
  - Campaign feature vector: $c_j$
  - Context feature vector: $z_k$
- Model:

$$y_{ijt} \sim Bernoulli(p_{ijt}),$$

$$p_{ijt} = \frac{1}{1+\exp(-s_{ijt})}.$$

$$\begin{aligned} s_{ijt} &= \omega + s^{1,c}_{ijt} + s^{2,c}_{ijt} + s^{2,w}_{ijt} \\ s^{1,c}_{ijt} &= x_i'\alpha + c_j'\beta + z_t'\gamma \\ s^{2,c}_{ijt} &= x_i' A c_j + x_i' C z_t + z_t' B c_j \\ s^{2,w}_{ijt} &= \delta_j + x_i'\eta_j + z_t'\xi_j \end{aligned}$$

Cold-start:

$$\Theta_c = \{\omega, \alpha, \beta, \gamma, A, B, C\}$$

Warm-start:

$$\Theta_w = \{\delta_j, \eta_j, \xi_j\}, j = 1, \cdots, J.$$

Both can have L2 penalties.

Cold-start component

Warm-start per-campaign component

# Model Fitting

- Single machine (well understood)
  - conjugate gradient
  - L-BFGS
  - Trusted region
  - …
- Model Training with Large scale data
  - Cold-start component $\Theta_c$ is more stable
    - Weekly/bi-weekly training good enough
    - However: difficulty from need for large-scale logistic regression
  - Warm-start per-campaign model $\Theta_w$ is more dynamic
    - New items can get generated any time
    - Big loss if opportunities missed
    - Need to update the warm-start component as frequently as possible

# Model Fitting

- Single machine (well understood)
  - conjugate gradient
  - L-BFGS
  - Trusted region
  - …
- Model Training with Large scale data
  - Cold-start component $\Theta_c$ is more stable
    - Weekly/bi-weekly training good enough
    - However: difficulty from need for large-scale logistic regression
  - Warm-start per-campaign model $\Theta_w$ is more dynamic
    - New items can get generated any time
    - Big loss if opportunities missed
    - Need to update the warm-start component as frequently as possible

Large Scale Logistic Regression

Per-item logistic regression given $\Theta_c$

# Challenges of Large Scale Logistic Regression

- Logistic regression for small data sets has been studied and widely applied since 1970s
- Unknown weights can be learned through Newton method, conjugate gradient ascent, coordinate descent, etc.
- However, for web applications we usually see
  - Hundreds of millions of samples
  - Hundreds of thousands of features
- Q: How to fit large scale Logistic regression in a scalable way?

# Large Scale Logistic Regression

- Naïve:
  - Partition the data and run logistic regression for each partition
  - Take the mean of the learned coefficients
  - Problem: Not guaranteed to converge to the model from single machine!
- Alternating Direction Method of Multipliers (ADMM)
  - Boyd et al. 2011
  - Set up a constraint that each partition's coefficient = global consensus
  - Solve the optimization problem using Lagrange Multipliers

# Model Fitting for Large-Scale Ads Data

**Algorithm 1** LASER Model Fitting Algorithm

**for** every D days **do**
  Obtain the most recent data and use ADMM to fit $\mathbf{\Theta}_c$ and $\mathbf{\Theta}_w$ simultaneously by treating item-id as features.
  **for** every M minutes **do**
    Obtain the most recent data
    Split the data by item id
    **for** each item $j$ in parallel **do**
      Treat $\mathbf{\Theta}_c$ as a given offset and fit (or update) $\{\delta_j, \eta_j, \xi_j\}$ (i.e. $\mathbf{\Theta}_w$ for item j)
    **end for**
  **end for**
**end for**

# A Deep-dive to ADMM

# Dual Ascent Method

- Consider a convex optimization problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & Ax = b, \end{array}$$

- Lagrangian for the problem:

$$L(x, y) = f(x) + y^T(Ax - b)$$

- Dual Ascent:

$$\begin{aligned} x^{k+1} &:= \operatorname*{argmin}_x L(x, y^k) \\ y^{k+1} &:= y^k + \alpha^k(Ax^{k+1} - b), \end{aligned}$$

# Augmented Lagrangians

- Bring robustness to the dual ascent method
- Yield convergence without assumptions like strict convexity or finiteness of f
- $L_\rho(x, y) = f(x) + y^T(Ax - b) + (\rho/2)\|Ax - b\|_2^2,$

  $\text{where } \rho > 0 \text{ is called the } \textit{penalty parameter}.$
- The value of ρ influences the convergence rate

# Alternating Direction Method of Multipliers (ADMM)

- Problem:

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Ax + Bz = c \end{array}$$

- Augmented Lagrangians

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + (\rho/2)\|Ax + Bz - c\|_2^2.$$

- ADMM:

$$\begin{aligned} x^{k+1} &:= \operatorname*{argmin}_x L_\rho(x, z^k, y^k) \\ z^{k+1} &:= \operatorname*{argmin}_z L_\rho(x^{k+1}, z, y^k) \\ y^{k+1} &:= y^k + \rho(Ax^{k+1} + Bz^{k+1} - c), \end{aligned}$$

# Large Scale Logistic Regression via ADMM

- Notation:
  - (A, b): data
  - $x_i$: Coefficients for partition i
  - z: Consensus coefficient vector
  - r(z): penalty component such as $||z||_2$
- Problem

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} l_i(A_i x_i - b_i) + r(z) \\ \text{subject to} & x_i - z = 0, \quad i = 1, \ldots, N, \end{array}$$

- ADMM

$$x_i^{k+1} := \underset{x_i}{\operatorname{argmin}} \left( l_i(A_i x_i - b_i) + (\rho/2)\|x_i - z^k + u_i^k\|_2^2 \right)$$

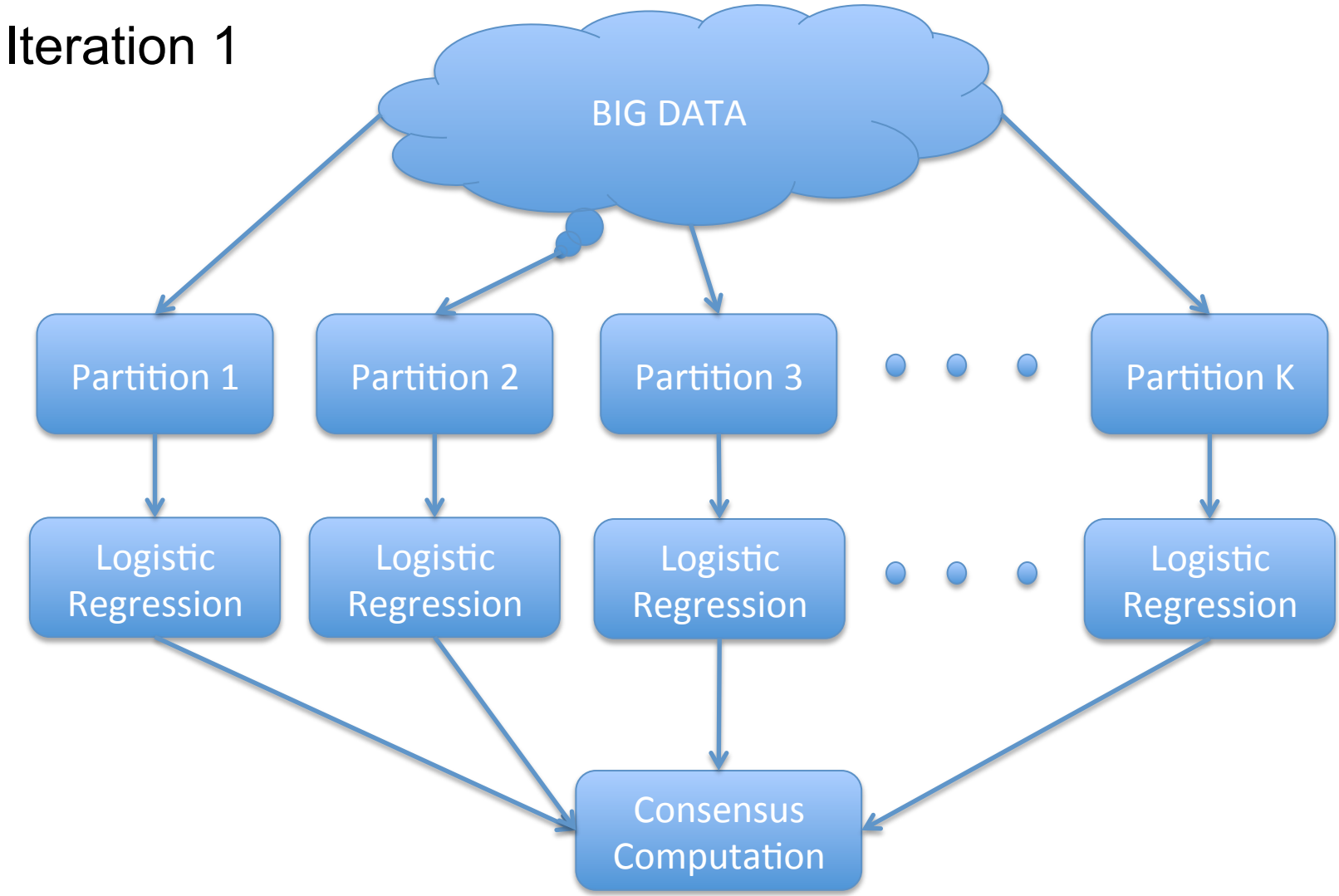
Per-partition Regression

$$z^{k+1} := \underset{z}{\operatorname{argmin}} \left( r(z) + (N\rho/2)\|z - \overline{x}^{k+1} - \overline{u}^k\|_2^2 \right)$$

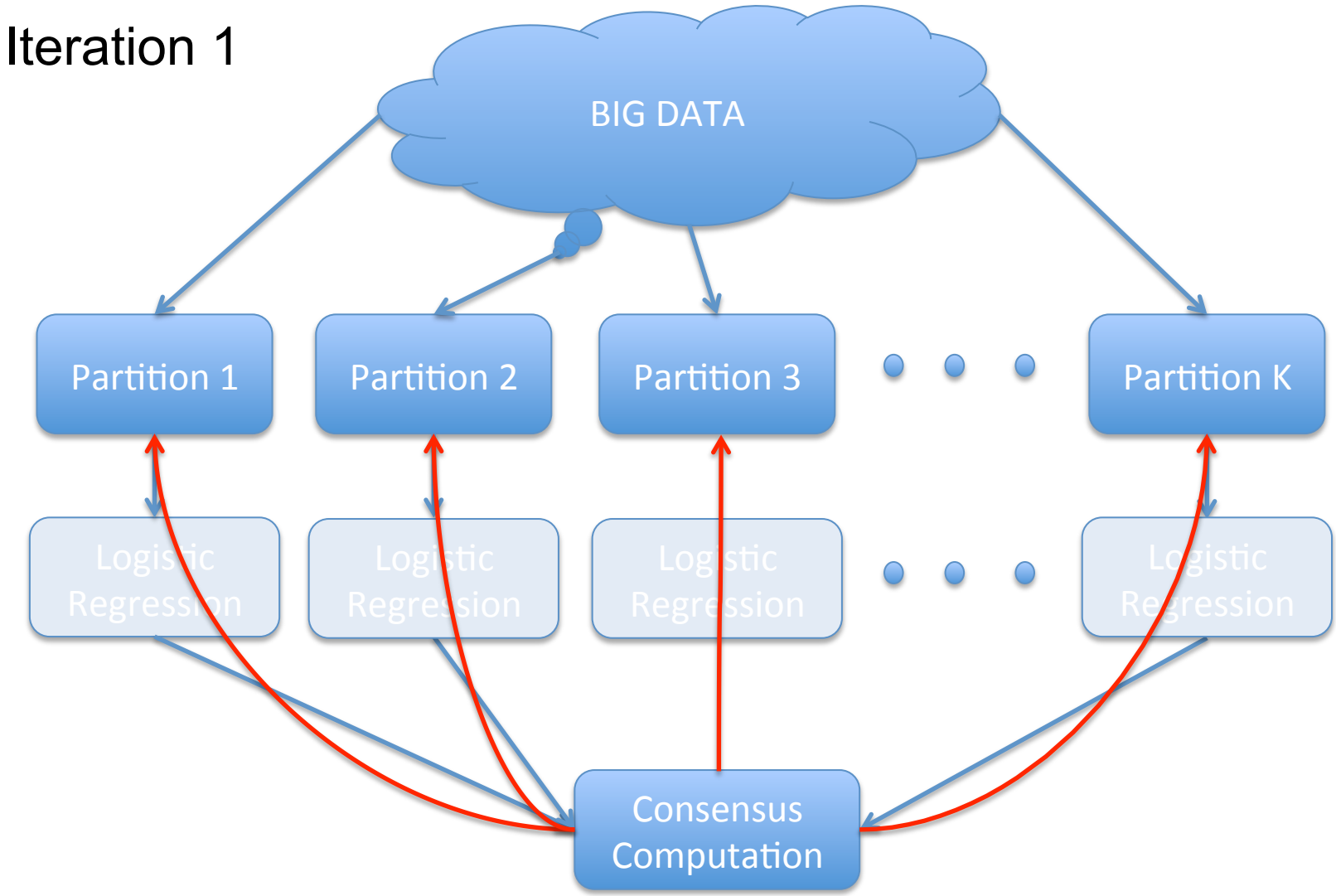
$$u_i^{k+1} := u_i^k + x_i^{k+1} - z^{k+1}.$$

