

# 大数据机器学习

李武军

LAMDA Group  
南京大学计算机科学与技术系  
软件新技术国家重点实验室

liwujun@nju.edu.cn

Dec 23, 2014

# Outline

- ① Introduction
- ② Learning to Hash
  - Isotropic Hashing
  - Supervised Hashing with Latent Factor Models
  - Supervised Multimodal Hashing with SCM
  - Multiple-Bit Quantization
- ③ Distributed Learning
  - Coupled Group Lasso for Web-Scale CTR Prediction
  - Distributed Power-Law Graph Computing
- ④ Stochastic Learning
  - Distributed Stochastic ADMM for Matrix Factorization
- ⑤ Conclusion

# Outline

## 1 Introduction

## 2 Learning to Hash

- Isotropic Hashing
- Supervised Hashing with Latent Factor Models
- Supervised Multimodal Hashing with SCM
- Multiple-Bit Quantization

## 3 Distributed Learning

- Coupled Group Lasso for Web-Scale CTR Prediction
- Distributed Power-Law Graph Computing

## 4 Stochastic Learning

- Distributed Stochastic ADMM for Matrix Factorization

## 5 Conclusion

# Big Data

Big data has attracted much attention from both academia and industry.

- Facebook: 750 million users
- Flickr: 6 billion photos
- Wal-Mart: 267 million items/day; 4PB data warehouse
- Sloan Digital Sky Survey: New Mexico telescope captures 200 GB image data/day



# Definition of Big Data

- Gartner (2012): “Big data is high **volume**, high **velocity**, and/or high **variety** information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization.” (“**3Vs**”)
- International Data Corporation (IDC) (2011): “Big data technologies describe a new generation of technologies and architectures, designed to economically extract **value** from very large **volumes** of a wide **variety** of data, by enabling high-**velocity** capture, discovery, and/or analysis.” (“**4Vs**”)
- McKinsey Global Institute (MGI) (2011): “Big data refers to datasets whose size is **beyond the ability** of typical database software tools to capture, store, manage, and analyze.”

Why not hot until recent years?

- Big data: 金矿
- Cloud computing: 采矿技术
- Big data machine learning: 冶金技术

# Big Data Machine Learning

- **Definition:** perform machine learning from big data.
  
- **Role:** key for big data
  - **Ultimate goal** of big data processing is to mine **value** from data.
  - **Machine learning** provides **fundamental theory** and **computational techniques** for big data mining and analysis.

# Challenge

- Storage: memory and disk
- Computation: CPU
- Communication: network

# Our Contribution

- Learning to hash (哈希学习): memory/disk/cpu/communication
- Distributed learning (分布式学习): memory/disk/cpu;  
but increase communication cost
- Stochastic learning (随机学习): memory/disk/cpu

# Outline

## 1 Introduction

## 2 Learning to Hash

- Isotropic Hashing
- Supervised Hashing with Latent Factor Models
- Supervised Multimodal Hashing with SCM
- Multiple-Bit Quantization

## 3 Distributed Learning

- Coupled Group Lasso for Web-Scale CTR Prediction
- Distributed Power-Law Graph Computing

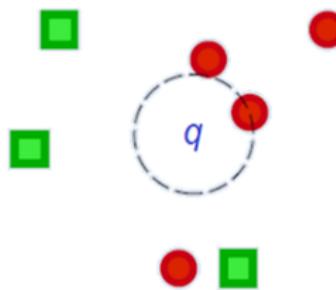
## 4 Stochastic Learning

- Distributed Stochastic ADMM for Matrix Factorization

## 5 Conclusion

# Nearest Neighbor Search (Retrieval)

- Given a query point  $q$ , return the points closest (similar) to  $q$  in the database(e.g. images).
- Underlying many machine learning, data mining, information retrieval problems



## Challenge in Big Data Applications:

- Curse of dimensionality
- Storage cost
- Query speed

# Similarity Preserving Hashing



$$h(\text{Statue of Liberty}) = \\ 10001010$$

$$h(\text{Napoléon}) = \\ 01100001$$

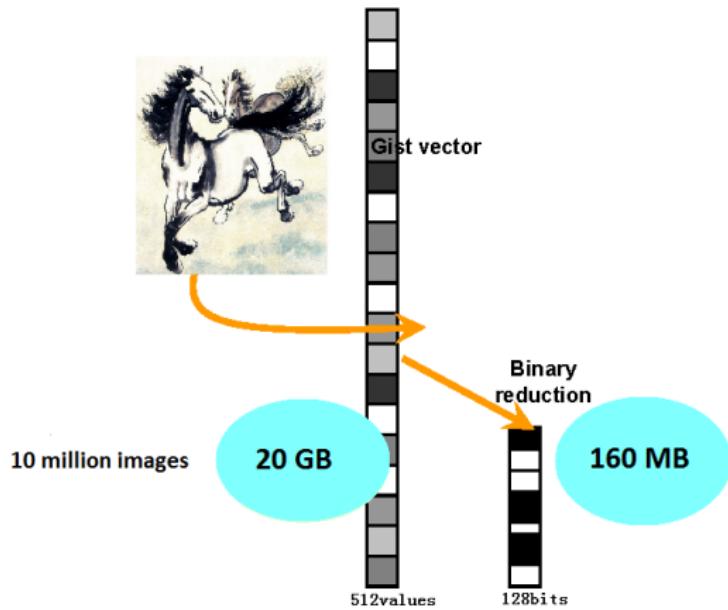
$$h(\text{Napoléon}) = \\ 01100101$$

Should be very different

Should be similar

flipped bit

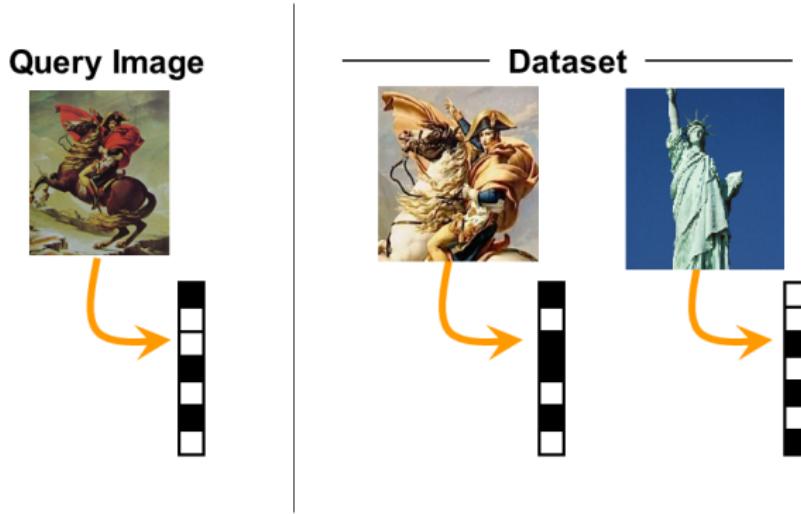
# Reduce Dimensionality and Storage Cost



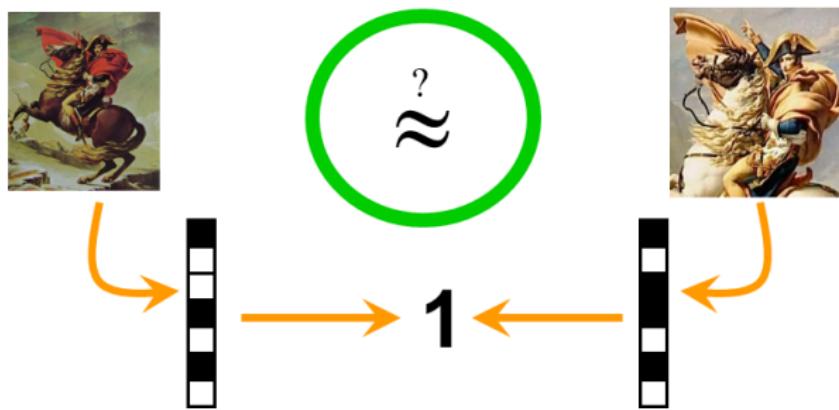
# Querying

Hamming distance:

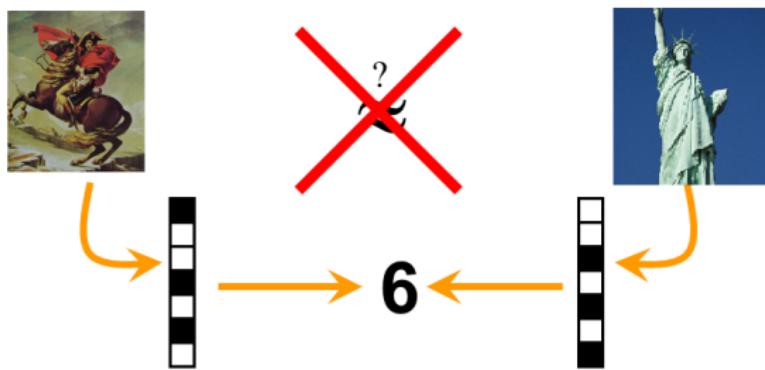
- $\|01101110, 00101101\|_H = 3$
- $\|11011, 01011\|_H = 1$



# Querying



# Querying



# Fast Query Speed

- By using hashing scheme, we can achieve **constant** or **sub-linear** search time complexity.
- **Exhaustive search** is also acceptable because the distance calculation cost is cheap now.

