Big Data Machine Learning

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

Oct 31, 2014
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 academic 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

- 10 million images
- 20 GB
- 512 values
- Gist vector
- Binary reduction
- 128 bits
- 160 MB
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
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 (Unfortunately, have not found an effective way)
- 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 $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$. Let $\lambda = [\lambda_1, \lambda_2, \cdots, \lambda_m]^T$. Then

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

Define hash function

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

<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># bits</td>
</tr>
<tr>
<td>IsoHash</td>
<td></td>
</tr>
<tr>
<td>PCAH</td>
<td></td>
</tr>
<tr>
<td>ITQ</td>
<td></td>
</tr>
<tr>
<td>SH</td>
<td></td>
</tr>
<tr>
<td>SIKH</td>
<td></td>
</tr>
<tr>
<td>LSH</td>
<td></td>
</tr>
</tbody>
</table>

Li ([http://cs.nju.edu.cn/lwj](http://cs.nju.edu.cn/lwj))
## Training Time

![Graph showing training time vs number of training data for various hashing methods](image)

<table>
<thead>
<tr>
<th>Method</th>
<th>Training Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsoHash–GF</td>
<td></td>
</tr>
<tr>
<td>IsoHash–LP</td>
<td></td>
</tr>
<tr>
<td>ITQ</td>
<td></td>
</tr>
<tr>
<td>SH</td>
<td></td>
</tr>
<tr>
<td>SIKH</td>
<td></td>
</tr>
<tr>
<td>LSH</td>
<td></td>
</tr>
<tr>
<td>PCAH</td>
<td></td>
</tr>
</tbody>
</table>

Li ([http://cs.nju.edu.cn/lwj](http://cs.nju.edu.cn/lwj))
Problem Definition

Input:
- Feature vectors: \( x_i \in \mathbb{R}^D, i = 1, \ldots, N. \) (Compact form: \( X \in \mathbb{R}^{N \times D} \))
- Similarity labels: \( s_{ij}, i, j = 1, \ldots, N. \) (Compact form: \( S = \{s_{ij}\} \))
  - \( s_{ij} = 1 \) 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: \( b_i \in \{-1, 1\}^Q, i = 1, \ldots, N. \) (Compact form: \( B \in \{-1, 1\}^{N \times Q} \))
  - When \( s_{ij} = 1 \), the Hamming distance between \( b_i \) and \( b_j \) should be low.
  - When \( s_{ij} = 0 \), the Hamming distance between \( b_i \) and \( b_j \) should be high.
Motivation

Existing supervised methods:

- High training complexity
- Semantic information is poorly utilized
The likelihood on the observed similarity labels $S$ is defined as:

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

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

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

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

$$\Theta_{ij} = \frac{1}{2} b_i^T b_j$$

Relationship between the Hamming distance and the inner product:

$$\text{dist}_H(b_i, b_j) = \frac{1}{2}(Q - b_i^T b_j) = \frac{1}{2}(Q - 2\Theta_{ij})$$
Relaxation

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

$$\Theta_{ij} = \frac{1}{2} U_{i*}^T U_{j*}$$

$p(S \mid B)$, $p(B)$, $p(B \mid S)$ become $p(S \mid U)$, $p(U)$, $p(U \mid S)$. Define a normal distribution of $p(U)$ as:

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

The log posteriori of $U$ can be derived as:

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

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

Supervised Hashing with Latent Factor Models

MAP (CIFAR-10)

Li (http://cs.nju.edu.cn/lwj)
Training Time (CIFAR-10)

![Graph showing training time for different codes lengths. The x-axis represents code length, and the y-axis represents log training time. The graph includes different methods such as LFH, KSH, MLH, ITQ, AGH, LSH, PCAH, SH, and SIKH. Each method is represented by a different marker and color. The graph shows the relationship between code length and log training time for each method.](http://cs.nju.edu.cn/lwj)
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

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

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 $w^{(1)}_x, ..., w^{(t-1)}_x$ and $w^{(1)}_y, ..., w^{(t-1)}_y$ have been learned, to learn the next projection vectors $w^{(t)}_x$ and $w^{(t)}_y$. Define a residue matrix

$$R_t = cS - \sum_{k=1}^{t-1} \text{sgn}(Xw^{(k)}_x)\text{sgn}(Yw^{(k)}_y)^T.$$ 

Objective function can be written as

$$\min_{w^{(t)}_x, w^{(t)}_y} \left\| \text{sgn}(Xw^{(t)}_x)\text{sgn}(Yw^{(t)}_y)^T - R_t \right\|_F^2.$$
Algorithm 1 Learning Algorithm of SCM Hashing Method.