Consensus Computation

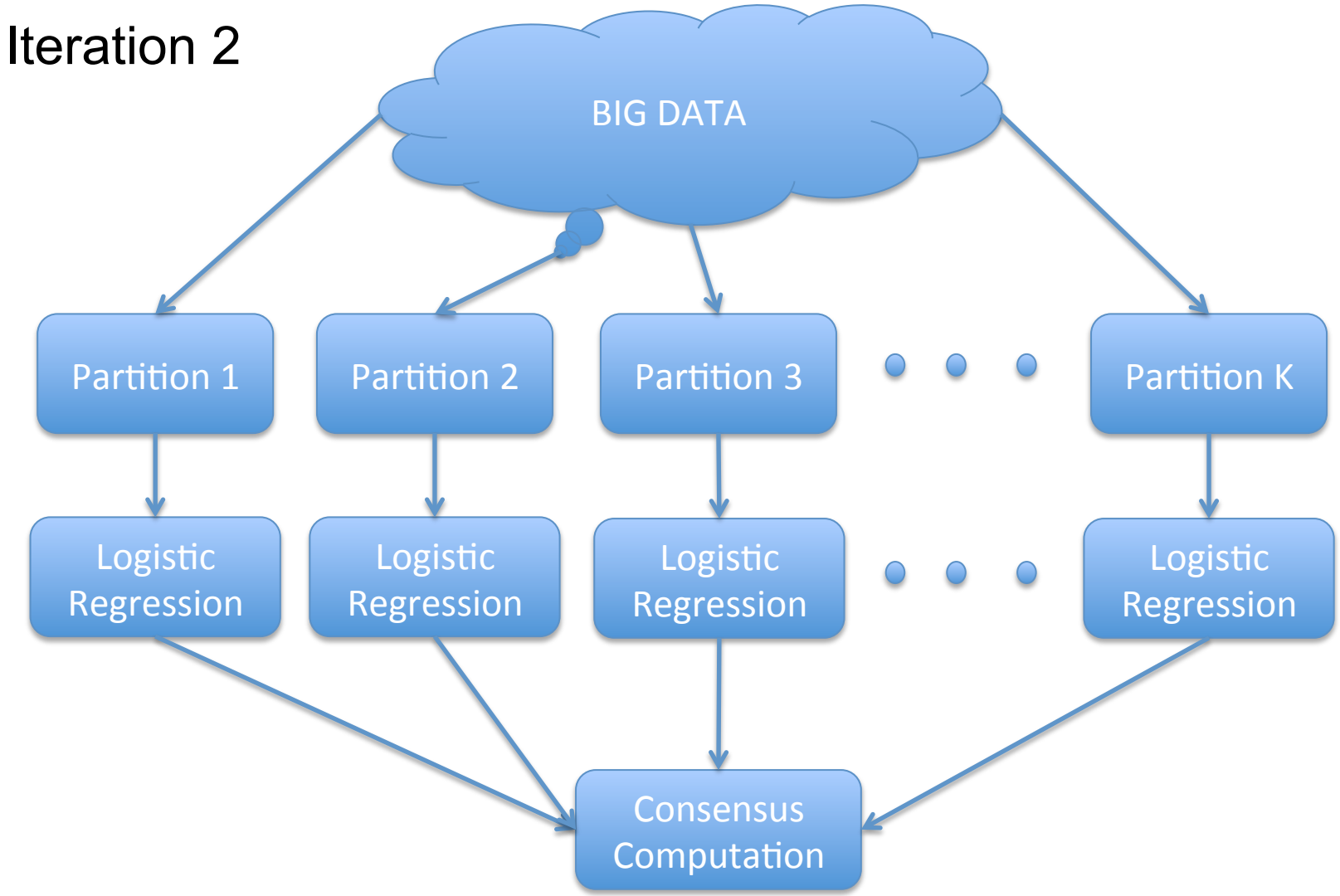
# Large Scale Logistic Regression via ADMM



# Large Scale Logistic Regression via ADMM



# Large Scale Logistic Regression via ADMM



# An Example Implementation

- ADMM for Logistic regression model fitting with L2 penalty

- Each iteration of ADMM is a Map-Reduce job
  - Mapper: partition the data into K partitions
  - Reducer: For each partition, use liblinear/glmnet to fit a L2 logistic regression
  - Gateway: consensus computation by results from all reducers, and sends back the consensus to each reducer node

# Better Convergence Can Be Achieved By

1. Better Initialization: Use results from Naïve method to initialize the parameters
2. Adaptively change step size (ρ) for each iteration based on the convergence status of the consensus

- ADMM-M: ADMM with tuning 1
- ADMM-MA: ADMM with tuning 1+2

# Evaluation of ADMM Convergence

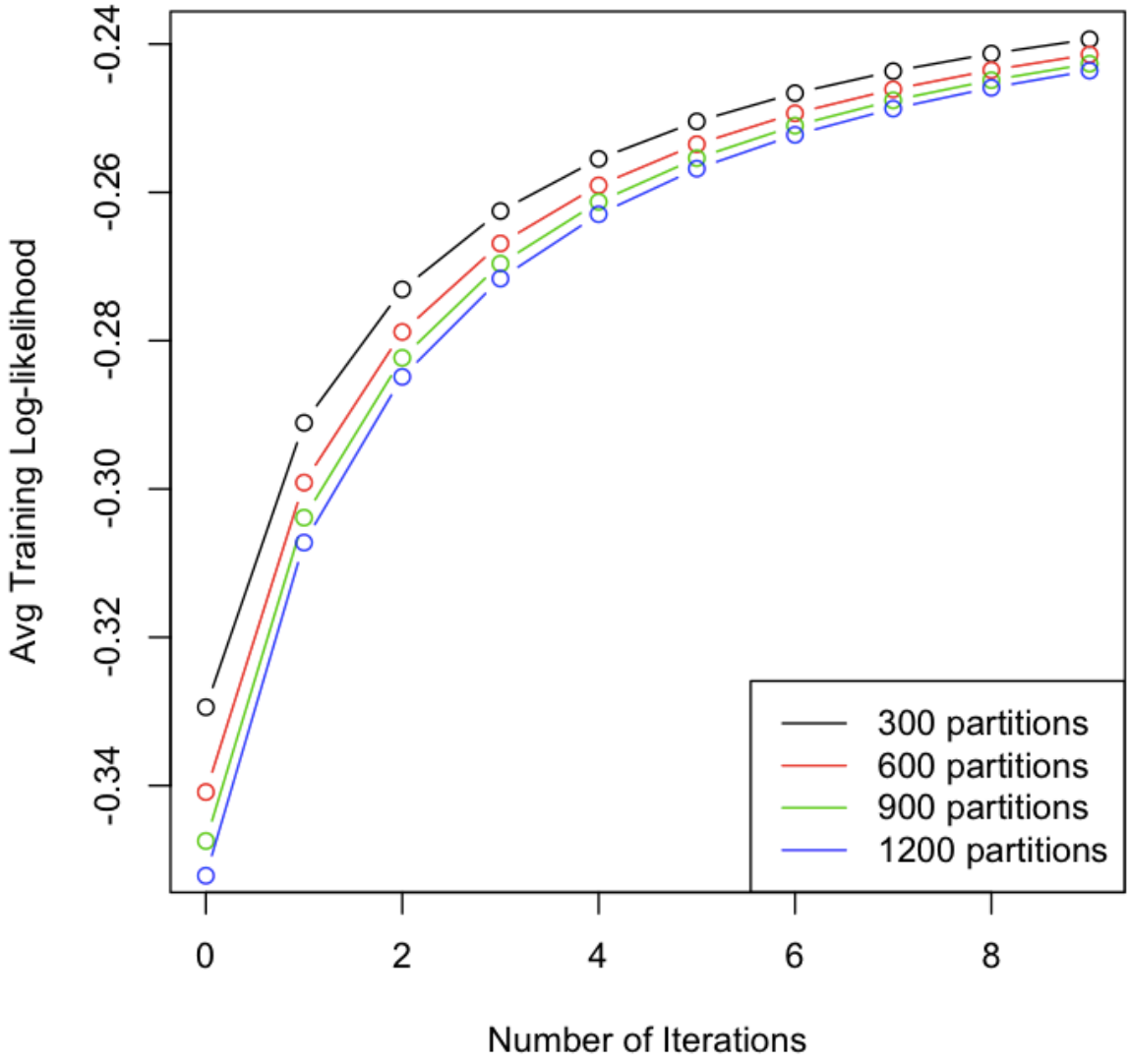
# KDD CUP 2010 Data

- Bridge to Algebra 2008-2009 data in

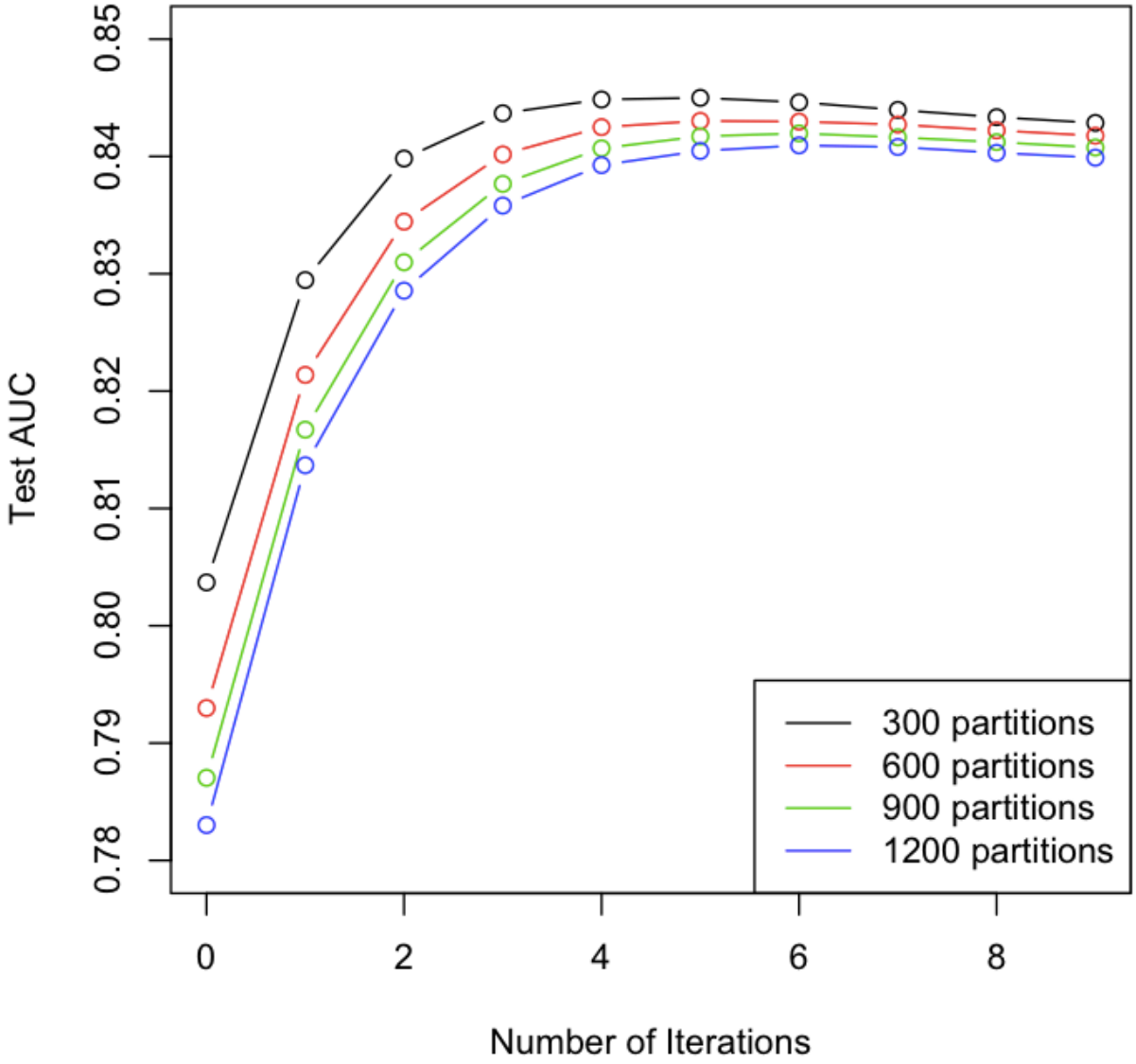
https://pslcdatashop.web.cmu.edu/KDDCup/downloads.jsp

- Binary response, 20M covariates
- Only keep covariates with >= 10 occurrences => 2.2M covariates
- Training data: 8,407,752 samples
- Test data : 510,302 samples

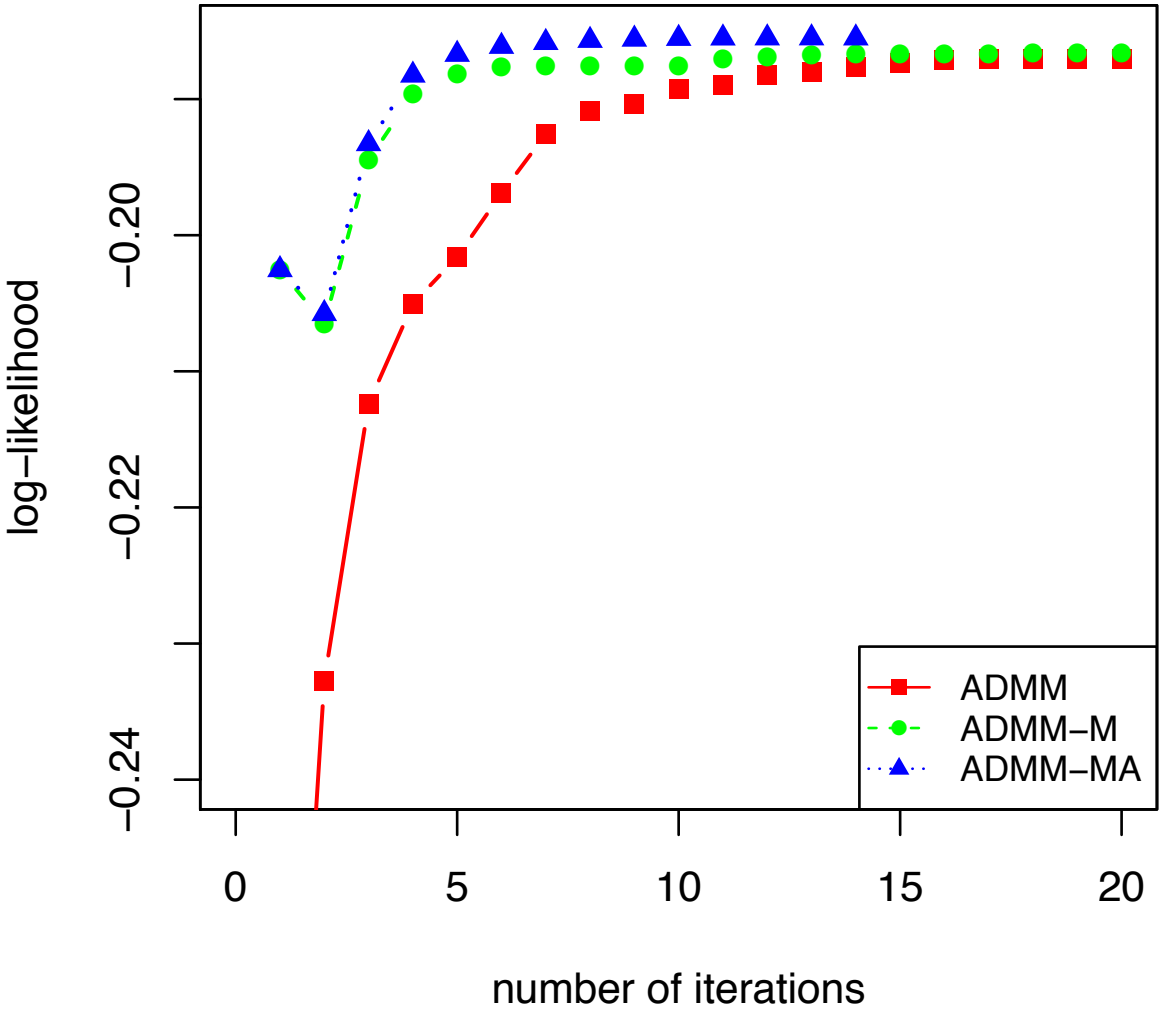
# Avg Training Loglikelihood vs Number of Iterations