# Two Stages of Hash Function Learning

- Projection Stage (Dimension Reduction)
  - Projected with real-valued projection function
  - Given a point  $x$ , each projected dimension  $i$  will be associated with a real-valued projection function  $f_i(x)$  (e.g.  $f_i(x) = w_i^T x$ )
- Quantization Stage
  - Turn real into binary
  - Essential difference between metric learning and learning to hash

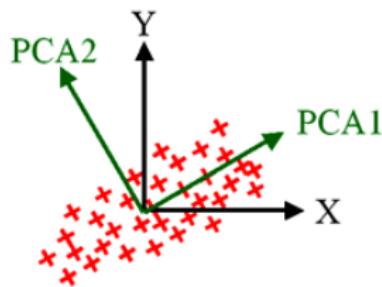
# Our Contribution

- Unsupervised Hashing [NIPS 2012]:  
**Isotropic hashing (IsoHash)**
- Supervised Hashing [SIGIR 2014]:  
**Supervised hashing with latent factor models**
- Multimodal Hashing [AAAI 2014]:  
**Large-scale supervised multimodal hashing with semantic correlation maximization**
- Multiple-Bit Quantization:
  - Double-bit quantization (DBQ) [AAAI 2012]
  - Manhattan quantization (MQ) [SIGIR 2012]

# Motivation

## Problem:

All existing methods use the **same number of bits** for different projected dimensions with **different variances**.



## Possible Solutions:

- Different number of bits for different dimensions  
**(Variable bit quantization (Moran et al, ACL 2013))**
- Isotropic (equal) variances for all dimensions

# PCA Hash

To generate a code of  $m$  bits, PCAH performs PCA on  $X$ , and then use the top  $m$  eigenvectors of the matrix  $XX^T$  as columns of the projection matrix  $W \in \mathbb{R}^{d \times m}$ . Here, top  $m$  eigenvectors are those corresponding to the  $m$  largest eigenvalues  $\{\lambda_k\}_{k=1}^m$ , generally arranged with the non-increasing order  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_m$ . Let  $\lambda = [\lambda_1, \lambda_2, \dots, \lambda_m]^T$ . Then

$$\Lambda = W^T XX^T W = \text{diag}(\lambda)$$

Define hash function

$$h(\mathbf{x}) = \text{sgn}(W^T \mathbf{x})$$

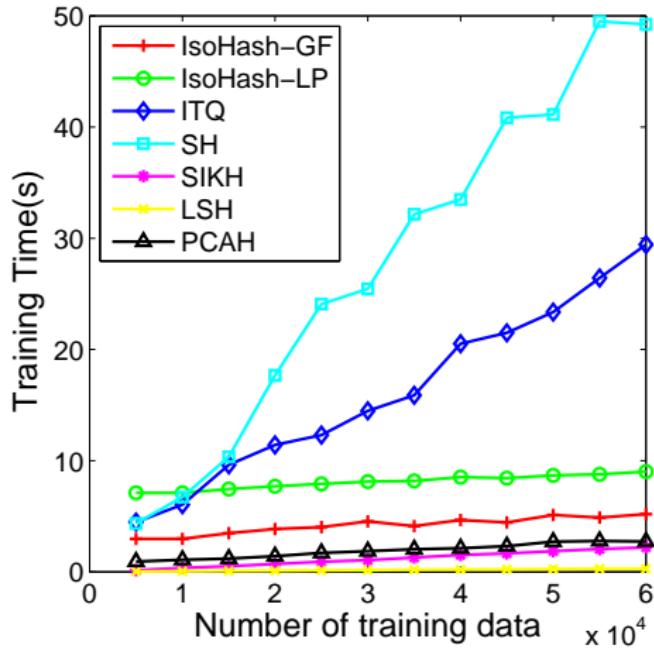
# Idea of IsoHash

- Learn an **orthogonal** matrix  $Q \in \mathbb{R}^{m \times m}$  which makes  $Q^T W^T X X^T W Q$  become a matrix with **equal diagonal values**.
- **Effect of  $Q$ :** to make each projected dimension has the **same variance** while **keeping the Euclidean distances** between any two points **unchanged**.

# Accuracy (mAP)

Method	CIFAR				
	32	64	96	128	256
IsoHash	0.2249	0.2969	<b>0.3256</b>	<b>0.3357</b>	<b>0.3651</b>
PCAH	0.0319	0.0274	0.0241	0.0216	0.0168
ITQ	<b>0.2490</b>	<b>0.3051</b>	0.3238	0.3319	0.3436
SH	0.0510	0.0589	0.0802	0.1121	0.1535
SIKH	0.0353	0.0902	0.1245	0.1909	0.3614
LSH	0.1052	0.1907	0.2396	0.2776	0.3432

# Training Time



# Problem Definition

## Input:

- Feature vectors:  $\mathbf{x}_i \in \mathbb{R}^D, i = 1, \dots, N.$   
(Compact form:  $\mathbf{X} \in \mathbb{R}^{N \times D}$ )
- Similarity labels:  $s_{ij}, i, j = 1, \dots, N.$   
(Compact form:  $\mathcal{S} = \{s_{ij}\}$ )
  - $s_{ij} = 1$  if points  $i$  and  $j$  belong to the same class.
  - $s_{ij} = 0$  if points  $i$  and  $j$  belong to different classes.

## Output:

- Binary codes:  $\mathbf{b}_i \in \{-1, 1\}^Q, i = 1, \dots, N.$   
(Compact form:  $\mathbf{B} \in \{-1, 1\}^{N \times Q}$ )
  - When  $s_{ij} = 1$ , the Hamming distance between  $\mathbf{b}_i$  and  $\mathbf{b}_j$  should be low.
  - When  $s_{ij} = 0$ , the Hamming distance between  $\mathbf{b}_i$  and  $\mathbf{b}_j$  should be high.

# Motivation

Existing supervised methods:

- High training complexity
- Semantic information is poorly utilized

# Model

The likelihood on the observed similarity labels  $\mathcal{S}$  is defined as:

$$p(\mathcal{S} \mid \mathbf{B}) = \prod_{s_{ij} \in \mathcal{S}} p(s_{ij} \mid \mathbf{B})$$

$$p(s_{ij} \mid \mathbf{B}) = \begin{cases} a_{ij}, & s_{ij} = 1 \\ 1 - a_{ij}, & s_{ij} = 0 \end{cases}$$

$a_{ij}$  is defined as  $a_{ij} = \sigma(\Theta_{ij})$  with:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\Theta_{ij} = \frac{1}{2} \mathbf{b}_i^T \mathbf{b}_j$$

Relationship between the Hamming distance and the inner product:

$$\text{dist}_H(\mathbf{b}_i, \mathbf{b}_j) = \frac{1}{2}(Q - \mathbf{b}_i^T \mathbf{b}_j) = \frac{1}{2}(Q - 2\Theta_{ij})$$

# Relaxation

Re-define  $\Theta_{ij}$  as:

$$\Theta_{ij} = \frac{1}{2} \mathbf{U}_{i*}^T \mathbf{U}_{j*}$$

$p(\mathcal{S} | \mathbf{B})$ ,  $p(\mathbf{B})$ ,  $p(\mathbf{B} | \mathcal{S})$  become  $p(\mathcal{S} | \mathbf{U})$ ,  $p(\mathbf{U})$ ,  $p(\mathbf{U} | \mathcal{S})$ .

Define a normal distribution of  $p(\mathbf{U})$  as:

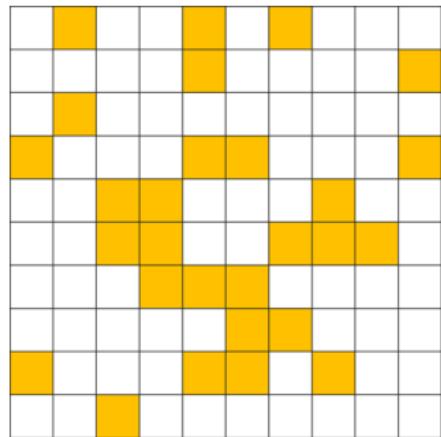
$$p(\mathbf{U}) = \prod_{d=1}^Q \mathcal{N}(\mathbf{U}_{*d} | \mathbf{0}, \beta \mathbf{I})$$

The log posterior of  $\mathbf{U}$  can be derived as:

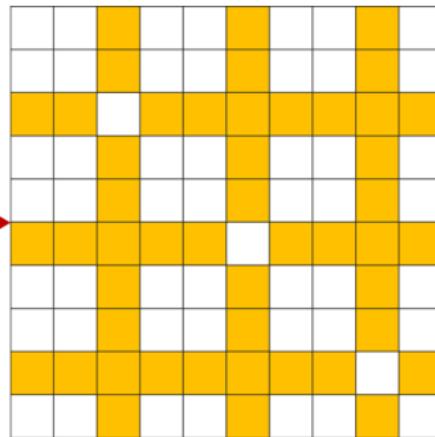
$$L = \log p(\mathbf{U} | \mathcal{S}) = \sum_{s_{ij} \in \mathcal{S}} (s_{ij} \Theta_{ij} - \log(1 + e^{\Theta_{ij}})) - \frac{1}{2\beta} \|\mathbf{U}\|_F^2 + c$$

# Stochastic Learning

Furthermore, if we choose the subset of  $\mathcal{S}$  by randomly selecting  $\mathcal{O}(Q)$  of its columns and rows, we can further reduce the time cost to  $\mathcal{O}(NQ^2)$  per iteration.