\[
C_{xy}^{(0)} \leftarrow 2 (X^T \tilde{L} (Y^T \tilde{L}))^T - (X^T 1_n)(Y^T 1_n)^T; \\
C_{xy}^{(1)} \leftarrow c \times C_{xy}^{(0)}; \\
C_{xx} \leftarrow X^T X + \gamma I_{d_x}; \\
C_{yy} \leftarrow Y^T Y + \gamma I_{d_y}; \\
\text{for } t = 1 \rightarrow c \text{ do} \\
\quad \text{Solving the following generalized eigenvalue problem} \\
\quad C_{xy}^{(t)} C_{yy}^{-1} [C_{xy}^{(t)}]^T w_x = \lambda^2 C_{xx} w_x, \\
\quad \text{we can obtain the optimal solution } w_x^{(t)} \text{ corresponding to the largest} \\
\quad \text{eigenvalue } \lambda_{max}; \\
\quad w_y^{(t)} \leftarrow \frac{C_{yy}^{-1} C_{xy}^T w_x^{(t)}}{\lambda_{max}}; \\
\quad h_x^{(t)} \leftarrow \text{sgn}(X w_x^{(t)}); \\
\quad h_y^{(t)} \leftarrow \text{sgn}(Y w_y^{(t)}); \\
\quad C_{xy}^{(t+1)} \leftarrow C_{xy}^{(t)} - (X^T \text{sgn}(X w_x^{(t)}))(Y^T \text{sgn}(Y w_y^{(t)}))^T; \\
\text{end for}
\]
Scalability

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

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

Li (http://cs.nju.edu.cn/lwj)
### Accuracy

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

<table>
<thead>
<tr>
<th>Task</th>
<th>Method</th>
<th>Code Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$c = 16$</td>
</tr>
<tr>
<td>Image Query v.s. Text Database</td>
<td>SCM-Seq</td>
<td><strong>0.4385</strong></td>
</tr>
<tr>
<td></td>
<td>SCM-Orth</td>
<td>0.3804</td>
</tr>
<tr>
<td></td>
<td>CCA</td>
<td>0.3625</td>
</tr>
<tr>
<td></td>
<td>CCA-3V</td>
<td>0.3826</td>
</tr>
<tr>
<td></td>
<td>CVH</td>
<td>0.3608</td>
</tr>
<tr>
<td></td>
<td>CRH</td>
<td>0.3957</td>
</tr>
<tr>
<td></td>
<td>MLBE</td>
<td>0.3697</td>
</tr>
<tr>
<td>Text Query v.s. Image Database</td>
<td>SCM-Seq</td>
<td><strong>0.4273</strong></td>
</tr>
<tr>
<td></td>
<td>SCM-Orth</td>
<td>0.3757</td>
</tr>
<tr>
<td></td>
<td>CCA</td>
<td>0.3619</td>
</tr>
<tr>
<td></td>
<td>CCA-3V</td>
<td>0.3801</td>
</tr>
<tr>
<td></td>
<td>CVH</td>
<td>0.3640</td>
</tr>
<tr>
<td></td>
<td>CRH</td>
<td>0.3926</td>
</tr>
<tr>
<td></td>
<td>MLBE</td>
<td>0.3877</td>
</tr>
</tbody>
</table>
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).
### mAP on LabelMe data set