# Test AUC vs Number of Iterations



# Offline Ads Data: Effect of ADMM Convergence Tunings



# Explore-Exploit

# Recap: Challenges from Data Collection

- Assume most-popular model for all traffic
- Item A: true CTR 5%, 500 clicks, 10000 views
- Item B: true CTR 10%, 0 click, 10 views
- We ends up always serving item A!
- Multi-armed bandit problem (Explore-Exploit):
  - A naïve approach: ε-greedy -- Use x% bucket to serve random traffic
  - Q: Can we do better?
    - Number of items are very large and ads clicks are very sparse
    - ε-greedy will also have revenue loss

# Existing Approaches for Explore-Exploit

- ε-greedy
- UCB (Upper Confidence Bound)
- Gittins Index
- Thompson sampling

# Thompson Sampling (Thompson 1933)

- Basic Idea:
  - For every user request & item to rank, draw a sample from posterior of $\Theta_c$ and $\Theta_w$
  - Compute CTR using the sampled coefficients instead of posterior mode
  - More exploration for items that have high posterior variances due to small sample sizes
- Has been proven to be optimal for large scale data (Chapelle and Li 2011)
- Our approach
  - $\Theta_c$ is learned from a lot of data, posterior variance can be approximated as 0
  - $\Theta_w$ is campaign-specific, need to draw sample from its posterior that is learned from per-item logistic regression
  - Laplace approximation for computing posterior variance

# Experiments

# Models Considered

- CONTROL: per-campaign CTR counting model
- COLD-ONLY: only cold-start component
- LASER: our model (cold-start + warm-start)
- LASER-EE: our model with Explore-Exploit using Thompson sampling

# Metrics

- Model metrics
  - Test Log-likelihood
  - AUC/ROC
  - Observed/Expected ratio
- Business metrics (Online A/B Test)
  - CTR
  - CPM (Revenue per impression)

# Observed / Expected Ratio

- Observed: #Clicks in the data
- Expected: Sum of predicted CTR for all impressions
- Not a “standard” classifier metric, but in many ways more useful for this application
- What we usually see: Observed / Expected < 1

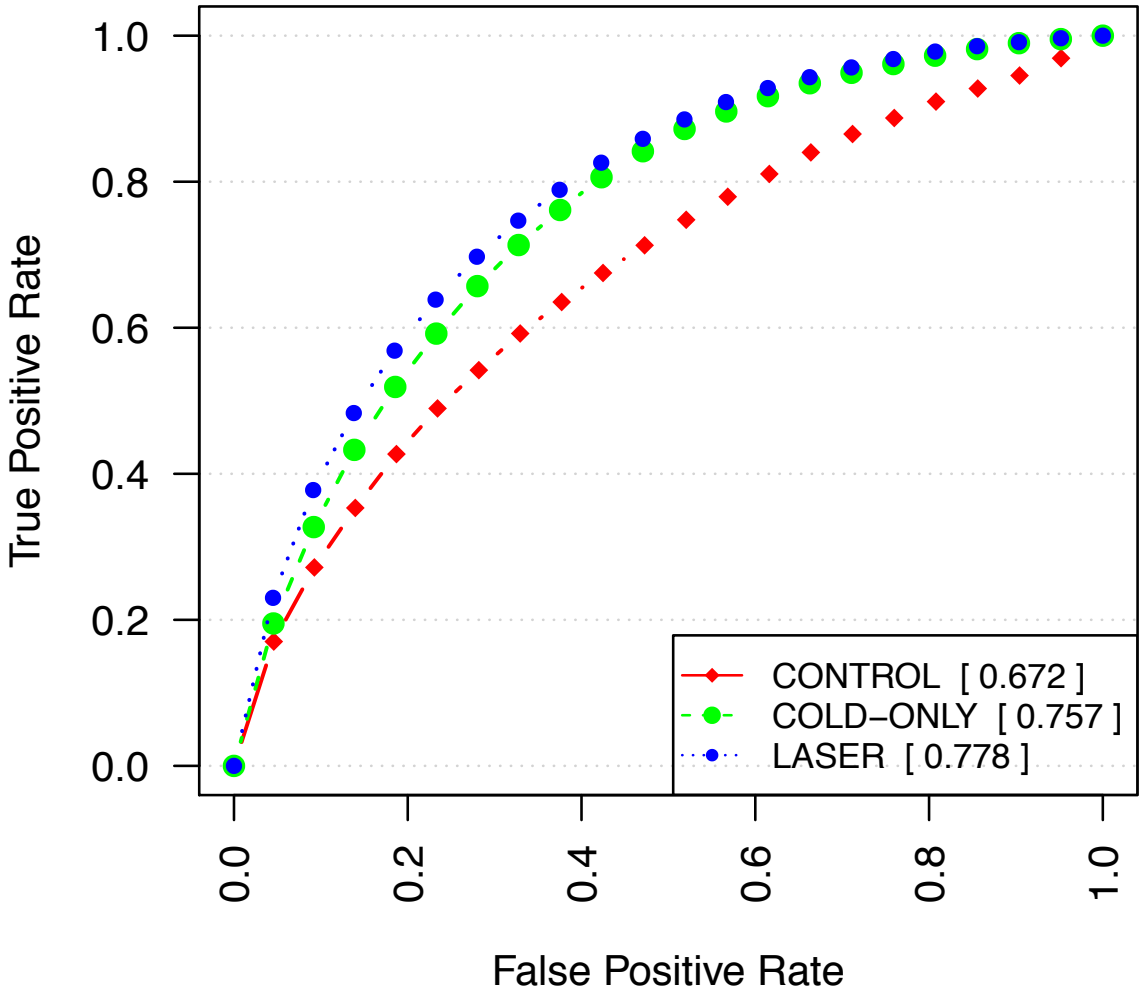
# Observed / Expected Ratio

- Over-prediction is a serious problem with ads
  - Instance of the "winner's curse"
  - When choosing from among thousands of candidates, an item with mistakenly over-estimated CTR may end up winning the auction
- Particularly helpful for examining per-segment data
  - E.g. by bid, number of impressions in training (warmness), geo, etc.
  - Allows us to see where the model might be giving too much weight to the wrong campaigns
- High correlation between O/E ratio and model performance online

# Offline Experiment

- 45 Days of advertising events log
- First 30 days for training data
- Last 15 days for testing data
- After down sampling, both train and test data have hundreds of millions of events
- Thousands of features

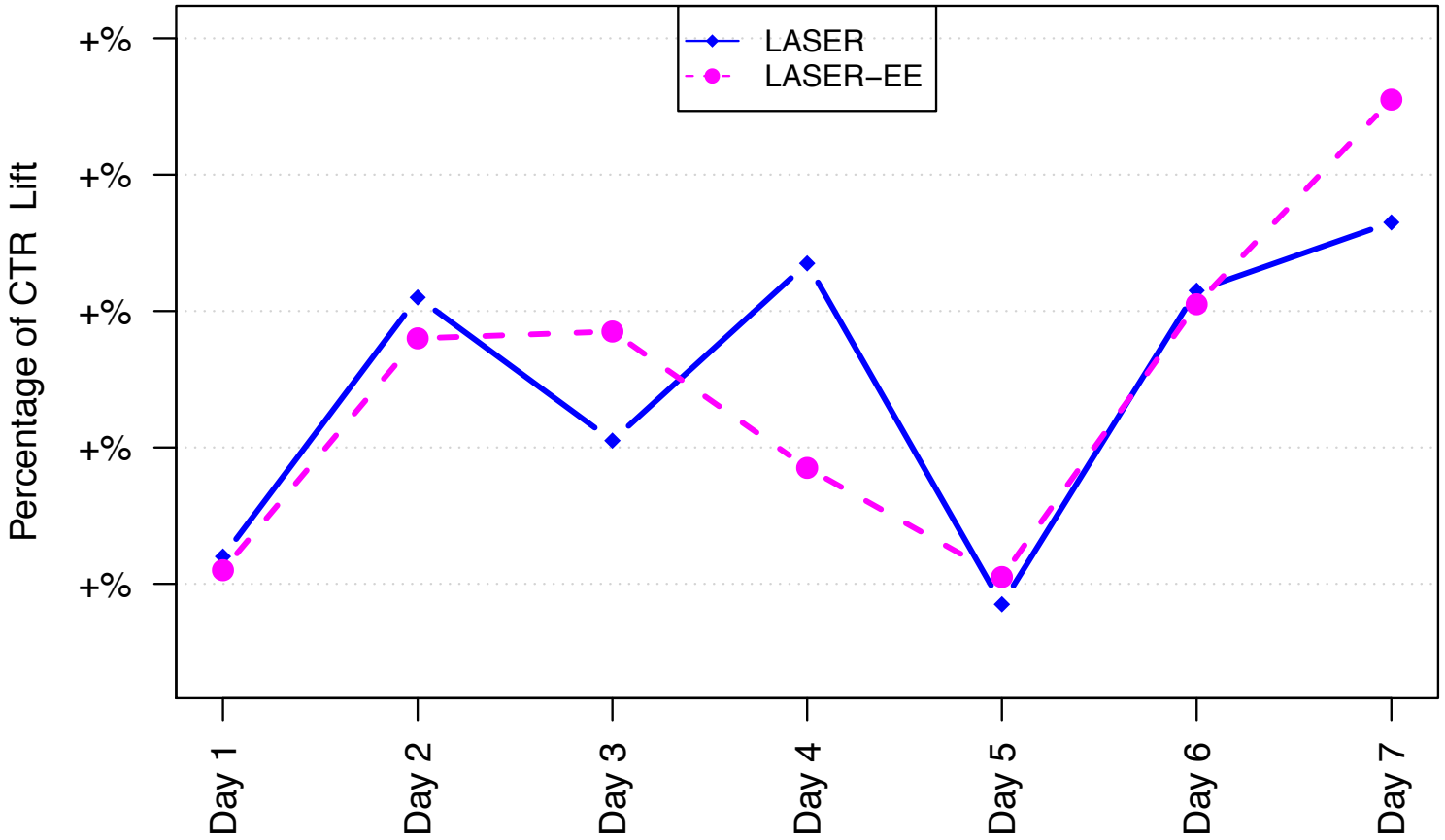
# Offline: ROC Curves



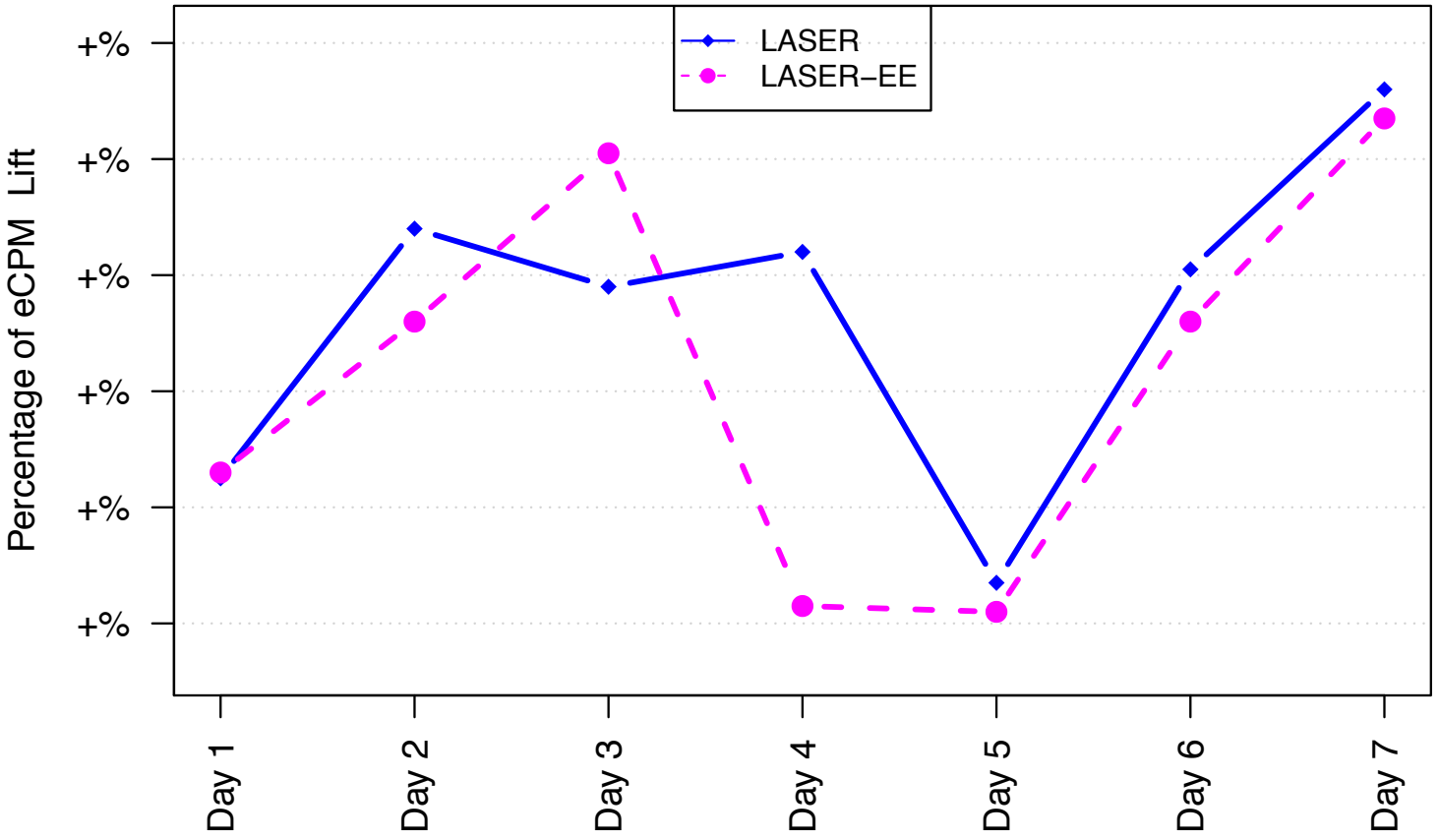
# Online A/B Test

- Three models
  - CONTROL (10%)
  - LASER (85%)
  - LASER-EE (5%)
- Segmented Analysis
  - 8 segments by campaign warmness
    - Degree of warmness: the number of training samples available in the training data for the campaign
    - Segment #1: Campaigns with almost no data in training
    - Segment #8: Campaigns that are served most heavily in the previous batches so that their CTR estimate can be quite accurate