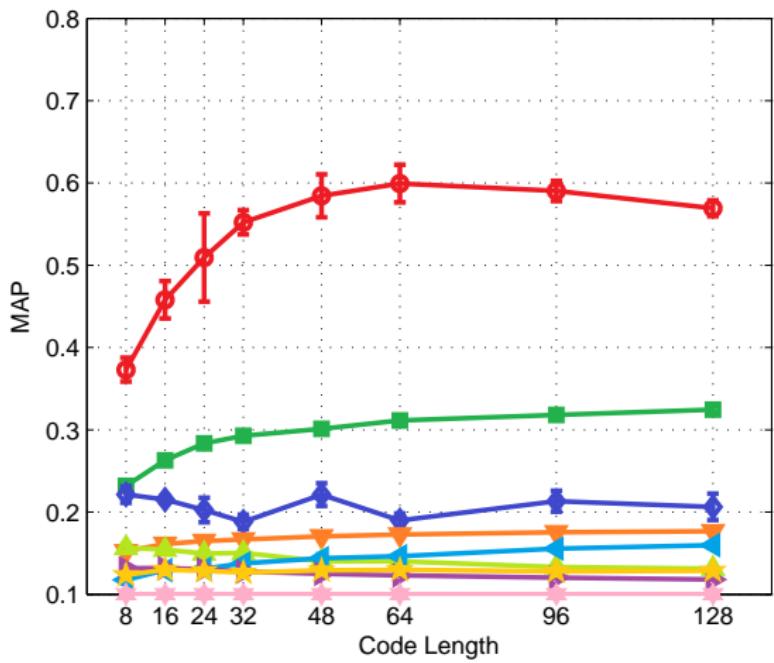


Disordered

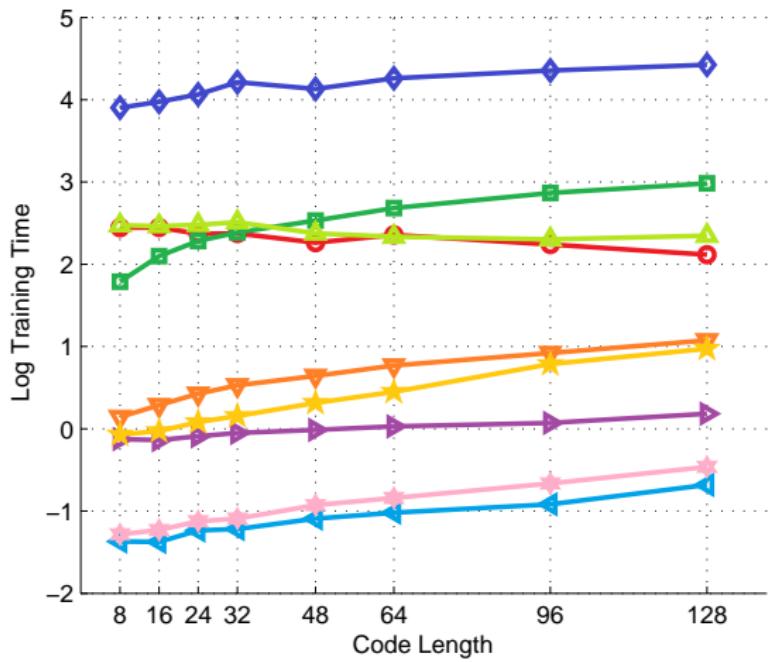


Aligned

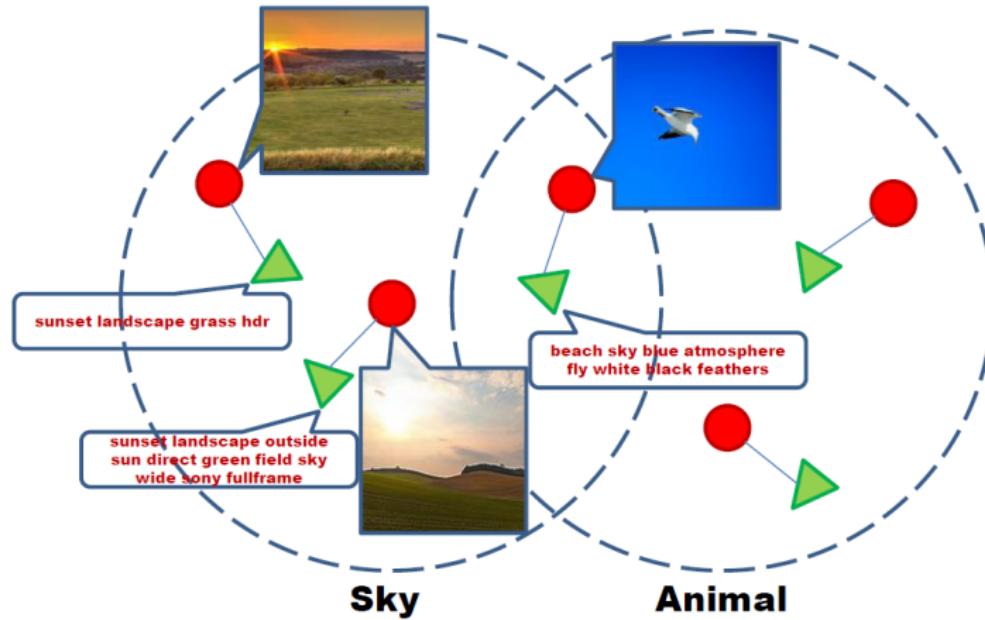
# MAP (CIFAR-10)



# Training Time (CIFAR-10)



# Supervised Multimodal Similarity Search



- Given a query of either image or text, return images or texts similar to it in both feature space and **semantics** (label information).

# Motivation and Contribution

## Motivation

- Existing supervised methods are not scalable

## Contribution

- Avoiding explicitly computing the pairwise similarity matrix, **linear-time** complexity w.r.t. the size of training data.
- A sequential learning method with **closed-form solution** to each bit, **no hyper-parameters** and **stopping conditions** are needed.

In matrix form, we can rewrite the problem as follows

$$\min_{W_x, W_y} \| \text{sgn}(XW_x) \text{sgn}(YW_y)^T - cS \|_F^2$$

$$\text{s.t. } \text{sgn}(XW_x)^T \text{sgn}(XW_x) = nI_c$$

$$\text{sgn}(YW_y)^T \text{sgn}(YW_y) = nI_c.$$

# Sequential Strategy

Assuming that the projection vectors  $w_x^{(1)}, \dots, w_x^{(t-1)}$  and  $w_y^{(1)}, \dots, w_y^{(t-1)}$  have been learned, to learn the next projection vectors  $w_x^{(t)}$  and  $w_y^{(t)}$ . Define a **residue matrix**

$$R_t = cS - \sum_{k=1}^{t-1} sgn(Xw_x^{(k)})sgn(Yw_y^{(k)})^T.$$

Objective function can be written as

$$\min_{w_x^{(t)}, w_y^{(t)}} \left\| sgn(Xw_x^{(t)})sgn(Yw_y^{(t)})^T - R_t \right\|_F^2.$$

**Algorithm 1** Learning Algorithm of SCM Hashing Method.

$$C_{xy}^{(0)} \leftarrow 2(X^T \tilde{L})(Y^T \tilde{L})^T - (X^T \mathbf{1}_n)(Y^T \mathbf{1}_n)^T;$$

$$C_{xy}^{(1)} \leftarrow c \times C_{xy}^{(0)};$$

$$C_{xx} \leftarrow X^T X + \gamma I_{d_x};$$

$$C_{yy} \leftarrow Y^T Y + \gamma I_{d_y};$$

**for**  $t = 1 \rightarrow c$  **do**

Solving the following generalized eigenvalue problem

$$C_{xy}^{(t)} C_{yy}^{-1} [C_{xy}^{(t)}]^T w_x = \lambda^2 C_{xx} w_x,$$

we can obtain the optimal solution  $w_x^{(t)}$  corresponding to the largest eigenvalue  $\lambda_{max}$ ;

$$w_y^{(t)} \leftarrow \frac{C_{yy}^{-1} C_{xy}^T w_x^{(t)}}{\lambda_{max}};$$

$$h_x^{(t)} \leftarrow sgn(X w_x^{(t)});$$

$$h_y^{(t)} \leftarrow sgn(Y w_y^{(t)});$$

$$C_{xy}^{(t+1)} \leftarrow C_{xy}^{(t)} - (X^T sgn(X w_x^{(t)}))(Y^T sgn(Y w_y^{(t)}))^T;$$

**end for**

# Scalability

**Table:** Training time (in seconds) on NUS-WIDE dataset by varying the size of training set.

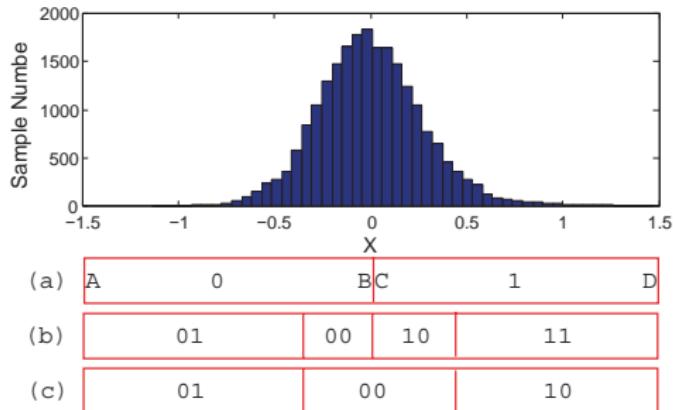
Method \ Size of Training Set	500	1000	1500	2000	2500	3000	5000	10000	20000
SCM-Seq	276	249	303	222	236	260	248	228	230
SCM-Orth	36	80	85	77	83	76	110	87	102
CCA	25	20	23	22	25	22	28	38	44
CCA-3V	69	57	68	69	62	55	67	70	86
CVH	62	116	123	149	155	170	237	774	1630
CRH	68	253	312	515	760	1076	-	-	-
MLBE	67071	126431	-	-	-	-	-	-	-

# Accuracy

Table: MAP results on NUS-WIDE. The best performance is shown in boldface.