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

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>0</th>
<th>B</th>
<th>C</th>
<th>DE</th>
<th>1</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>01</td>
<td>00</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
<td>110</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th># bits</th>
<th>32</th>
<th></th>
<th>64</th>
<th></th>
<th>96</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SBQ</td>
<td>HQ</td>
<td>2-MQ</td>
<td>SBQ</td>
<td>HQ</td>
<td>2-MQ</td>
</tr>
<tr>
<td>ITQ</td>
<td>0.1657</td>
<td>0.2500</td>
<td><strong>0.2750</strong></td>
<td>0.4641</td>
<td>0.4745</td>
<td><strong>0.5087</strong></td>
</tr>
<tr>
<td>SIKH</td>
<td>0.0394</td>
<td>0.0217</td>
<td><strong>0.0570</strong></td>
<td>0.2027</td>
<td>0.0822</td>
<td><strong>0.2356</strong></td>
</tr>
<tr>
<td>LSH</td>
<td>0.1163</td>
<td>0.0961</td>
<td><strong>0.1173</strong></td>
<td>0.2340</td>
<td>0.2815</td>
<td><strong>0.3111</strong></td>
</tr>
<tr>
<td>SH</td>
<td>0.0889</td>
<td>0.2482</td>
<td><strong>0.2771</strong></td>
<td>0.1828</td>
<td>0.3841</td>
<td><strong>0.4576</strong></td>
</tr>
<tr>
<td>PCA</td>
<td>0.1087</td>
<td>0.2408</td>
<td><strong>0.2882</strong></td>
<td>0.1671</td>
<td>0.3956</td>
<td><strong>0.4683</strong></td>
</tr>
</tbody>
</table>
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
Perform machine learning on clusters with several machines (nodes).
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 \( \{(x^{(i)}, y^{(i)}) \mid i = 1, \ldots, N\} \)
- \( x^T = (x^T_u, x^T_a, x^T_o) \)
- \( y \in \{0, 1\} \) with \( y = 1 \) denoting click and \( y = 0 \) denoting non-click
- Learn \( h(x) = h(x_u, x_a, x_o) \) to predict CTR
Coupled Group Lasso (CGL)

1. Likelihood

\[ h(x) = Pr(y = 1|x, W, V, b) = g \left( (x_u^T W)(x_a^T V)^T + b^T x_o \right) \]

where

\[ W \in \mathbb{R}^{d_u \times k}, \ V \in \mathbb{R}^{d_a \times k}, \ (x_u^T W)(x_a^T V)^T = x_u (WV^T) x_a \]

2. Objective function

\[
\min_{W, V, b} \sum_{i=1}^{N} \xi \left( W, V, b; x^{(i)}, y^{(i)} \right) + \lambda \Omega(W, V),
\]

in which

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

\[
\Omega(W, V) = \| W \|_{2,1} + \| V \|_{2,1} = \sum_{i=1}^{d_u} \| W_{i*} \|_2 + \sum_{i=1}^{d_a} \| V_{i*} \|_2
\]
Distributed Learning Framework

- Compute gradient $g'_p$ locally on each node $p$ in parallel.
- Compute gradient $g' = \sum_{p=1}^{P} g'_p$ 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.

![Diagram](image_url)
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)

\[
\text{RelaImpr} = \frac{AUC(\text{model}) - 0.5}{AUC(\text{baseline}) - 0.5} \times 100\%
\]
Data Sets

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

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

(c) RelImpr w.r.t. Baseline

(d) Speedup

Accuracy and Scalability
## Feature Selection

<table>
<thead>
<tr>
<th>Important Features</th>
<th>Ad Part</th>
<th>User Part</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Women’s clothes, Skirt, Dress, Children’s wear, Shoes, Cellphone</td>
<td>Watch, Underwear, Fur clothing, Furniture</td>
</tr>
</tbody>
</table>

| 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.

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.

(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 $M(e) \in \{1, \ldots, p\}$ for each edge $e$.

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

| 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

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

(d) Replication Factor

(e) Speedup relative to baselines

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
Definition

Given a (large) set of training data \( \{(x_i, y_i) | i = 1, 2, \cdots, n\} \), each time we randomly sample one training instance for training.

Many machine learning problems can be formulated as:

\[
\arg\min_w L(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w | x_i, y_i)
\]

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

\[
\min_{U,V} \frac{1}{2} \sum_{(i,j) \in \Omega} \left[ (R_{i,j} - U_{*i}^T V_{*j})^2 + \lambda_1 U_{*i}^T U_{*i} + \lambda_2 V_{*j}^T V_{*j} \right]
\]
Data Split Strategy