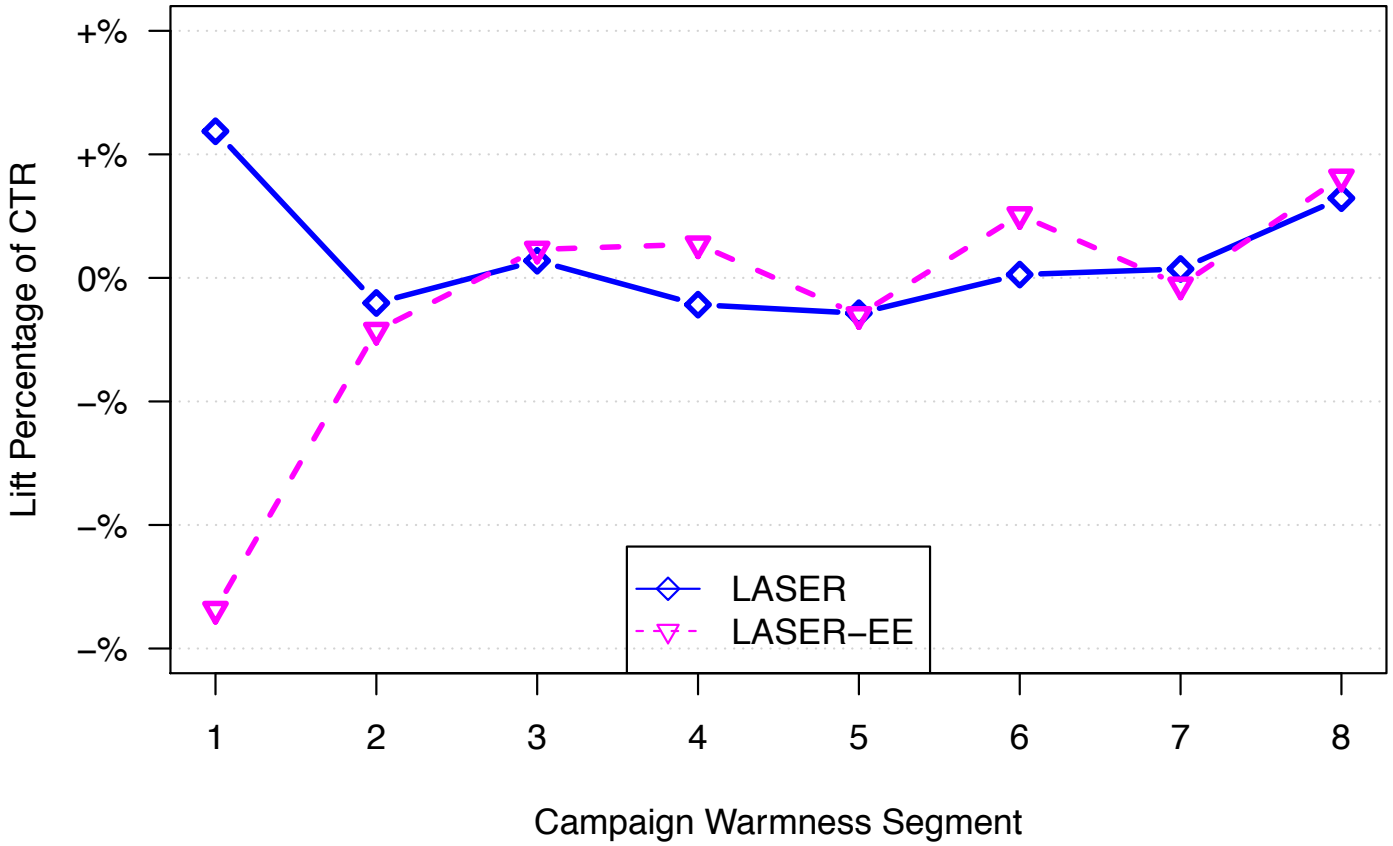
# Daily CTR Lift Over Control



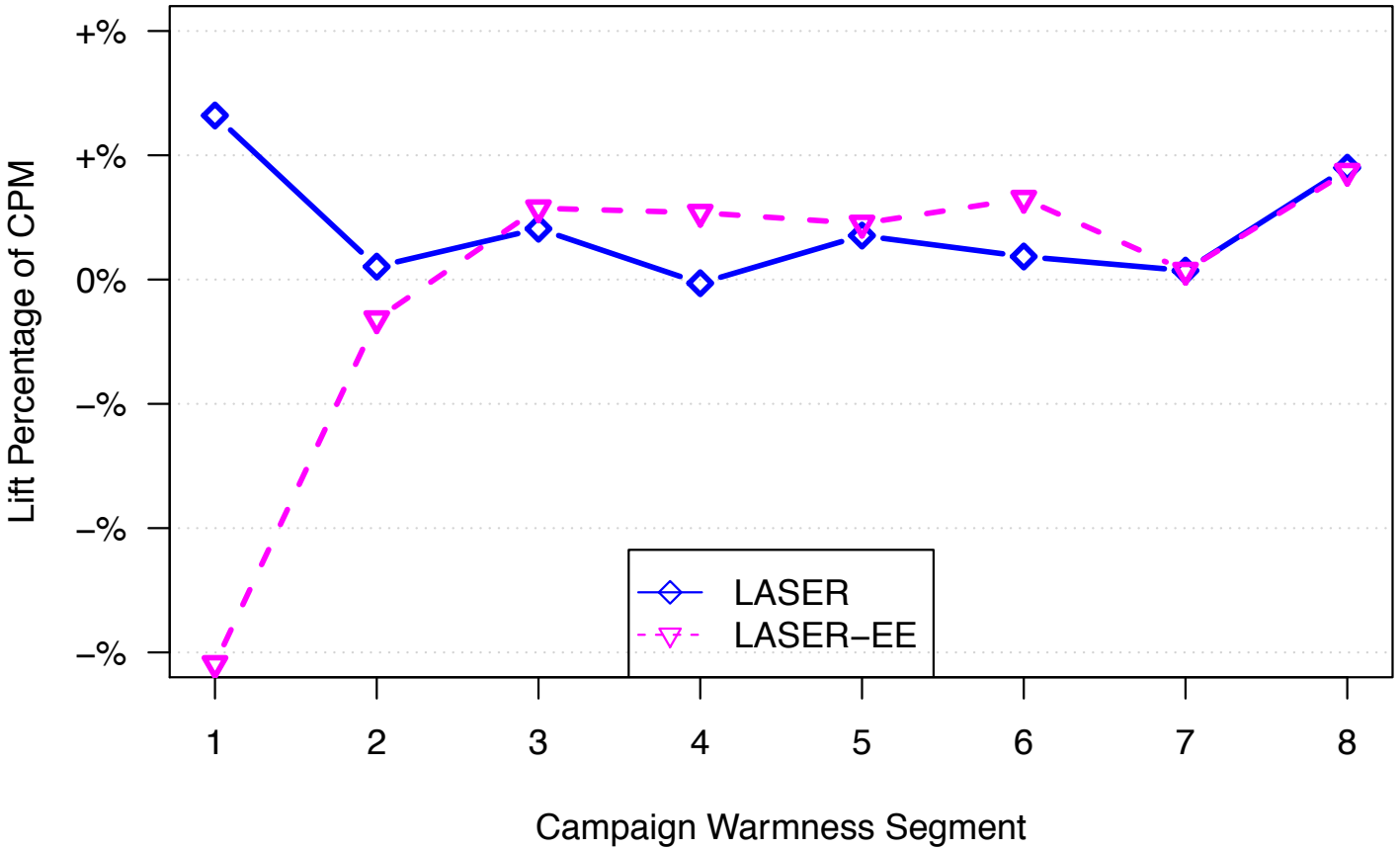
# Daily CPM Lift Over Control



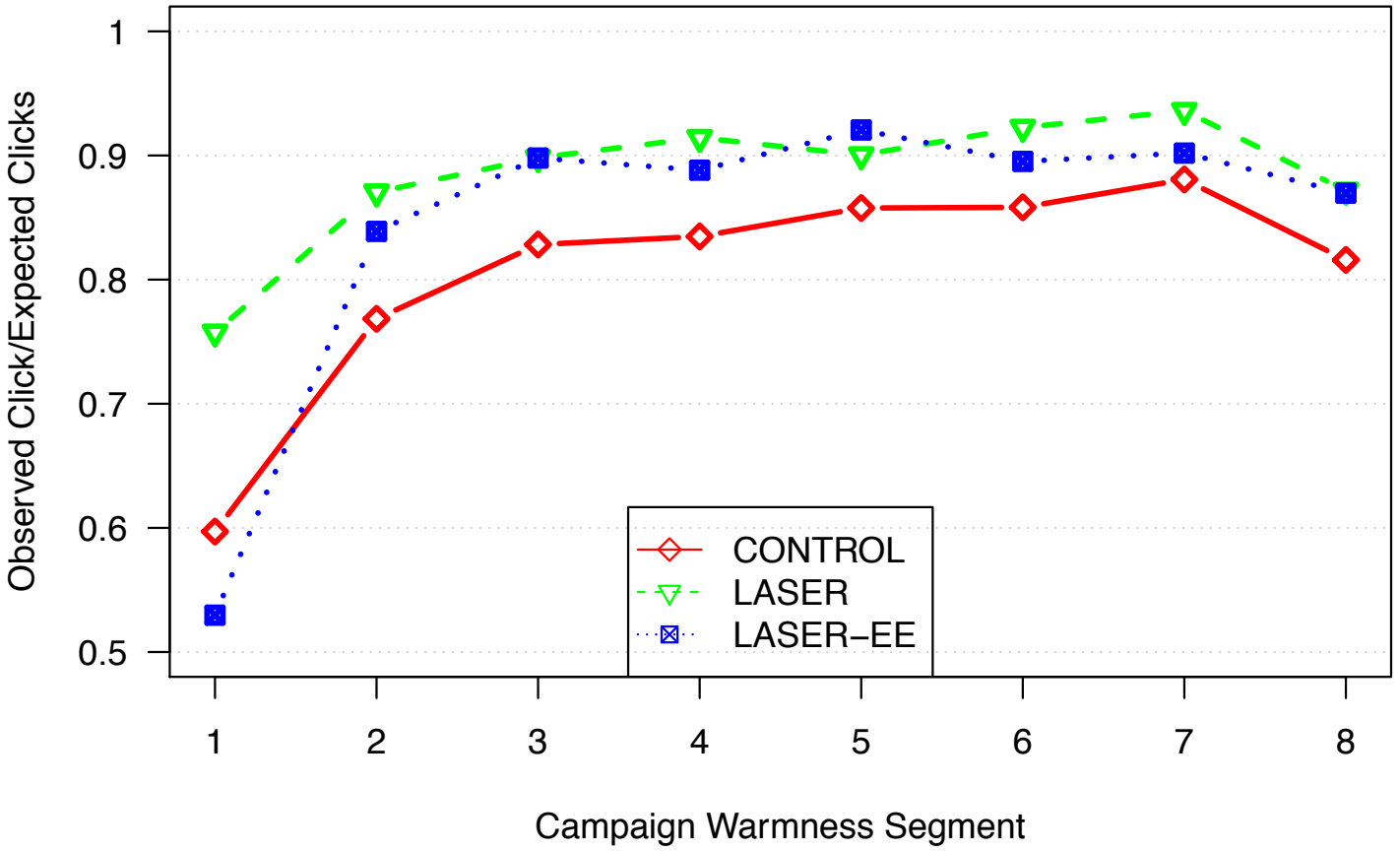
# CTR Lift By Campaign Warmness Segments