Task	Method	Code Length		
		$c = 16$	$c = 24$	$c = 32$
Image Query v.s. Text Database	SCM-Seq	<b>0.4385</b>	<b>0.4397</b>	<b>0.4390</b>
	SCM-Orth	0.3804	0.3746	0.3662
	CCA	0.3625	0.3586	0.3565
	CCA-3V	0.3826	0.3741	0.3692
	CVH	0.3608	0.3575	0.3562
	CRH	0.3957	0.3965	0.3970
	MLBE	0.3697	0.3620	0.3540
Text Query v.s. Image Database	SCM-Seq	<b>0.4273</b>	<b>0.4265</b>	<b>0.4259</b>
	SCM-Orth	0.3757	0.3625	0.3581
	CCA	0.3619	0.3580	0.3560
	CCA-3V	0.3801	0.3721	0.3676
	CVH	0.3640	0.3596	0.3581
	CRH	0.3926	0.3910	0.3904
	MLBE	0.3877	0.3636	0.3551

# Double Bit Quantization



Point distribution of the real values computed by PCA on 22K LabelMe data set, and different coding results based on the distribution:

- (a) single-bit quantization (SBQ);
- (b) hierarchical hashing (HH);
- (c) double-bit quantization (DBQ).

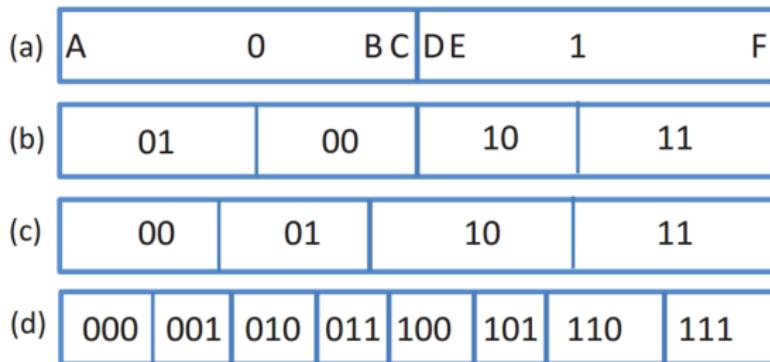
# Experiment

mAP on LabelMe data set

# bits	32			64		
	SBQ	HH	DBQ	SBQ	HH	DBQ
ITQ	0.2926	0.2592	<b>0.3079</b>	0.3413	0.3487	<b>0.4002</b>
SH	0.0859	0.1329	<b>0.1815</b>	0.1071	0.1768	<b>0.2649</b>
PCA	0.0535	0.1009	<b>0.1563</b>	0.0417	0.1034	<b>0.1822</b>
LSH	<b>0.1657</b>	0.105	0.12272	<b>0.2594</b>	0.2089	0.2577
SIKH	0.0590	0.0712	<b>0.0772</b>	0.1132	0.1514	<b>0.1737</b>

# bits	128			256		
	SBQ	HH	DBQ	SBQ	HH	DBQ
ITQ	0.3675	0.4032	<b>0.4650</b>	0.3846	0.4251	<b>0.4998</b>
SH	0.1730	0.2034	<b>0.3403</b>	0.2140	0.2468	<b>0.3468</b>
PCA	0.0323	0.1083	<b>0.1748</b>	0.0245	0.1103	<b>0.1499</b>
LSH	0.3579	0.3311	<b>0.4055</b>	0.4158	0.4359	<b>0.5154</b>
SIKH	0.2792	0.3147	<b>0.3436</b>	0.4759	0.5055	<b>0.5325</b>

# Manhattan Quantization



**Figure 1: Different quantization methods: (a) single-bit quantization (SBQ); (b) hierarchical quantization (HQ); (c) 2-bit Manhattan quantization (2-MQ); (d) 3-bit Manhattan quantization (3-MQ).**

# Experiment

**Table:** mAP on ANN\_SIFT1M data set. The best mAP among SBQ, HQ and 2-MQ under the same setting is shown in bold face.

# bits	32			64			96		
	SBQ	HQ	2-MQ	SBQ	HQ	2-MQ	SBQ	HQ	2-MH
ITQ	0.1657	0.2500	<b>0.2750</b>	0.4641	0.4745	<b>0.5087</b>	0.5424	0.5871	<b>0.6263</b>
SIKH	0.0394	0.0217	<b>0.0570</b>	0.2027	0.0822	<b>0.2356</b>	0.2263	0.1664	<b>0.2768</b>
LSH	0.1163	0.0961	<b>0.1173</b>	0.2340	0.2815	<b>0.3111</b>	0.3767	0.4541	<b>0.4599</b>
SH	0.0889	0.2482	<b>0.2771</b>	0.1828	0.3841	<b>0.4576</b>	0.2236	0.4911	<b>0.5929</b>
PCA	0.1087	0.2408	<b>0.2882</b>	0.1671	0.3956	<b>0.4683</b>	0.1625	0.4927	<b>0.5641</b>

# Outline

## 1 Introduction

## 2 Learning to Hash

- Isotropic Hashing
- Supervised Hashing with Latent Factor Models
- Supervised Multimodal Hashing with SCM
- Multiple-Bit Quantization

## 3 Distributed Learning

- Coupled Group Lasso for Web-Scale CTR Prediction
- Distributed Power-Law Graph Computing

## 4 Stochastic Learning

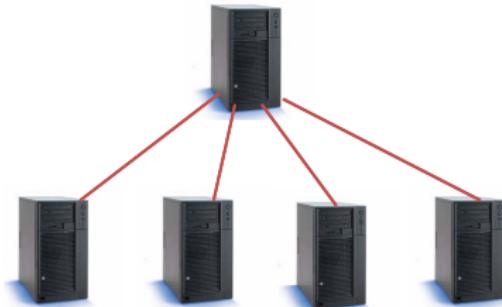
- Distributed Stochastic ADMM for Matrix Factorization

## 5 Conclusion

# Introduction

## Definition:

- Perform machine learning on clusters with several machines (nodes).

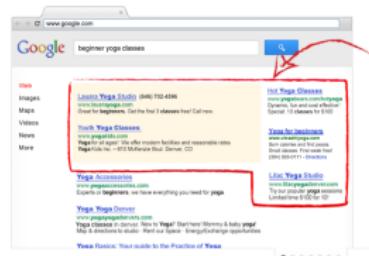


## Our contributions:

- Coupled Group Lasso for Web-Scale CTR Prediction in Display Advertising (ICML 2014)
- Distributed Power-law Graph Computing: Theoretical and Empirical Analysis (NIPS 2014)

# CTR Prediction for Online Advertising

- Multi-billion business on the web and accounts for the majority of the income for the major internet companies.
- Display advertising is a big part of online advertising.
- Click through rate (CTR) prediction is the problem of estimating the probability that an ad is clicked when displayed to a user in a specific context.