Decouple $U$ and $V$ as possible as we can:

$$\begin{align*}
\begin{bmatrix}
R^1 \\
R^2 \\
\vdots \\
R^p
\end{bmatrix}
& \begin{bmatrix}
[U^1]^T \\
[U^2]^T \\
\vdots \\
[U^p]^T
\end{bmatrix}
\end{align*}$$

$$\begin{align*}
\begin{bmatrix}
V^1 \\
V^2 \\
\vdots \\
V^p
\end{bmatrix}
& \begin{bmatrix}
\bar{V}
\end{bmatrix}
\end{align*}$$
Reformulated MF problem:

\[
\min_{\mathbf{U}, \mathbf{V}, \nu} \frac{1}{2} \sum_{p=1}^{P} \sum_{(i,j) \in \Omega^p} \left[ (R_{i,j} - \mathbf{U}^*_i \mathbf{V}^*_j)^2 + \lambda_1 \mathbf{U}^*_i \mathbf{U}^*_i + \lambda_2 [\mathbf{V}^*_j]^T \mathbf{V}^*_j \right]
\]

s.t. : \( \mathbf{V}^p - \overline{\mathbf{V}} = 0; \quad \forall p \in \{1, 2, ..., P\} \)

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:

\[ L^p(U, V^p, \Theta^p, \overline{V}) = f^p(U, V^p) + l^p(V^p, \overline{V}, \Theta^p) \]
\[ = \sum_{(i,j) \in \Omega^p} \hat{f}_{i,j}(U_{*i}, V^p_{*j}) \]
\[ + \left[ \frac{\rho}{2} \| V^p - \overline{V} \|^2_F + \text{tr}(\Theta^p T (V^p - \overline{V})) \right], \]

we can get

\[ L(U, V, \Theta, \overline{V}) = \sum_{p=1}^P L^p(U, V^p, \Theta^p, \overline{V}). \]
Distributed ADMM

Get the solutions by repeating the following three steps:

\[ U_{t+1}, V^p_{t+1} \leftarrow \arg\min_{U, V^p} L^p(U, V^p, \Theta^p_t, \overline{V}_t), \forall p \in \{1, 2, ..., P\} \]

\[ \overline{V}_{t+1} \leftarrow \arg\min_{\overline{V}} L(U_{t+1}, V_{t+1}, \Theta_t, \overline{V}) \]

\[ \Theta^p_{t+1} \leftarrow \Theta^p_t + \rho(V^p_{t+1} - \overline{V}_{t+1}), \forall p \in \{1, 2, ..., P\}. \]

The solution for \( \overline{V}_{t+1} \) is:

\[ \overline{V}_{t+1} = \frac{1}{P} \sum_{p=1}^{P} V^p_{t+1} \]

which can be calculated efficiently.

The problem lies in getting \( U_{t+1}, V_{t+1} \) efficiently.
Batch learning is still not very efficient:

\[ U_{t+1} = U_t - \tau_t \nabla_{U}^T f_p(U_t, V_t^p), \]
\[ V_{t+1}^p = \frac{\tau_t}{1 + \rho \tau_t} \left[ \frac{V_t^p}{\tau_t} + \rho V_t - \Theta_t^p - \nabla_{V_p}^T f_p(U_t, V_t^p) \right] \]

Stochastic Learning

\[ (U_{*_i})_{t+1} = (U_{*_i})_t + \tau_t (\epsilon_{ij}(V_{*_j})_t - \lambda_1(U_{*_i})_t), \]
\[ (V_{*_j})_{t+1} = \frac{\tau_t}{1 + \rho \tau_t} \left[ \frac{1 - \lambda_2 \tau_t}{\tau_t} (V_{*_j})_t \right. \]
\[ + \epsilon_{ij}(U_{*_i})_t + \rho (V_{*_j})_t - (\Theta_{*_j})_t \],

where \( \epsilon_{ij} = R_{ij} - [(U_{*_i})_t]^T (V_{*_j})_t \).
Scheduler Comparison

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

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

(a) CCD++, total synchronizes $k$ times
(b) DSGD, totally synchronizes $P$ times
(c) DS-ADMM, totally 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:

\[
\text{test RMSE} : \frac{1}{Q} \sqrt{\sum (R_{i,j} - U_{i}^{T}V_{j})^2}
\]
## Data Sets

**Table:** Data sets and parameter settings

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

(a) Netflix

(b) Yahoo-Music-R1

(c) Yahoo-Music-R2

(d) Time to fixed-RMSE (0.922)
Speedup

<table>
<thead>
<tr>
<th>Node Number</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

- CCD++
- DSGD
- DS−ADMM
- Linear−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, Spark, Parameter Server, 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]


Thanks!