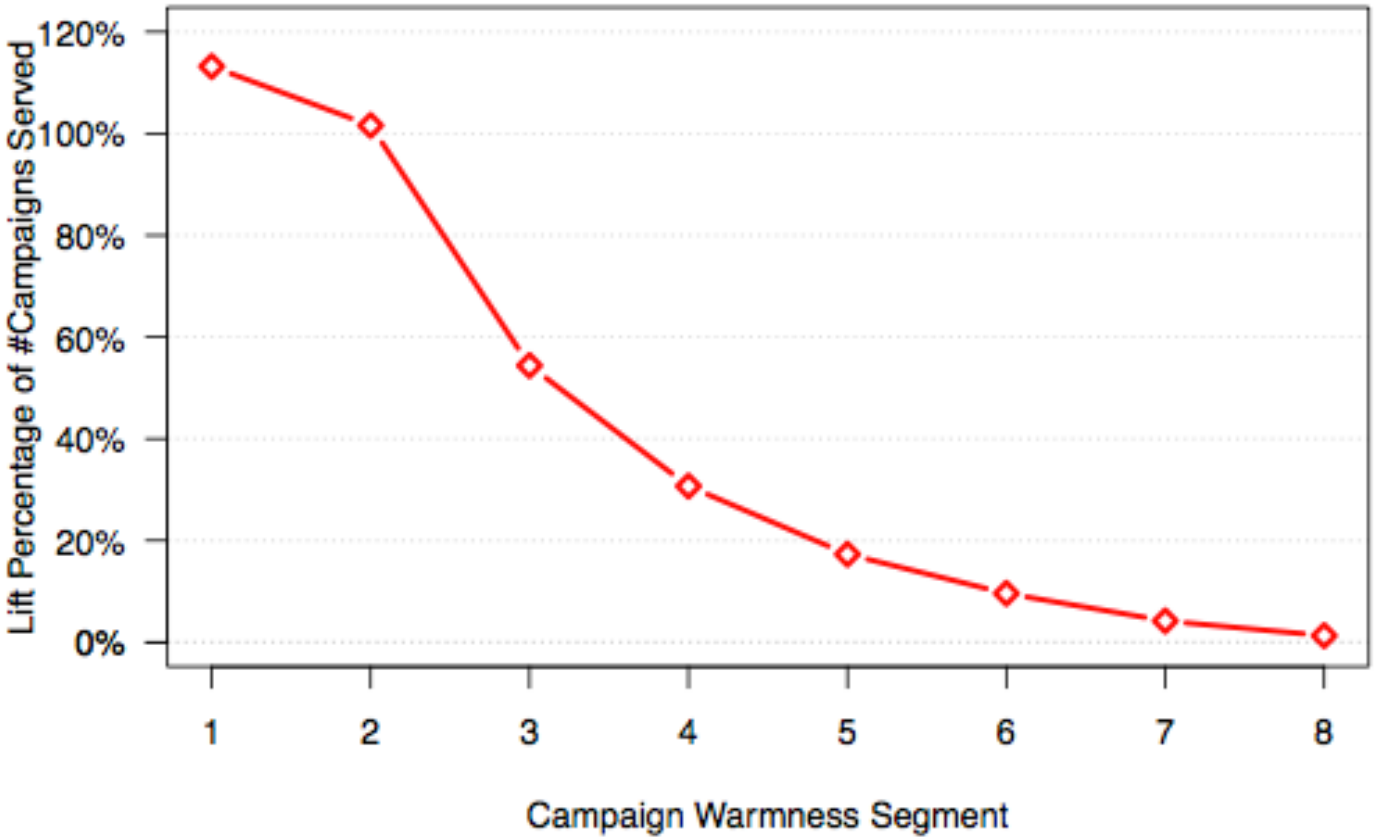
# CPM Lift By Campaign Warmness Segments



# O/E Ratio By Campaign Warmness Segments



# Number of Campaigns Served Improvement from E/E



# Insights

- Overall performance:
  - LASER and LASER-EE are both much better than control
  - LASER and LASER-EE performance are very similar
- Segmented analysis by campaign warmness
  - Segment #1 (very cold)
    - LASER-EE much worse than LASER due to its exploration property
    - LASER much better than CONTROL due to cold-start features
  - Segments #3 - #5
    - LASER-EE significantly better than LASER
    - Winner's curse hit LASER
  - Segment #6 - #8 (very warm)
    - LASER-EE and LASER are equivalent
- Number of campaigns served
  - LASER-EE serves significantly more campaigns than LASER
  - Provides healthier market place

# Parallel Matrix Factorization

# Recap: Properties of Web Recommender Systems

- One or multiple metrics to optimize
  - Click Through Rate (CTR) (focus of this talk)
  - Revenue per impression
  - Time spent on the landing page
  - Ad conversion rate
  - …
- Imbalanced data (clicks are very sparse)
- Large scale data
- Cold-start
  - User features: Age, gender, position, industry, …
  - Item features: Category, key words, creator features, …

# Matrix Factorization Models

- Response matrix Y
  - $y_{ij}$ = 1 if user i clicked item j
  - $y_{ij}$ = 0 if user i viewed item j but did not click
- Y is a highly incomplete matrix
- $Y \approx U'V$
  - $u_i'$ is the i-th row of U, latent profile of user i
  - $v_j'$ is the j-th row of V, latent profile of item j
- Widely used approach such as Netflix competition

# Probabilistic Matrix Factorization

- $$y_{ij} \sim Bernoulli(p_{ij}),$$

$$s_{ij} = \log \frac{p_{ij}}{1-p_{ij}}$$

$$s_{ij} = f(x_{ij}) + \alpha_i + \beta_j + \boldsymbol{u}_i' \boldsymbol{v}_j.$$

Global Features | User effect | Item effect | User factors | Item factors

Salakhutdinov and Mnih (2007)

# Flexible Regression Priors

- User covariates

$$\alpha_i \sim N(g(x_i), \sigma_\alpha^2), \quad \boldsymbol{u}_i \sim N(G(x_i), \sigma_u^2 I),$$

$$\beta_j \sim N(h(x_j), \sigma_\beta^2), \quad \boldsymbol{v}_j \sim N(H(x_j), \sigma_v^2 I),$$

Item covariates

- g(·), h(·), G(·), H(·) can be any regression functions, e.g. simple linear regression

- Agarwal and Chen (KDD 2009); Zhang et al. (RecSys 2011)

# Model Fitting Using MCEM (Single Machine)

- Monte Carlo EM (Booth and Hobert 1999)
- Let $\boldsymbol{\Theta} = (f, g, h, G, H, \sigma_\alpha^2, \sigma_u^2, \sigma_\beta^2, \sigma_v^2)$
  $\boldsymbol{\Delta} = \{\alpha_i, \beta_j, \boldsymbol{u}_i, \boldsymbol{v}_j\}_{\forall i,j}$
- Let $q_t(\boldsymbol{\Theta}) = E_{\boldsymbol{\Delta}}[\log L(\boldsymbol{\Theta}; \boldsymbol{\Delta}, \boldsymbol{y}) \mid \hat{\boldsymbol{\Theta}}^{(t)}]$
- E Step:
  - Obtain N samples of conditional posterior

$$p(\alpha_i| \sim), p(\beta_j| \sim), p(\boldsymbol{u}_i| \sim), p(\boldsymbol{v}_j| \sim)$$

- M Step: $\hat{\boldsymbol{\Theta}}^{(t+1)} = \arg\max_{\boldsymbol{\Theta}} q_t(\boldsymbol{\Theta})$.

# Handling Binary Responses

- Gaussian responses:

  $p(\alpha_i|\sim), p(\beta_j|\sim), p(\boldsymbol{u}_i|\sim), p(\boldsymbol{v}_j|\sim)$ have closed form

- Binary responses + Logistic: no longer closed form

- Variational approximation (VAR)

- Adaptive rejection sampling (ARS)

# Variational Approximation (VAR)

- Initially let $\xi_{ij} = 1$
- Before each E-step, create pseudo Gaussian response for each binary observation

$$r_{ij} = \frac{2y_{ij}-1}{4\lambda(\xi_{ij})} \text{ with variance } \sigma^2_{ij} = \frac{1}{2\lambda(\xi_{ij})},$$

$$\lambda(\xi) = \frac{1}{4\xi}\tanh\left(\frac{\xi}{2}\right)$$

- Run E-Step and M-Step using the Gaussian pseudo response
- After M-step, let $\xi_{ij} = \sqrt{E[s^2_{ij}]}$.

Agarwal and Chen (KDD 2009)

# Adaptive Rejection Sampling (ARS)

- For each E-step, obtain precise conditional posterior samples of

$$p(\alpha_i|\sim), p(\beta_j|\sim), p(\boldsymbol{u}_i|\sim), p(\boldsymbol{v}_j|\sim)$$

- Adaptively update the upper and lower bound of the density function to do rejection sampling

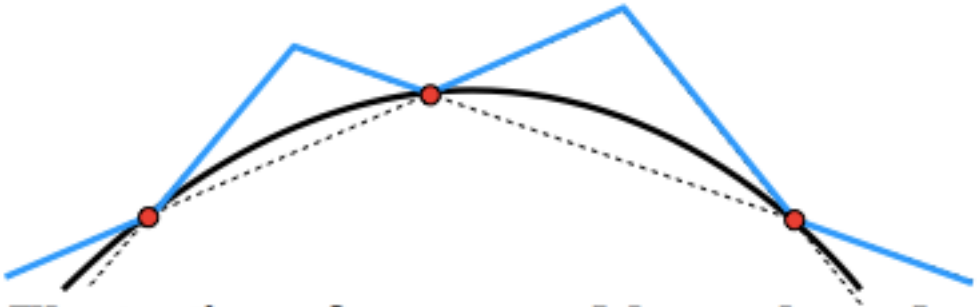


**Figure 1: Illustration of upper and lower bounds of an arbitrary (log) density function**

# Adaptive Rejection Sampling (ARS)

- Goal: Obtain a sample x* from a log-concave target density function p(x)
- Try to avoid evaluating p(x) as much as possible
- Initialize point set S with 3 initial points
- Construct lower bound *lower(x)* and upper bound *upper(x)* using the current S
- Let e(x) = exp(upper(x)), s(x) = exp(lower(x))
- Let e1(x) be the density function from e(x)
- Repeat the following steps until a sample is obtained:
  - Draw x* ~ e1(x) and z ~ Unif(0,1)
  - If z<=s(x*)/e(x*), accept x* and break
  - If z<=p(x*)/e(x*), accept x* and break; otherwise reject x*
  - If x* rejected, update e(x) and s(x) by constructing new lower and upper bounds

# Single machine model fitting is good, but…

- We are working with very large scale data sets
- Parallel matrix factorization methods using Map-Reduce have to be developed

# Model Fitting Using MCEM

- Monte Carlo EM (Booth and Hobert 1999)
- Let $\boldsymbol{\Theta} = (f, g, h, G, H, \sigma^2_\alpha, \sigma^2_u, \sigma^2_\beta, \sigma^2_v)$
  $\boldsymbol{\Delta} = \{\alpha_i, \beta_j, \boldsymbol{u}_i, \boldsymbol{v}_j\}_{\forall i,j}$
- Let $q_t(\boldsymbol{\Theta}) = E_{\boldsymbol{\Delta}}[\log L(\boldsymbol{\Theta}; \boldsymbol{\Delta}, \boldsymbol{y}) \mid \hat{\boldsymbol{\Theta}}^{(t)}]$
- E Step:
  - Obtain N samples of conditional posterior

$$p(\alpha_i| \sim), p(\beta_j| \sim), p(\boldsymbol{u}_i| \sim), p(\boldsymbol{v}_j| \sim)$$

- M Step: $\hat{\boldsymbol{\Theta}}^{(t+1)} = \arg\max_{\boldsymbol{\Theta}} q_t(\boldsymbol{\Theta}).$

# Parallel Matrix Factorization

- Partition data into m partitions
- For each partition $\ell \in \{1, \ldots, m\}$ run MCEM algorithm and get $\hat{\Theta}_\ell$.
- $$\text{Let } \hat{\Theta} = \frac{1}{m} \sum_{\ell=1}^{m} \hat{\Theta}_\ell.$$
- Ensemble runs: for k = 1, … , n
  - Repartition data into m partitions with a new seed
  - Run E-step only job for each partition given $\hat{\Theta}$
- Average over user/item factors for all partitions and k's to obtain the final estimate

# Parallel Matrix Factorization

One Map-Reduce job

- Partition data into m partitions
- For each partition $\ell \in \{1, \ldots, m\}$ run MCEM algorithm and get $\hat{\Theta}_\ell$.
- Let $\hat{\Theta} = \frac{1}{m} \sum_{\ell=1}^{m} \hat{\Theta}_\ell$.

Each ensemble run is a Map-Reduce job

- Ensemble runs: for k = 1, … , n
  - Repartition data into m partitions with a new seed
  - Run E-step only job for each partition given $\hat{\Theta}$
- Average over user/item factors for all partitions and k's to obtain the final estimate

# Key Points

- Partitioning is tricky!
  - By events? By items? By users?

- Empirically, "divide and conquer" + average over $\hat{\Theta}_\ell$ to obtain $\hat{\Theta}$ work well!

- Ensemble runs: After obtained $\hat{\Theta}$, we run n E-step-only jobs and take average, for each job using a different user-item mix.