(a) Google



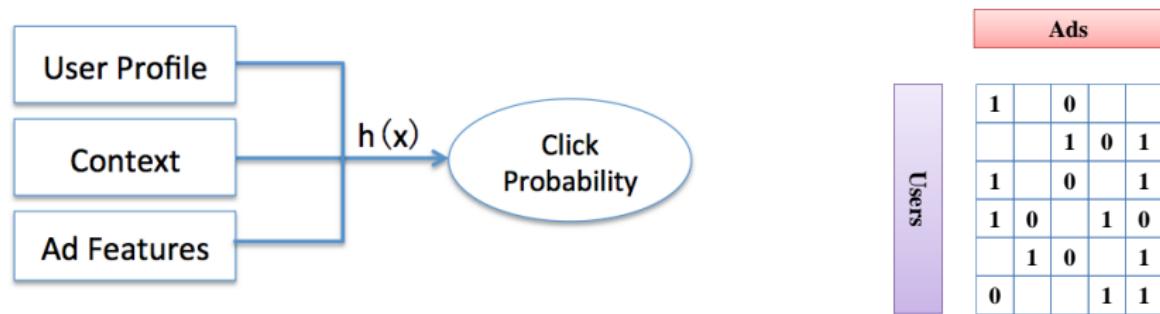
(b) Amazon



(c) Taobao

# Notation

- Impression (instance): ad + user + context
- Training set  $\{(\mathbf{x}^{(i)}, y^{(i)}) \mid i = 1, \dots, N\}$
- $\mathbf{x}^T = (\mathbf{x}_u^T, \mathbf{x}_a^T, \mathbf{x}_o^T)$
- $y \in \{0, 1\}$  with  $y = 1$  denoting *click* and  $y = 0$  denoting *non-click*
- Learn  $h(\mathbf{x}) = h(\mathbf{x}_u, \mathbf{x}_a, \mathbf{x}_o)$  to predict CTR



# Coupled Group Lasso (CGL)

## ① Likelihood

$$h(\mathbf{x}) = \Pr(y = 1 | \mathbf{x}, \mathbf{W}, \mathbf{V}, \mathbf{b}) = g((\mathbf{x}_u^T \mathbf{W})(\mathbf{x}_a^T \mathbf{V})^T + \mathbf{b}^T \mathbf{x}_o)$$

where

$$\mathbf{W} \in \mathbb{R}^{d_u \times k}, \mathbf{V} \in \mathbb{R}^{d_a \times k}, (\mathbf{x}_u^T \mathbf{W})(\mathbf{x}_a^T \mathbf{V})^T = \mathbf{x}_u^T (\mathbf{W} \mathbf{V}^T) \mathbf{x}_a$$

## ② Objective function

$$\min_{\mathbf{W}, \mathbf{V}, \mathbf{b}} \sum_{i=1}^N \xi(\mathbf{W}, \mathbf{V}, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)}) + \lambda \Omega(\mathbf{W}, \mathbf{V}),$$

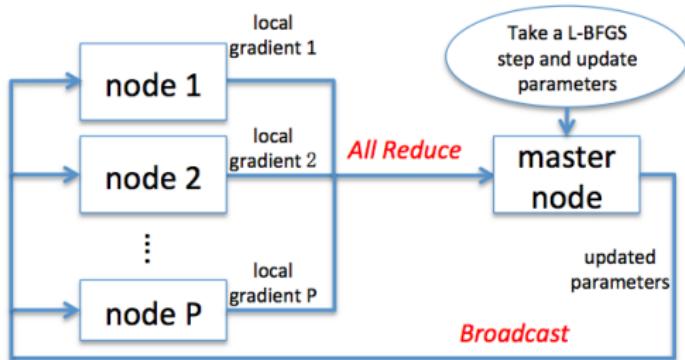
in which

$$\xi(\mathbf{W}, \mathbf{V}, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)}) = -\log \left( (h(\mathbf{x}^{(i)}))^{y^{(i)}} (1 - h(\mathbf{x}^{(i)}))^{1-y^{(i)}} \right)$$

$$\Omega(\mathbf{W}, \mathbf{V}) = \|\mathbf{W}\|_{2,1} + \|\mathbf{V}\|_{2,1} = \sum_{i=1}^{d_u} \|\mathbf{W}_{i*}\|_2 + \sum_{i=1}^{d_a} \|\mathbf{V}_{i*}\|_2$$

# Distributed Learning Framework

- Compute gradient  $\mathbf{g}'_p$  locally on each node  $p$  *in parallel*.
- Compute gradient  $\mathbf{g}' = \sum_{p=1}^P \mathbf{g}'_p$  with *AllReduce*.
- Add the gradient of the regularization term and take an L-BFGS step in the master node.
- *Broadcast* the updated parameters to each slaver node.



# Experiment

## Experiment Environment

- MPI-Cluster with hundreds of nodes, each of which is a 24-core server with 2.2GHz CPU and 96GB of RAM

## Baseline and Evaluation Metric

- Baseline: LR (with L2-norm) [Chapelle et al., 2013]
- Evaluation Metric (Discrimination)

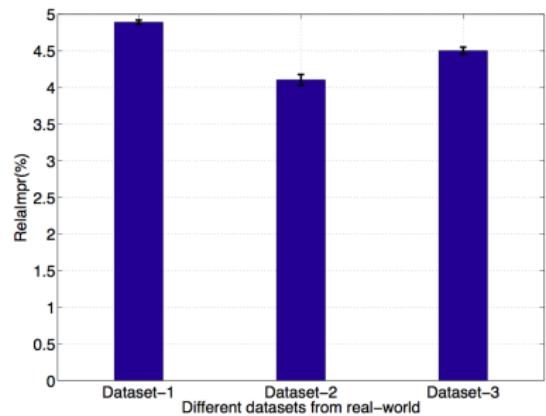
$$RelaImpr = \frac{AUC(model) - 0.5}{AUC(baseline) - 0.5} \times 100\%$$

# Data Sets

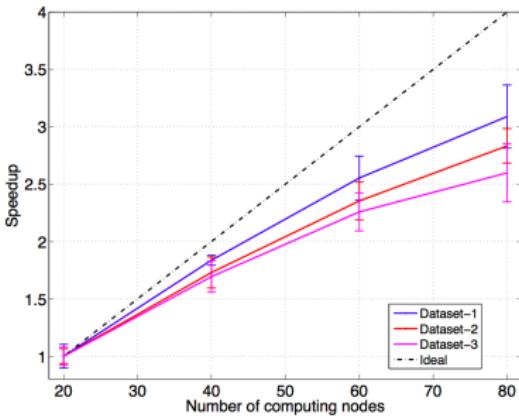
Data set	# Instances (Billion)	CTR (%)	# Ads	# Users (Million)	Storage (TB)
Train 1	1.011	1.62	21, 318	874.7	1.895
Test 1	0.295	1.70	11, 558	331.0	0.646
Train 2	1.184	1.61	21, 620	958.6	2.203
Test 2	0.145	1.64	6, 848	190.3	0.269
Train 3	1.491	1.75	33, 538	1119.3	2.865
Test 3	0.126	1.70	9, 437	183.7	0.233

Real-world data sets collected from *taobao* of Alibaba Group

# Performance



(c) RelalImpr w.r.t. Baseline



(d) Speedup

## Accuracy and Scalability

# Feature Selection

	Ad Part	User Part
Important Features	Women's clothes, Skirt, Dress, Children's wear, Shoes, Cellphone	Watch, Underwear, Fur clothing, Furniture
Useless Features	Movie, Act, Take-out, Food booking service	Stage Costume, Flooring, Pencil, Outdoor sock

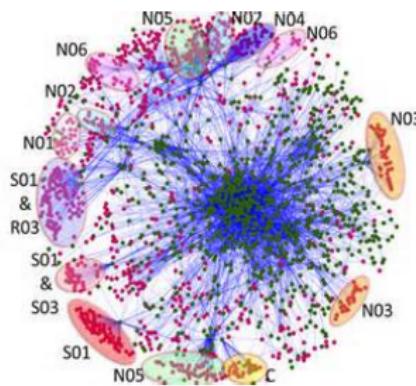
## Feature Selection Results

# Graph-based Machine Learning

- Big graphs emerge in many real applications
- Graph-based machine learning is a hot research topic with wide applications: *relational learning, manifold learning, PageRank, community detection, etc*
- Distributed graph computing frameworks for big graphs



(a) Social Network



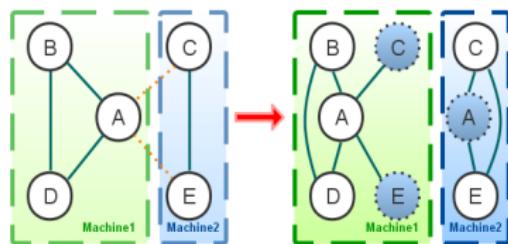
(b) Biological Network

# Graph Partitioning

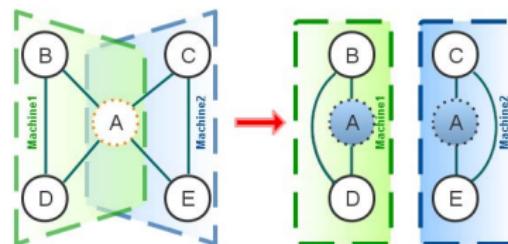
Graph partitioning (GP) plays a key role to affect the performance of distributed graph computing:

- Workload balance
- Communication cost

Two strategies for graph partitioning. Shaded vertices are ghosts and mirrors, respectively.



(a) Edge-Cut

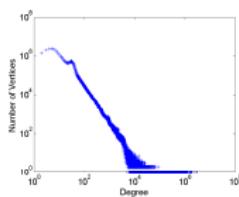


(b) Vertex-Cut

Theoretical and empirical results show vertex-cut is better than edge-cut.

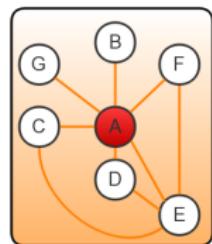
# Power-Law Graph Partitioning

Natural graphs from real world typically follow skewed power-law degree distributions:  $\Pr(d) \propto d^{-\alpha}$ .

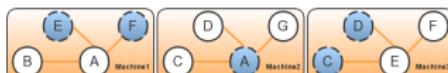


Different vertex-cut methods can result in different performance.