# Identifiability Issues (MCEM-ARSID)

- Same log-likelihood can be achieved by
  - g ( ) = g ( ) + r, h ( ) = h ( ) – r
    - Center **α, β, u** to zero-mean every E-step
  - **u** = -**u**, **v** = -**v**
    - Constrain v to be positive
  - Switching $u._1$, $v._1$ with $u._2$, $v._2$
    - $\mathbf{u}_i \sim N(G(x_i), \mathbf{I})$, $\mathbf{v}_j \sim N(H(x_j), \boldsymbol{\lambda}\mathbf{I})$
    - Constraint: Diagonal entries $\lambda_1 >= \lambda_2 >= \ldots$

# MovieLens 1M Data

- 75% training and 25% test split by time
- Imbalanced data
  - User rating = 1: Positive
  - User rating = 2, 3, 4, 5: Negative
  - 5% positive rate
- Balanced data
  - User rating = 1, 2, 3: Positive
  - User rating = 4, 5: Negative
  - 44% positive rate

**Table 1: AUC of different methods on the imbalanced and balanced MovieLens datasets (#partitions= 1 indicates single-machine runs)**

| Method | # Partitions | AUC | |
|---|---|---|---|
| | | Imbalanced | Balanced |
| SGD | 1 | 0.8090 | 0.7413 |
| MCEM-VAR | 1 | 0.8138 | 0.7576 |
| MCEM-ARS | 1 | 0.8195 | 0.7563 |
| MCEM-VAR | 2 | 0.7614 | 0.7599 |
| | 5 | 0.7191 | 0.7538 |
| | 15 | 0.6584 | 0.7421 |
| MCEM-ARS | 2 | 0.8194 | 0.7622 |
| | 5 | 0.7971 | 0.7597 |
| | 15 | 0.7775 | 0.7493 |

- MCEM-VAR and MCEM-ARS slightly better than SGD for 1 partition
- A lot of effort spent on tuning $L_2$ penalty parameters and learning rate for SGD

**Table 1: AUC of different methods on the imbalanced and balanced MovieLens datasets (#partitions= 1 indicates single-machine runs)**

| Method | # Partitions | AUC Imbalanced | AUC Balanced |
|---|---|---|---|
| SGD | 1 | 0.8090 | 0.7413 |
| MCEM-VAR | 1 | 0.8138 | 0.7576 |
| MCEM-ARS | 1 | 0.8195 | 0.7563 |
| MCEM-VAR | 2 | 0.7614 | 0.7599 |
| | 5 | 0.7191 | 0.7538 |
| | 15 | 0.6584 | 0.7421 |
| MCEM-ARS | 2 | 0.8194 | 0.7622 |
| | 5 | 0.7971 | 0.7597 |
| | 15 | 0.7775 | 0.7493 |

Big difference between VAR and ARS for imbalanced data!

# Yahoo! FrontPage Today Module Recommendation

# How to Handle New Articles

- New articles can get generated any time during the day
- Problem: How to quickly build models to reflect user preference for the fresh articles
- Offline large-scale matrix factorization can not be real-time!

# Online Logistic Regression (OLR)

- User i with feature $\mathbf{x}_i$, article j
- Binary response y (click/non-click)
- $y_{ij} = Bernoulli(p_{ij})$
- $s_{ij} = \log \frac{p_{ij}}{1 - p_{ij}} = \boldsymbol{x}_i' \boldsymbol{\beta}_j$
- Prior $\boldsymbol{\beta}_j \sim N(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$
- Using Laplace approximation or variational Bayesian methods to obtain posterior

$$\boldsymbol{\beta}_j | y_{ij} \sim N(\hat{\boldsymbol{\mu}}_j, \hat{\boldsymbol{\Sigma}}_j)$$

- Prior for next batch $\boldsymbol{\beta}_j \sim N(\hat{\boldsymbol{\mu}}_j, \hat{\boldsymbol{\Sigma}}_j)$
- Can approximate $\boldsymbol{\Sigma}_j$ and $\hat{\boldsymbol{\Sigma}}_j$ as diagonal for high dim $\mathbf{x}_i$

# User Features for OLR

- Age, gender, job position etc for login users
- General behavior targeting (BT) features
  - Music? Finance? Politics?
- User profiles from historical view/click behavior on previous items in the data, e.g.
  - Item-profile: use previously clicked item ids as the user profile
  - Category-profile: use item category affinity score as profile. The score can be simply user's historical CTR on each category.
  - Are there better ways to generate user profiles?
  - Yes! By matrix factorization!

# Matrix Factorization For User Profile in OLR

- Offline user profile building period, obtain the user factor $\boldsymbol{u}_i$ for user i

- Online modeling using OLR
  - If a user has a profile (warm-start), use $\boldsymbol{u}_i$ as the user covariates
  - If not (cold-start), use $G(x_i)$ as the user covariates

# Large Yahoo! Front Page Data

- Data for building user profile: 8M users with at least 10 clicks (heavy users) in June 2011, 1B events
- Data for training and testing OLR model: Random served data with 2.4M clicks in July 2011
- Heavy users contributed around 30% of clicks
- User covariates / features for OLR:
  - Intercept-only (MOST POPULAR)
  - 124 Behavior targeting features (BT-ONLY)
  - BT + top 1000 clicked article ids (ITEM-PROFILE)
  - BT + user profile with CTR on 43 binary content categories (CATEGORY-PROFILE)
  - BT + user latent factor from matrix factorization

# Offline Evaluation Metric Related to Clicks

- For model M and J live items (articles) at any time

$$S(M) = J \sum_{visits\ with\ click} 1(\text{item clicked} = \text{item selected by M}).$$

- If M = random (constant) model

  E[S(M)] = #clicks

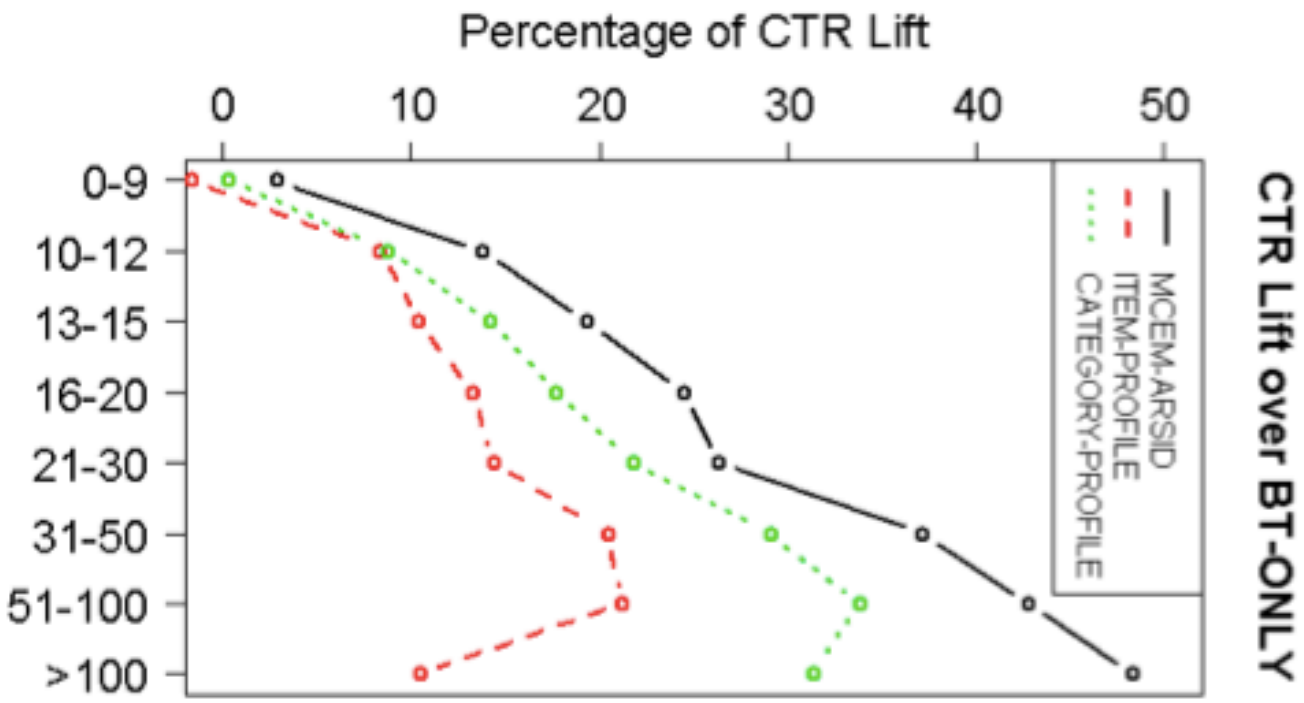
- Unbiased estimate of expected total clicks (Langford et al. 2008)

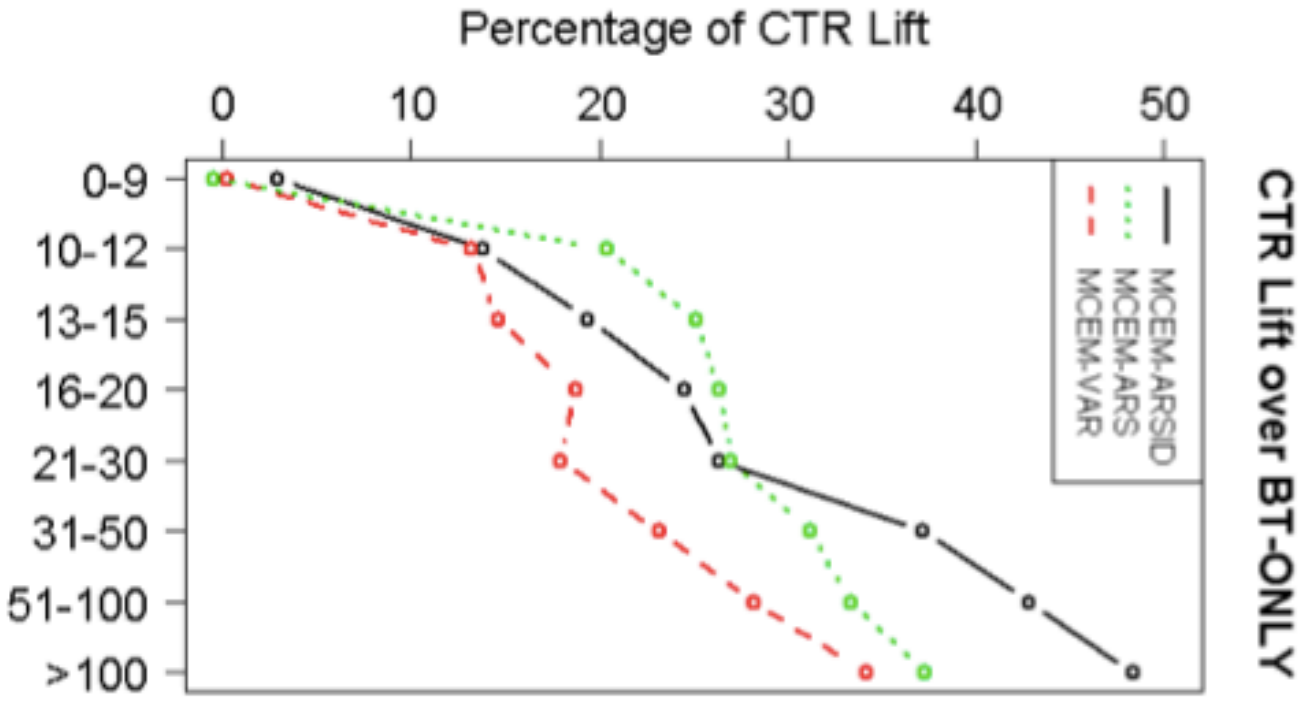
# Click Lift Performance For Different User Profiles

**Table 3: The overall click lift over the user behavior feature (BT) only model.**

Warm Start: Users with at least one sample in training data
Cold Start: Users with no data in training data

| Method | #Ensembled Runs | Overall | Warm Start | Cold Start |
|---|---|---|---|---|
| ITEM-PROFILE | – | 3.0% | 14.1% | -1.6% |
| CATEGORY-PROFILE | – | 6.0% | 20.0% | 0.3% |
| MCEM-VAR | 10 | 5.6% | 18.7% | 0.2% |
| MCEM-ARS | 10 | 7.4% | 26.8% | -0.5% |
| MCEM-ARSID | 1 | 9.1% | 24.6% | 2.8% |
| MCEM-ARSID | 10 | 9.7% | 26.3% | 2.9% |


Percentage of CTR Lift
0
10
20
30
40
50
0-9
10-12
13-15
16-20
21-30
31-50
51-100
>100
MCEM-ARSID
ITEM-PROFILE
CATEGORY-PROFILE
CTR Lift over BT-ONLY


Percentage of CTR Lift
0
10
20
30
40
50
0-9
10-12
13-15
16-20
21-30
31-50
51-100
>100
MCEM-ARSID
MCEM-ARS
MCEM-VAR
CTR Lift over BT-ONLY

# Conclusion

- ARS to probabilistic matrix factorization framework for imbalanced binary data
- Model fitting approach for large scale data using Map-Reduce
- Identifiability the key for parallel matrix factorization (cold-start)
- Our proposed approach significantly outperforms baseline methods such as variational approximation

# Bag of Little Bootstraps

Kleiner et al. 2012

# Motivation: Challenges from Data Analysis

- Example: How to generate confidence intervals of model performance like AUC?
- Single machine:
  - Bootstrap the test data 100 times
  - Compute the std(AUC) from AUC in the 100 test data
- How do we do it for large scale data?
  - Bootstrap non-trivial!