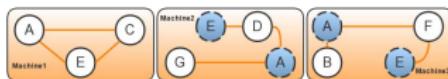
Intuition: cut the high-degree vertices



(a) Sample



(b) Bad partitioning



(c) Good partitioning

# Degree-based Hashing (DBH)

Existing GP methods, such as the **Random** in PowerGraph (Joseph E Gonzalez et al, 2012) and **Grid** in GraphBuilder (Nilesh Jain et al, 2013), do not make effective use of the power-law degree distribution.

We propose a novel GP method called **degree-based hashing (DBH)**:

---

## Algorithm 2 GP with DBH

---

**Input:** The set of edges  $E$ ; the set of vertices  $V$ ; number of machines  $p$ .

**Output:** The assignment  $M(e) \in \{1, \dots, p\}$  for each edge  $e$ .

Initialization: count the degree  $d_i$  for each vertex  $i \in \{1, \dots, n\}$  in parallel

**for all**  $e = (v_i, v_j) \in E$  **do**

    Hash each edge in parallel:

**if**  $d_i < d_j$  **then**

$M(e) \leftarrow \text{vertex\_hash}(v_i)$

**else**

$M(e) \leftarrow \text{vertex\_hash}(v_j)$

**end if**

**end for**

# Theoretical Analysis

We theoretically prove that our DBH method can outperform Random and Grid in terms of:

- reducing replication factor (**communication and storage cost**)
- keeping good edge-balance (**workload balance**)

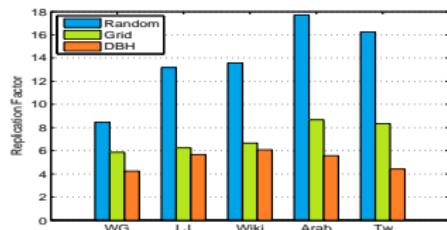
**Nice property:** Our DBH reduces more replication factor when the power-law graph is more skewed.

# Data Set

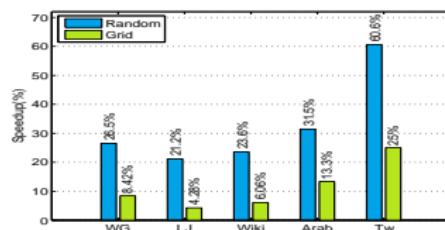
Table: Datasets

Alias	Graph	$ V $	$ E $
Tw	Twitter	42M	1.47B
Arab	Arabic-2005	22M	0.6B
Wiki	Wiki	5.7M	130M
LJ	LiveJournal	5.4M	79M
WG	WebGoogle	0.9M	5.1M

# Empirical Results (PageRank)

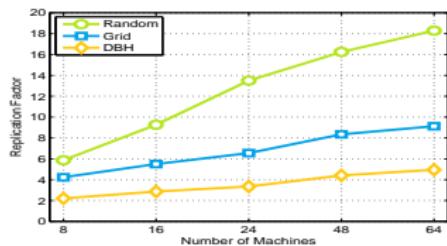


(d) Replication Factor

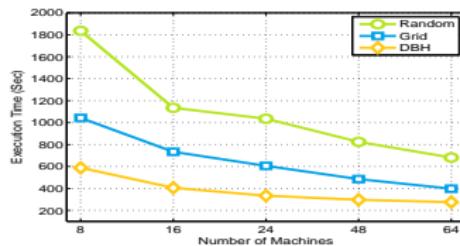


(e) Speedup relative to baselines

Figure: Experiments on real-world graphs. The number of machines is 48.



(a) Replication Factor



(b) Execution Time

Figure: The number of machines ranges from 8 to 64 on Twitter graph.

# Outline

## 1 Introduction

## 2 Learning to Hash

- Isotropic Hashing
- Supervised Hashing with Latent Factor Models
- Supervised Multimodal Hashing with SCM
- Multiple-Bit Quantization

## 3 Distributed Learning

- Coupled Group Lasso for Web-Scale CTR Prediction
- Distributed Power-Law Graph Computing

## 4 Stochastic Learning

- Distributed Stochastic ADMM for Matrix Factorization

## 5 Conclusion

# Introduction

## Definition:

- Given a (large) set of training data  $\{(\mathbf{x}_i, y_i) | i = 1, 2, \dots, n\}$ , each time we **randomly sample one training instance** for training.

Many machine learning problems can be formulated as:

$$\operatorname{argmin}_{\mathbf{w}} L(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{w} | \mathbf{x}_i, y_i)$$

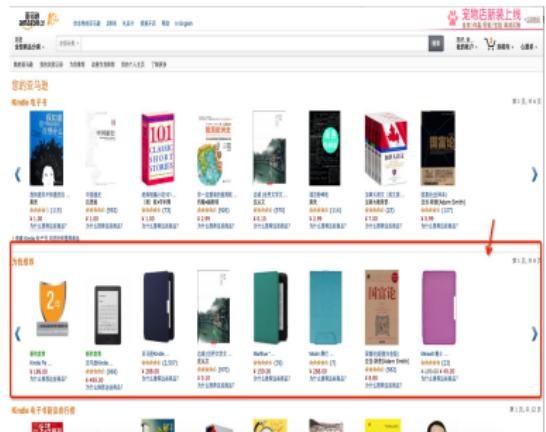
Hence, the computation cost can be greatly reduced by using stochastic learning.

## Our contribution:

- Distributed Stochastic ADMM for Matrix Factorization (CIKM 2014)

# Recommender System

- Recommend products (items) to customers (users) by utilizing the customers' historic preferences.



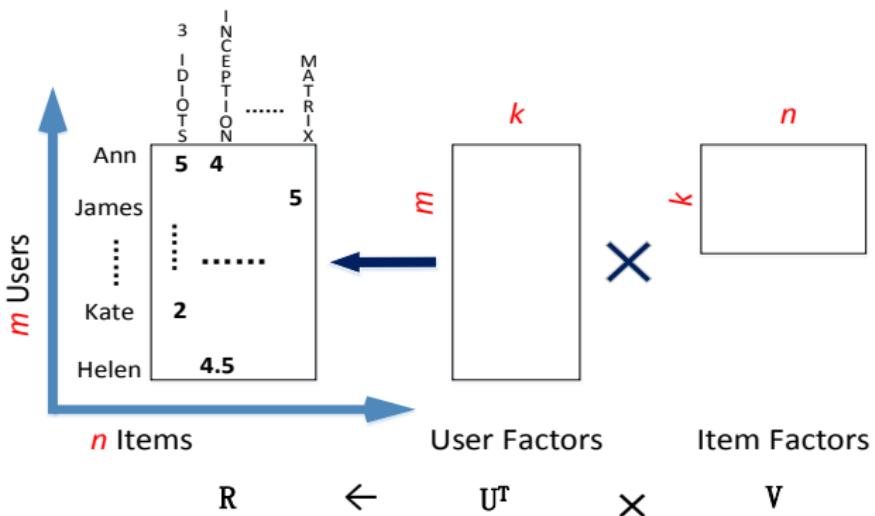
(a) Amazon



(b) Sina Weibo

# Matrix Factorization

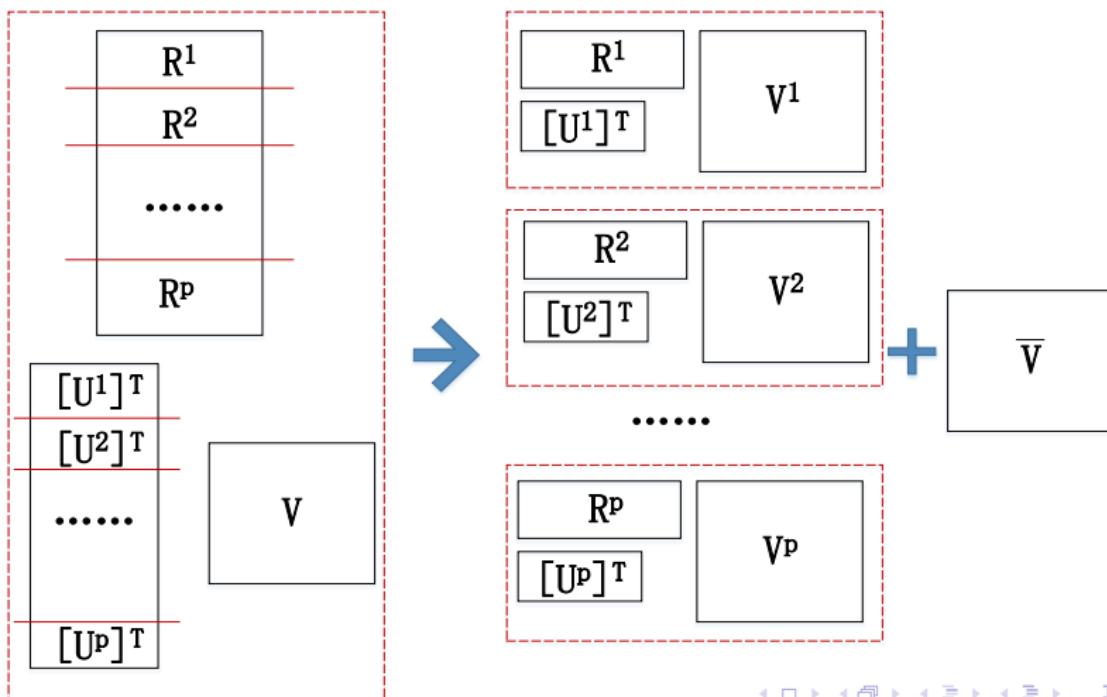
- Popular in recommender systems for its promising performance



$$\min_{\mathbf{U}, \mathbf{V}} \frac{1}{2} \sum_{(i,j) \in \Omega} \left[ (R_{i,j} - \mathbf{U}_{*i}^T \mathbf{V}_{*j})^2 + \lambda_1 \mathbf{U}_{*i}^T \mathbf{U}_{*i} + \lambda_2 \mathbf{V}_{*j}^T \mathbf{V}_{*j} \right]$$

# Data Split Strategy ( $P$ machines)

Decouple  $\mathbf{U}$  and  $\mathbf{V}$  as possible as we can:



# Distributed ADMM

Reformulated MF problem :

$$\begin{aligned} \min_{\mathbf{U}, \bar{\mathbf{V}}, \mathcal{V}} & \frac{1}{2} \sum_{p=1}^P \sum_{(i,j) \in \Omega^p} \left[ (R_{i,j} - \mathbf{U}_{*i}^T \mathbf{V}_{*j}^p)^2 + \lambda_1 \mathbf{U}_{*i}^T \mathbf{U}_{*i} + \lambda_2 [\mathbf{V}_{*j}^p]^T \mathbf{V}_{*j}^p \right] \\ \text{s.t. : } & \mathbf{V}^p - \bar{\mathbf{V}} = 0; \quad \forall p \in \{1, 2, \dots, P\} \end{aligned}$$

where  $\mathcal{V} = \{\mathbf{V}^p\}_{p=1}^P$ ,  $\Omega^p$  denotes the  $(i, j)$  indices of the ratings located in node  $p$

# Distributed ADMM

Define:

$$\begin{aligned} L^p(\mathbf{U}, \mathbf{V}^p, \boldsymbol{\Theta}^p, \bar{\mathbf{V}}) &= f^p(\mathbf{U}, \mathbf{V}^p) + l^p(\mathbf{V}^p, \bar{\mathbf{V}}, \boldsymbol{\Theta}^p) \\ &= \sum_{(i,j) \in \Omega^p} \hat{f}_{i,j}(\mathbf{U}_{*i}, \mathbf{V}_{*j}^p) \\ &\quad + \left[ \frac{\rho}{2} \|\mathbf{V}^p - \bar{\mathbf{V}}\|_F^2 + \text{tr}([\boldsymbol{\Theta}^p]^T (\mathbf{V}^p - \bar{\mathbf{V}})) \right], \end{aligned}$$

we can get

$$L(\mathbf{U}, \mathcal{V}, \mathcal{O}, \bar{\mathbf{V}}) = \sum_{p=1}^P L^p(\mathbf{U}, \mathbf{V}^p, \boldsymbol{\Theta}^p, \bar{\mathbf{V}}).$$

# Distributed ADMM

Get the solutions by repeating the following three steps:

$$\mathbf{U}_{t+1}, \mathbf{V}_{t+1}^p \leftarrow \underset{\mathbf{U}, \mathbf{V}^p}{\operatorname{argmin}} L^p(\mathbf{U}, \mathbf{V}^p, \boldsymbol{\Theta}_t^p, \bar{\mathbf{V}}_t), \forall p \in \{1, 2, \dots, P\}$$

$$\bar{\mathbf{V}}_{t+1} \leftarrow \underset{\bar{\mathbf{V}}}{\operatorname{argmin}} L(\mathbf{U}_{t+1}, \mathcal{V}_{t+1}, \mathcal{O}_t, \bar{\mathbf{V}}),$$

$$\boldsymbol{\Theta}_{t+1}^p \leftarrow \boldsymbol{\Theta}_t^p + \rho(\mathbf{V}_{t+1}^p - \bar{\mathbf{V}}_{t+1}), \forall p \in \{1, 2, \dots, P\}.$$

The solution for  $\bar{\mathbf{V}}_{t+1}$  is:

$$\bar{\mathbf{V}}_{t+1} = \frac{1}{P} \sum_{p=1}^P \mathbf{V}_{t+1}^p$$

which can be calculated efficiently.

The problem lies in getting  $\mathbf{U}_{t+1}, \mathcal{V}_{t+1}$  efficiently

# Stochastic Learning

Batch learning is still not very efficient:

$$\begin{aligned}\mathbf{U}_{t+1} &= \mathbf{U}_t - \tau_t * \nabla_{\mathbf{U}}^T f^p(\mathbf{U}_t, \mathbf{V}_t^p), \\ \mathbf{V}_{t+1}^p &= \frac{\tau_t}{1 + \rho \tau_t} \left[ \frac{\mathbf{V}_t^p}{\tau_t} + \rho \bar{\mathbf{V}}_t - \boldsymbol{\Theta}_t^p - \nabla_{\mathbf{V}^p}^T f^p(\mathbf{U}_t, \mathbf{V}_t^p) \right]\end{aligned}$$

## Stochastic Learning

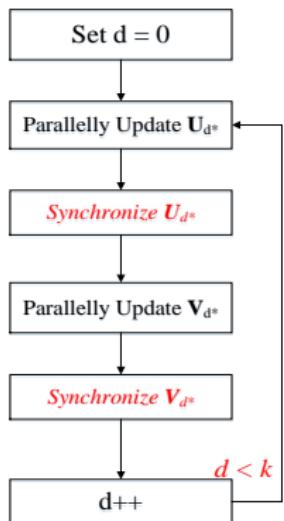
$$\begin{aligned}(\mathbf{U}_{*i})_{t+1} &= (\mathbf{U}_{*i})_t + \tau_t (\epsilon_{ij} (\mathbf{V}_{*j}^p)_t - \lambda_1 (\mathbf{U}_{*i})_t), \\ (\mathbf{V}_{*j}^p)_{t+1} &= \frac{\tau_t}{1 + \rho \tau_t} \left[ \frac{1 - \lambda_2 \tau_t}{\tau_t} (\mathbf{V}_{*j}^p)_t \right. \\ &\quad \left. + \epsilon_{ij} (\mathbf{U}_{*i})_t + \rho (\bar{\mathbf{V}}_{*j})_t - (\boldsymbol{\Theta}_{*j}^p)_t \right],\end{aligned}$$

where  $\epsilon_{ij} = R_{ij} - [(\mathbf{U}_{*i})_t]^T (\mathbf{V}_{*j}^p)_t$ .

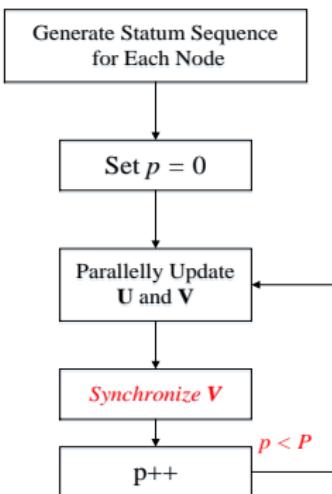
# Scheduler Comparison

Baselines: CCD++ (H.-F Yu etc, ICDM'12); DSGD (R. Gemulla etc, KDD'11)

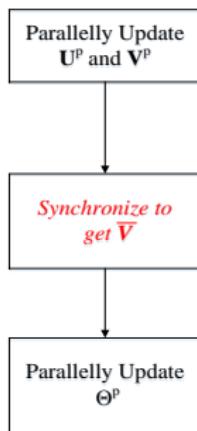
Number of synchronization operations for one iteration to fully scan all the ratings:



(a) CCD++  
synchronizes **k** times



(b) DSGD  
synchronizes **P** times



(c) DS-ADMM  
synchronizes **1** time

# Experiment

- Experiment environment
  - MPI-Cluster with 20 nodes, each of which is a 24-core server with 2.2GHz CPU and 96GB of RAM;
  - One core and 10GB memory for each node are actually used.
- Baseline and evaluation metric
  - Baseline: CCD++, DSGD, DSGD-Bias
  - Evaluation metric:

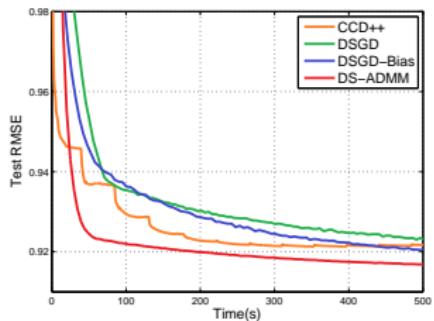
$$\text{test RMSE} : \frac{1}{Q} \sqrt{\sum (R_{i,j} - \mathbf{U}_{*i}^T \bar{\mathbf{V}}_{*j})^2}$$