# Bootstrap (Efron, 1979)

- A re-sampling based method to obtain statistical distribution of sample estimators
- Why are we interested ?
  - Re-sampling is embarrassingly parallelizable
- For example:
  - Standard deviation of the mean of N samples (μ)
  - For i = 1 to r do
    - Randomly sample with replacement N times from the original sample -> bootstrap data i
    - Compute mean of the i-th bootstrap data -> $\mu_i$
  - Estimate of Sd(μ) = Sd([$\mu_1$,…$\mu_r$])
  - r is usually a large number, e.g. 200

# Bootstrap for Big Data

- Can have r nodes running in parallel, each sampling one bootstrap data
- However…
  - N can be very large
  - Data may not fit into memory
  - Collecting N samples with replacement on each node can be computationally expensive

# M out of N Bootstrap (Bikel et al. 1997)

- Obtain $Sd_M(\mu)$ by sampling M samples with replacement for each bootstrap, where M<N
- Apply analytical correction to $Sd_M(\mu)$ to obtain $Sd(\mu)$ using prior knowledge of convergence rate of sample estimates
- However…
  - Prior knowledge is required
  - Choice of M is critical to performance
  - Finding optimal value of M needs more computation

# Bag of Little Bootstraps (BLB)

- Example: Standard deviation of the mean
- Generate S sampled data sets, each obtained by random sampling without replacement a subset of size b (or partition the original data into S partitions, each with size b)
- For each data p = 1 to S do
  - For i = 1 to r do
    - N samples with replacement on data of size b
    - Compute mean of the resampled data $\mu_{pi}$
  - Compute $Sd_p(\mu) = Sd([\mu_{p1},\ldots\mu_{pr}])$
- Estimate of $Sd(\mu) = Avg([Sd_1(\mu),\ldots, Sd_S(\mu)])$

# Bag of Little Bootstraps (BLB)

- Interest: $\xi(\theta)$, where $\theta$ is an estimate obtained from size N data
  - $\xi$ is some function of $\theta$, such as standard deviation, …
- Generate S sampled data sets, each obtained from random sampling without replacement a subset of size b (or partition the original data into S partitions, each with size b)
- For each data p = 1 to S do
  - For i = 1 to r do
    - Sample N samples with replacement on data of size b
    - Compute the estimate of the resampled data $\theta_{pi}$
  - Compute $\xi_p(\theta) = \xi([\theta_{p1},\ldots\theta_{pr}])$
- Estimate of $\xi(\mu) = \text{Avg}([\xi_1(\theta),\ldots, \xi_S(\theta)])$

# Bag of Little Bootstraps (BLB)

- Interest: $\xi(\theta)$, where $\theta$ is an estimate obtained from size N data
  - $\xi$ is some function of $\theta$, such as standard deviation, …
- Generate S sampled data sets, each obtained from random sampling without replacement a subset of size b (or partition the original data into S partitions, each with size b)
- For each data p = 1 to S do
  - For i = 1 to r do
    - Sample N samples with replacement on the data of size b
    - Compute the estimate of the resampled data $\theta_{pi}$
  - Compute $\xi_p(\theta) = \xi([\theta_{p1},\ldots\theta_{pr}])$
- Estimate of $\xi(\mu) = Avg([\xi_1(\theta),\ldots, \xi_S(\theta)])$

Mapper

Reducer

Gateway

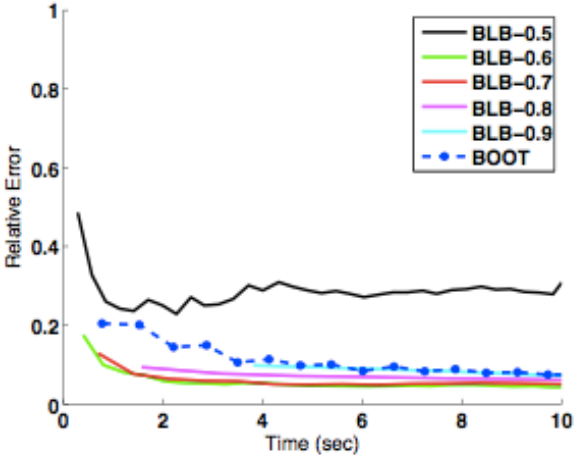
# Why is BLB Efficient

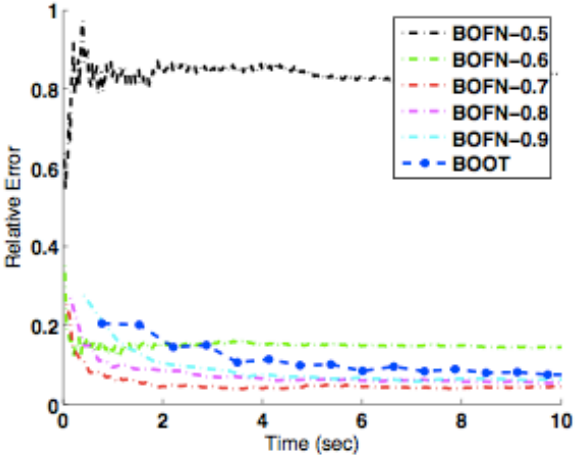
- Before:
  - N samples with replacement from size N data is expensive when N is large
- Now:
  - N samples with replacement from size b data
  - b can be several magnitude smaller than N (e.g. $b = N^{\gamma}$, $\gamma$ in [0.5, 1))
  - Equivalent to: A multinomial sampler with dim = b
  - Storage = O(b), Computational complexity = O(b)

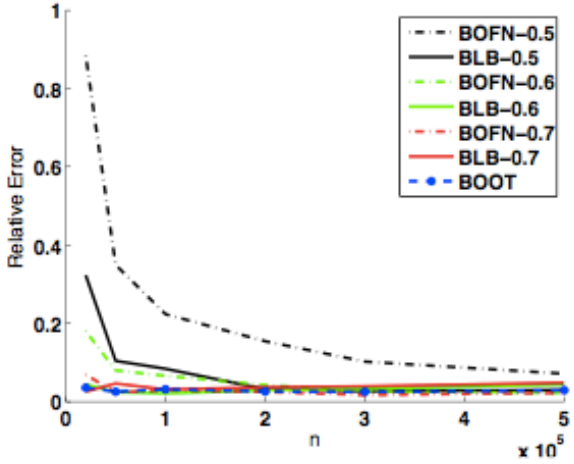
# Simulation Experiment

- 95% CI of Logistic Regression Coefficients
- N = 20000, 10 explanatory variables
- Relative Error = |Estimated CI width – True CI width | / True CI width
- BLB-γ: BLB with $b = N^{\gamma}$
- BOFN-γ: b out of N sampling with $b = N^{\gamma}$
- BOOT: Naïve bootstrap

# Simulation Experiment



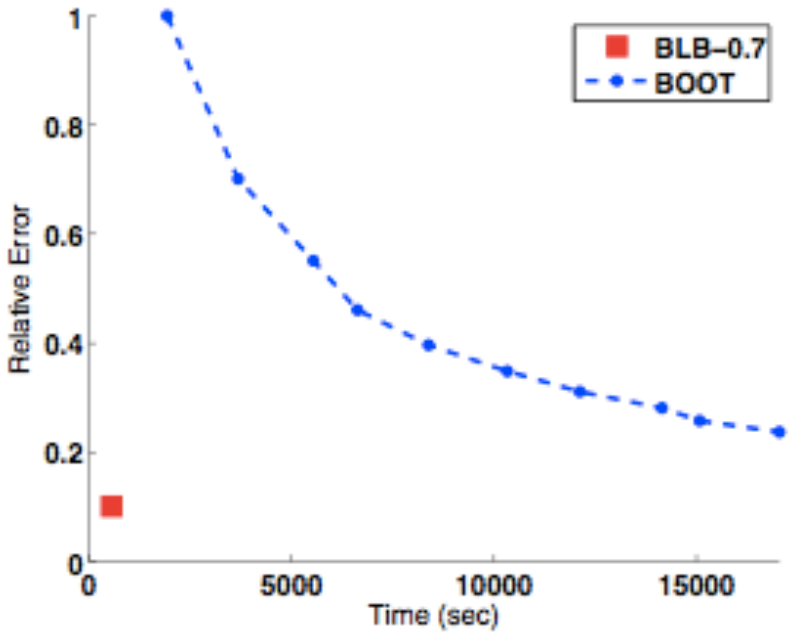



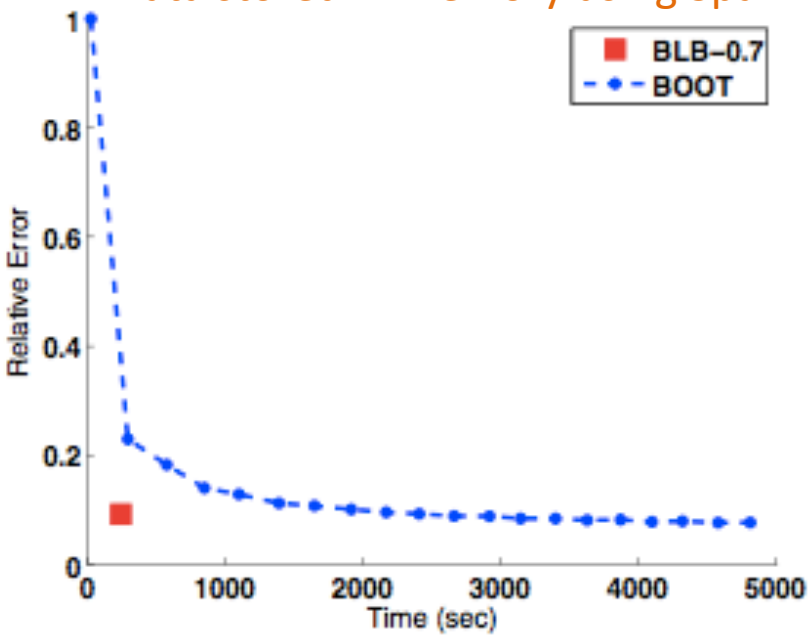
# Real Data

- 95% CI of Logistic Regression Coefficients
- N = 6M, 3000 explanatory variables
- Data size = 150GB, r = 50, S = 5, γ = 0.7

Data stored on disk




Data stored in memory using Spark



# Hyper-parameter Selection

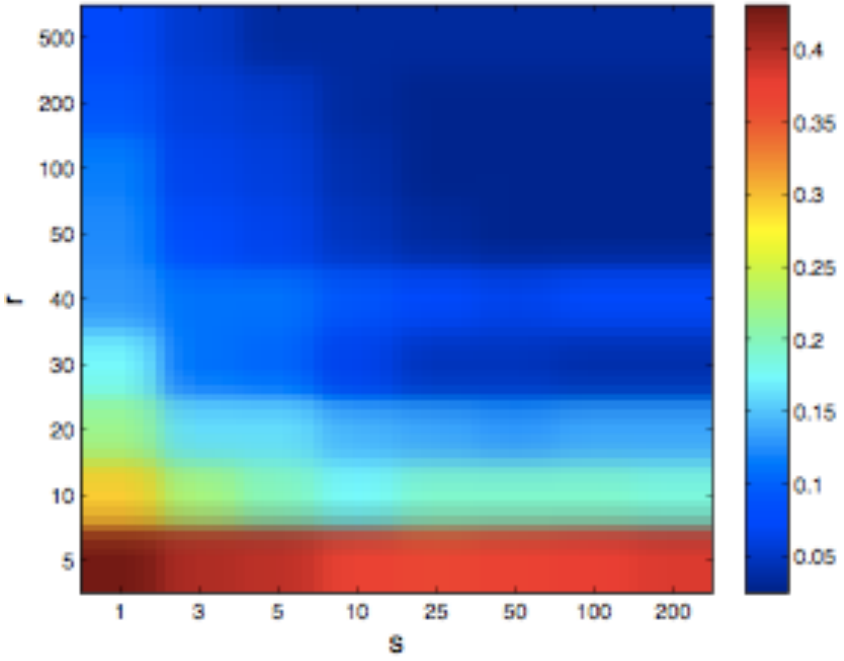
- From empirical experiments in the paper
  - S>=3 and r>=50 is sufficient for low relative errors

- Adaptively selecting r and S
  - Increase r and s until estimated value converges
  - For r: Continue to process resamples and update $\xi_p(\theta)$ until it has ceased to change significantly.
  - For S: Continue to process more subsamples (i.e., increasing S) until BLB's output value has stabilized.

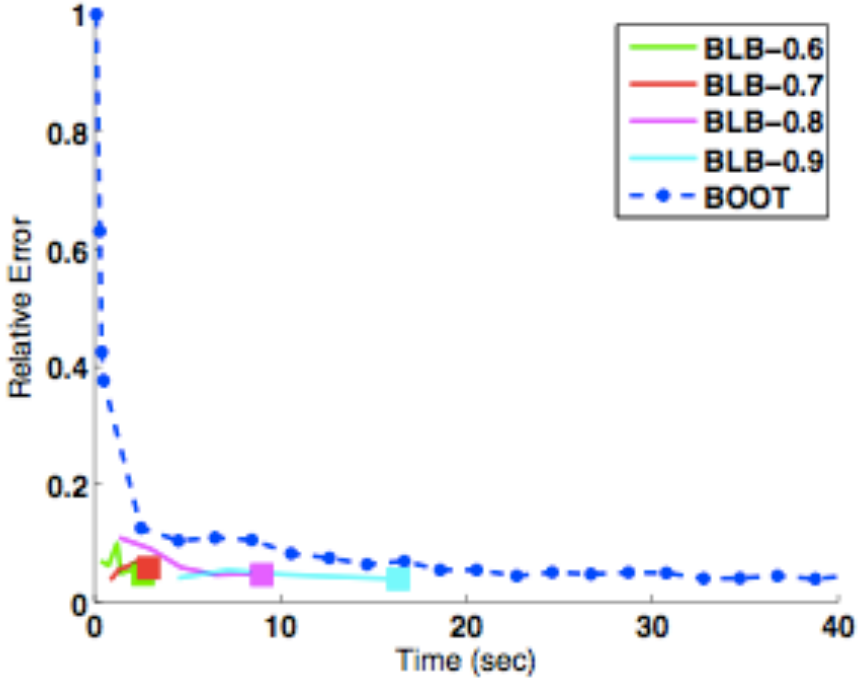
# Hyper-parameter Selection

- Adaptively selecting r and S
  - Increase r and S until estimated value converges

BLB-0.7




Adaptive BLB



# Summary of BLB

- A new algorithm for bootstrapping on big data

- Advantages
  - Fast and efficient
  - Easy to parallelize
  - Easy to understand and implement
  - Friendly to Hadoop, makes it routine to perform statistical calculations on Big data

# Part V: Future of Distributed Computing Infrastructure

# Problem of ADMM in Hadoop

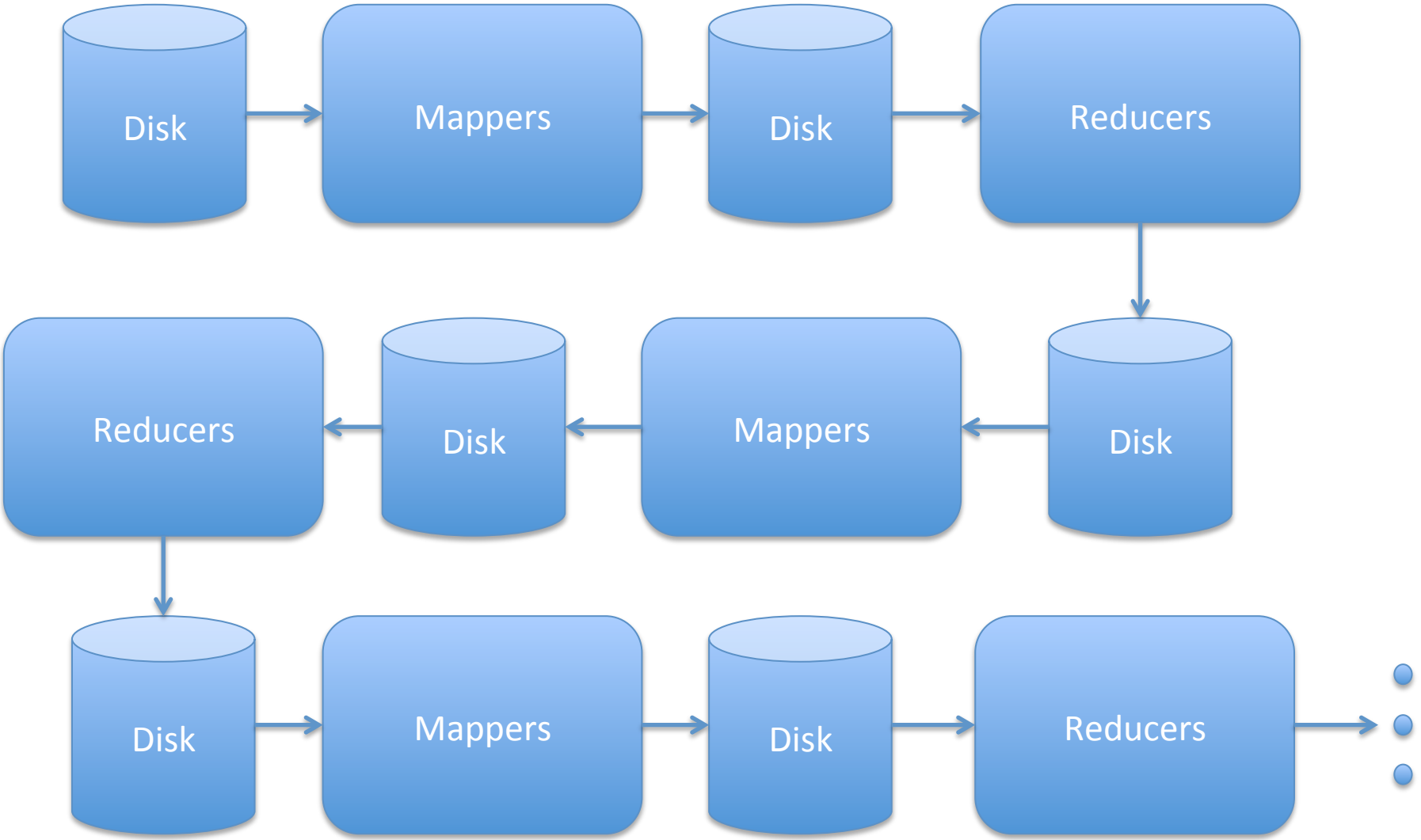
- ADMM in hadoop can take hours to converge
- Is there a better way to handle iterative learning process?

# Spark overview

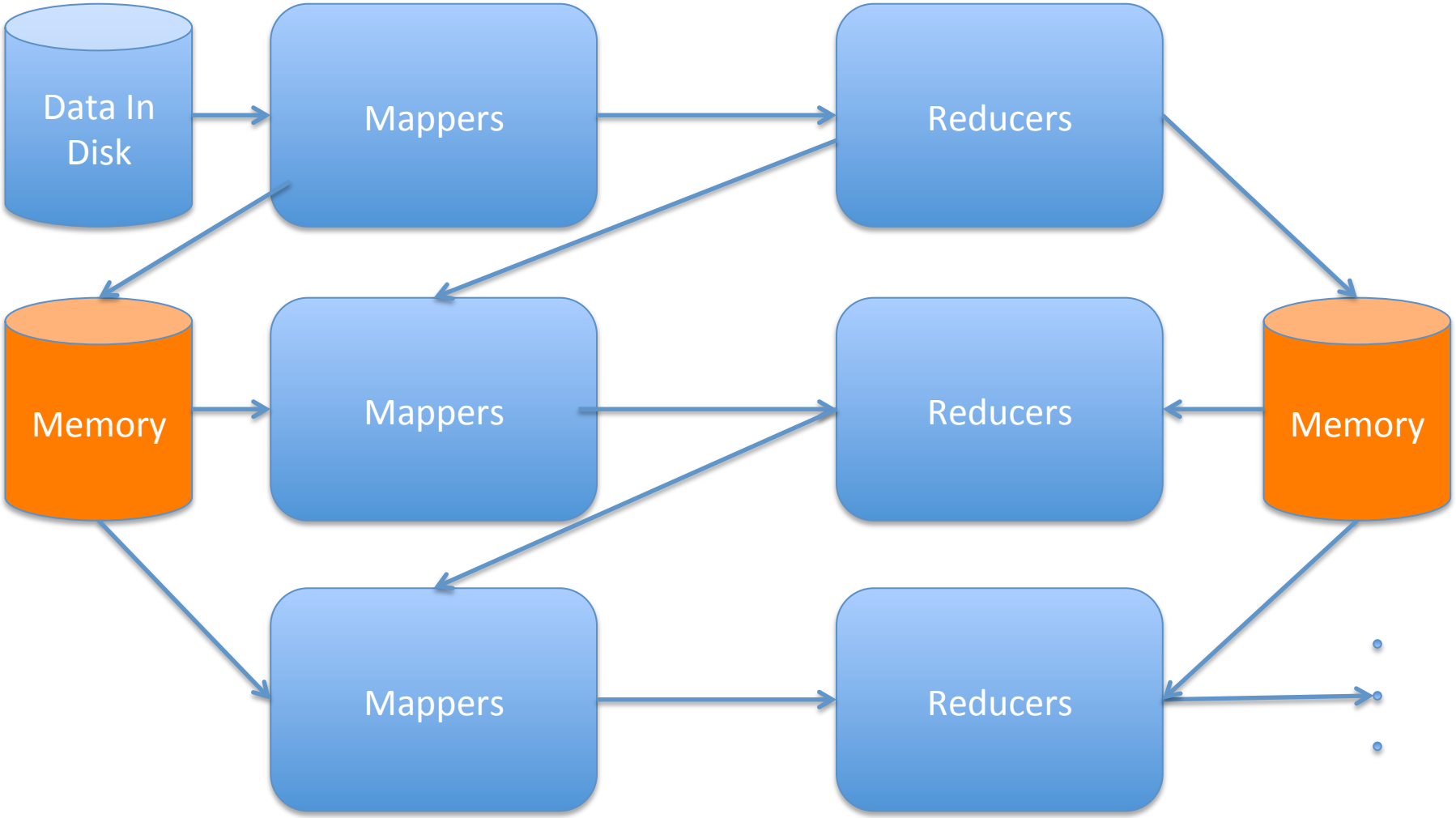

- What is Spark
  - A fast cluster computing system compatible with Hadoop
    - Works with HDFS
  - Improves efficiency through *in-memory* computing primitives
  - Written in Scala, provides APIs in Scala, Java and Python
    - Often 2-10× less code with Scala


- What can it do
  - Developed for *iterative* algorithms
    - Load data into memory and query it repeatedly more quickly than with disk-based systems like Hadoop
  - Retain the attractive properties of MapReduce
    - Fault tolerance, data locality, scalability

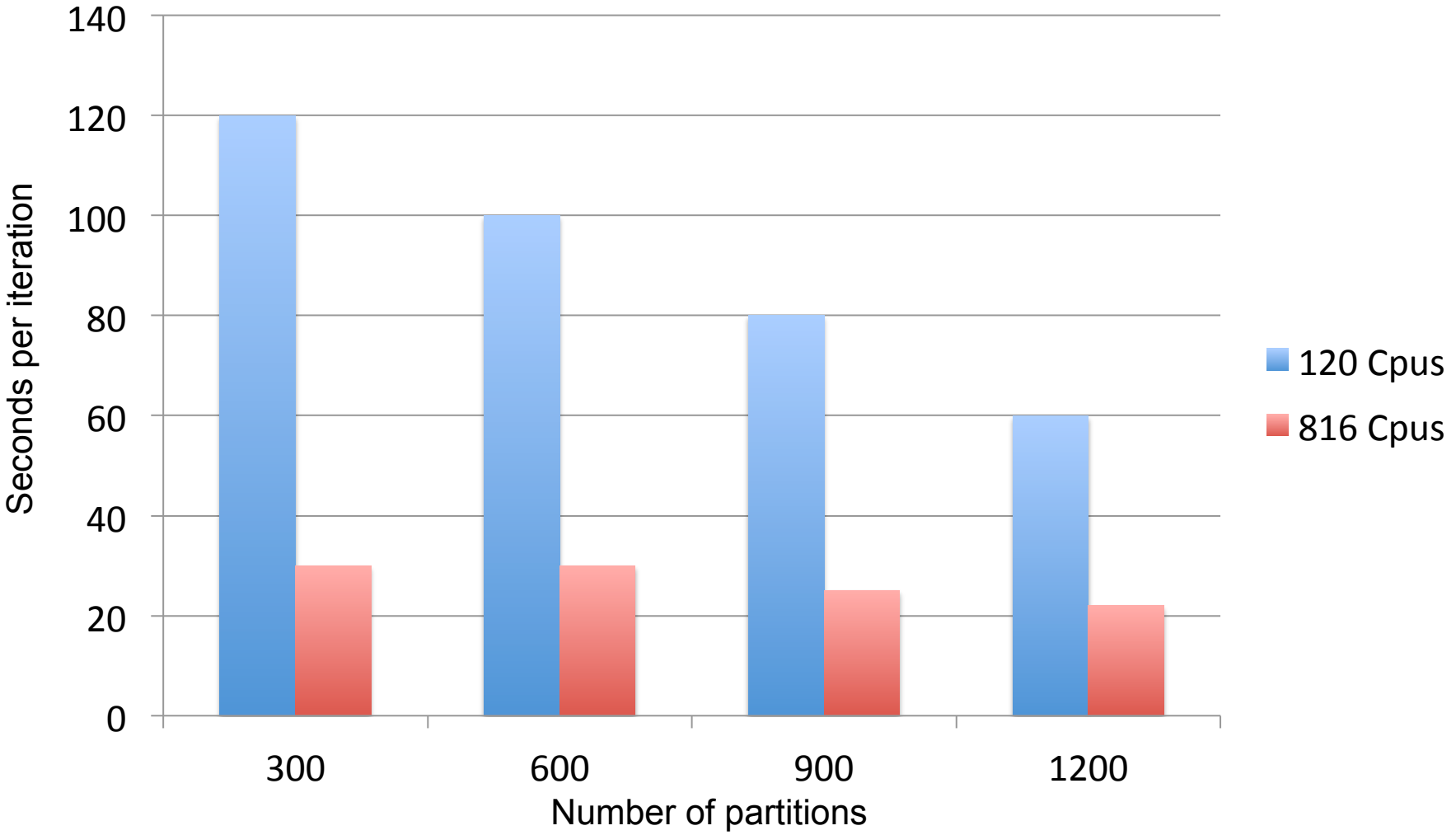
# Iterative Process in Hadoop



# Iterative Process in Spark



# Spark cluster speed-up *preliminary* results



# Initial Spark Result Summary

- ✓ Spark is very efficient for large-scale machine learning purposes
  - ADMM on Ads data, 20 mins (Spark) VS 4-5 hours (Hadoop)
  - Provides an offline experimental system to vet various modeling ideas with high agility
  - Models get trained faster thus can react to temporal shifts of online system more quickly

# GraphLab

- An open-source graph-based, high performance, distributed computation framework in C++
- http://graphlab.org/
- HDFS integration
- Major design
  - Sparse data with local dependencies
  - Iterative algorithms
  - Potentially asynchronous execution among nodes

# GraphLab

- Graph-parallel
- Map-Reduce: computation applied to independent records
- GraphLab: dependent records stored as vertices in a large distributed data-graph
- Computation in parallel on each vertex and can interact with neighboring vertices

# GraphX

- Combines the advantages of both data-parallel and graph-parallel systems

- Distributed graph computation on Spark

# Final Remarks

# Summary

- Large scale machine learning is still a new area, a lot of open problems
- The advancing distributed computing infrastructure plays a big role in the algorithms
- In today's tutorial, we covered
  - Distributed computing and Map-Reduce
  - Web recommender systems: our motivation problem
  - Large scale logistic regression
  - Parallel matrix factorization
  - Bag of little bootstraps: One analysis tool for big data
  - Future of distributed computing infrastructure

# Bibliography


Agarwal, D. and Chen, B. (2009). Regression-based latent factor models. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, 19–28. ACM.

Agarwal, D., Chen, B., and Elango, P. (2010). Fast online learning through offline initialization for time-sensitive recommendation. In Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, 703–712. ACM.

Bell, R., Koren, Y., and Volinsky, C. (2007). Modeling relationships at multiple scales to improve accuracy of large recommender systems. In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, 95–104. ACM.

Booth, J. G., & Hobert, J. P. (1999). Maximizing generalized linear mixed model likelihoods with an automated Monte Carlo EM algorithm. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 61(1), 265-285.

Boyd, S., Parikh, N., Chu, E., Peleato, B., & Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine Learning, 3(1), 1-122.

Bickel, P. J., Götze, F., & van Zwet, W. R. (2012). Resampling fewer than n observations: gains, losses, and remedies for losses (pp. 267-297). Springer New York.

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.

# Bibliography


Efron, B. (1979). Bootstrap methods: another look at the jackknife. The annals of Statistics, 1-26.

Kleiner, A., Talwalkar, A., Sarkar, P., & Jordan, M. (2012). The big data bootstrap. arXiv preprint arXiv:1206.6415.

Khanna, R., Zhang, L., Agarwal, D. and Chen, B. (2012). Parallel Matrix Factorization for Binary Response. IEEE BigData 2013.

Mnih, Andriy, and Ruslan Salakhutdinov. (2007). "Probabilistic matrix factorization." *Advances in neural information processing systems*.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294.

Zhang, L., Agarwal, D., and Chen, B. (2011). Generalizing matrix factorization through flexible regression priors. In Proceedings of the fifth ACM conference on Recommender systems, 13–20. ACM.