# Data Sets

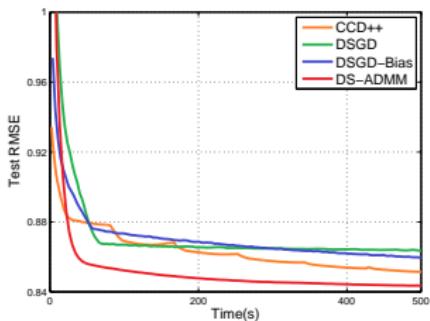
Table: Data sets and parameter settings

Data Set	Netflix	Yahoo! Music R1	Yahoo! Music R2
$m$	480,190	1,938,361	1,823,179
$n$	17,770	49,995	136,736
#Train	99,072,112	73,578,902	699,640,226
#Test	1,408,395	7,534,592	18,231,790
$k$	40	40	40
$\eta_0/\tau_0$	0.1	0.1	0.1
$\lambda_1/\lambda_2$	0.05	0.05	0.05
$\rho$	0.05	0.05	0.1
$\alpha$	0.002	0.002	0.006
$\beta$	0.7	0.7	0.7
$P$	8	10	20

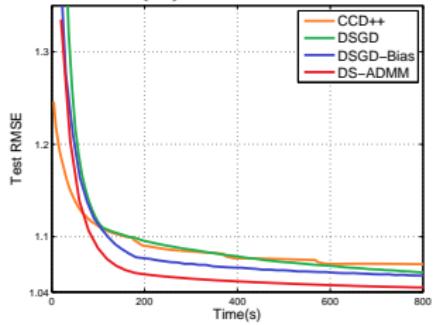
# Accuracy and Efficiency



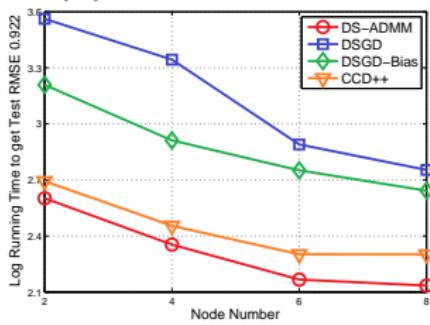
(a) Netflix



(b) Yahoo-Music-R1

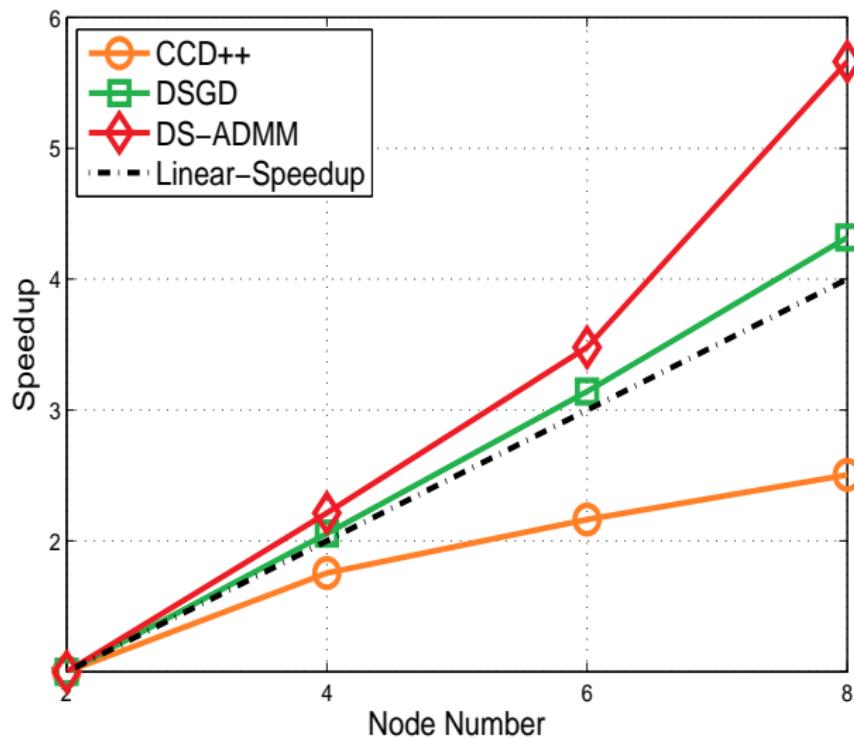


(c) Yahoo-Music-R2



(d) Time to fixed-RMSE (0.922)

# Speedup



# Outline

## 1 Introduction

## 2 Learning to Hash

- Isotropic Hashing
- Supervised Hashing with Latent Factor Models
- Supervised Multimodal Hashing with SCM
- Multiple-Bit Quantization

## 3 Distributed Learning

- Coupled Group Lasso for Web-Scale CTR Prediction
- Distributed Power-Law Graph Computing

## 4 Stochastic Learning

- Distributed Stochastic ADMM for Matrix Factorization

## 5 Conclusion

# Our Contribution

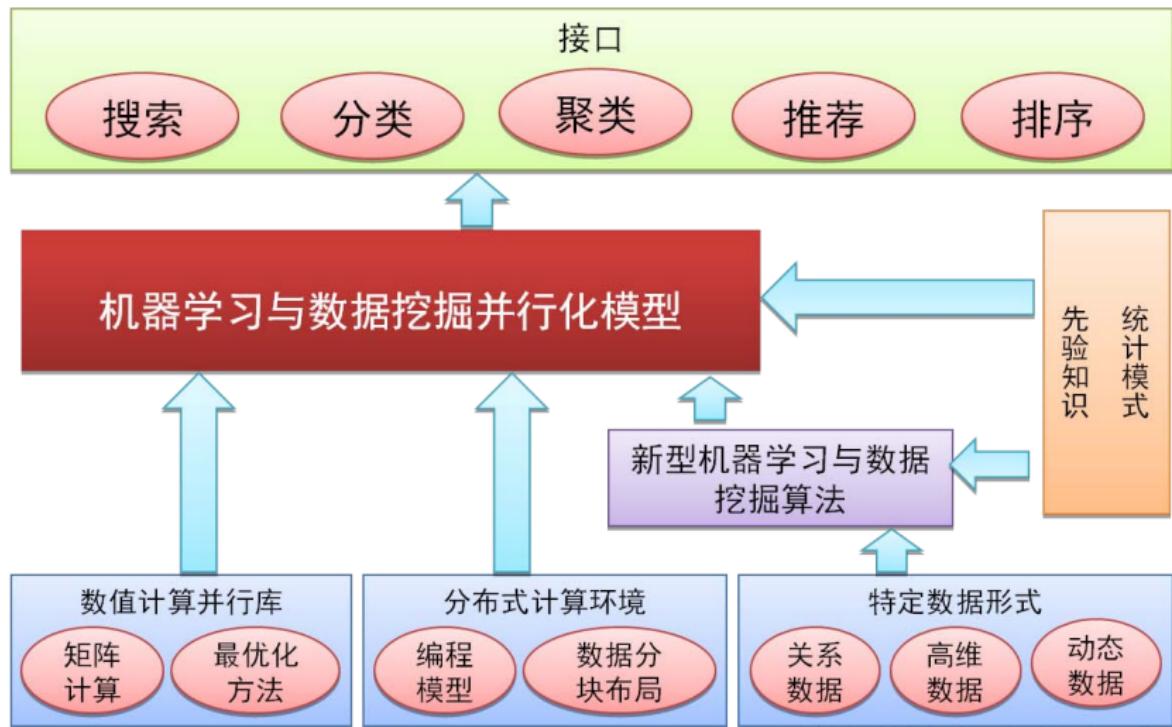
- Learning to hash (哈希学习): memory/disk/cpu/communication
- Distributed learning (分布式学习): memory/disk/cpu;  
but increase communication cost
- Stochastic learning (随机学习): memory/disk/cpu

# Future Work

- Learning to hash for decreasing communication cost
- Distributed **programming models** and **platforms** for machine learning:  
*MPI (fault tolerance, asynchronous), GraphLab, Parameter Server, Spark, MapReduce, Storm, GPU, etc*
- Distributed stochastic learning:  
*communication modeling*

# Future Work

Big data machine learning (big learning) framework



## Related Publication (1/2)[\*indicate my students]

- Ling Yan\*, Wu-Jun Li, Gui-Rong Xue, Dingyi Han. Coupled Group Lasso for Web-Scale CTR Prediction in Display Advertising. *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014.
- Peichao Zhang\*, Wei Zhang\*, Wu-Jun Li, Minyi Guo. Supervised Hashing with Latent Factor Models. *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2014.
- Cong Xie\*, Ling Yan\*, Wu-Jun Li, Zhihua Zhang. Distributed Power-law Graph Computing: Theoretical and Empirical Analysis. *To Appear in Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS)*, 2014.
- Zhi-Qin Yu\*, Xing-Jian Shi\*, Ling Yan\*, Wu-Jun Li. Distributed Stochastic ADMM for Matrix Factorization. *To Appear in Proceedings of the 23rd ACM International Conference on Information and Knowledge Management (CIKM)*, 2014.

## Related Publication (2/2)[\*indicate my students]

- Dongqing Zhang\*, Wu-Jun Li. Large-Scale Supervised Multimodal Hashing with Semantic Correlation Maximization. *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI)*, 2014.
- Weihao Kong\*, Wu-Jun Li. Isotropic Hashing. *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS)*, 2012.
- Weihao Kong\*, Wu-Jun Li, Minyi Guo. Manhattan Hashing for Large-Scale Image Retrieval. *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2012.
- Weihao Kong\*, Wu-Jun Li. Double-Bit Quantization for Hashing. *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

## Q &amp; A

Thanks!

