# A parallel and efficient algorithm for learning to match

Jingbo Shang, Tianqi Chen, Hang Li,
Zhengdong Lu, Yong Yu

**Presented by Jingbo Shang**

**University of Illinois at Urbana-Champaign**

# Outline

- What is learning to match?
- What is the state-of-the-art algorithm for learning to match?
- How to accelerate it?
- How to fully parallelize it?

# Learning to Match

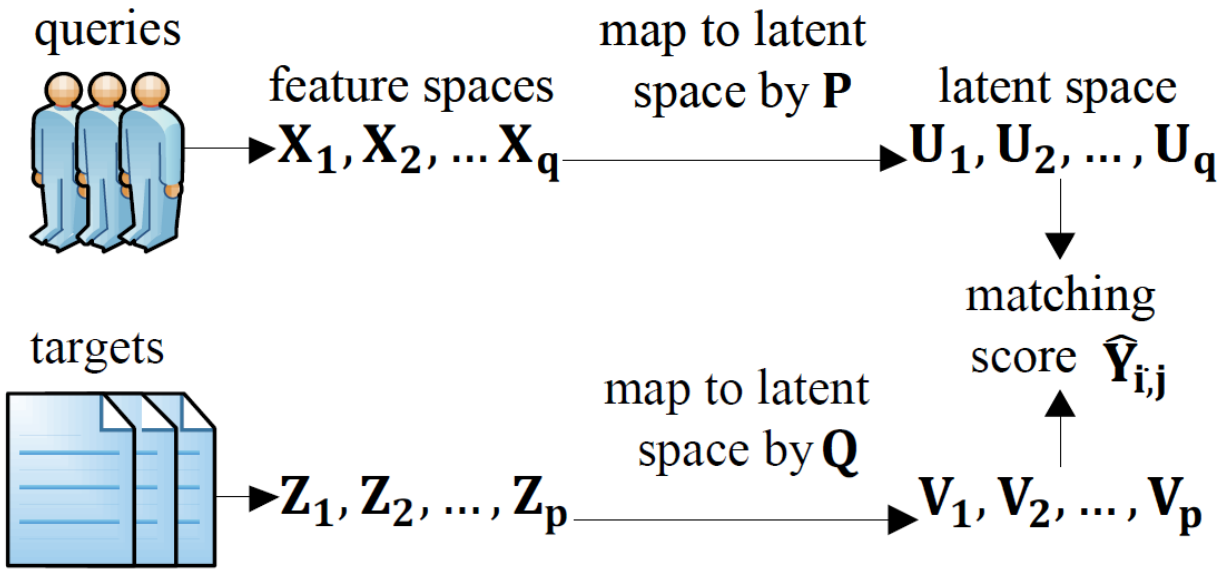
- Queries v.s. Targets
  - Heterogeneous space, need projection
- Instances
  - query domain: $\boldsymbol{X} = [\boldsymbol{X_1}, \boldsymbol{X_2}, \ldots, \boldsymbol{X_q}]$
  - target domain: $\boldsymbol{Z} = [\boldsymbol{Z_1}, \boldsymbol{Z_2}, \ldots, \boldsymbol{Z_p}]$
- Given some observations $\mathcal{O}$
- Learn to predict the matching score
  - $\hat{Y}_{ij} = f(\boldsymbol{X_i}, \boldsymbol{Z_j})$

# Feature-based Matrix Factorization

- One of the state-of-the-art algorithms
- Collaborative Filtering
  - KDD Cup 2011 Yahoo! Music
- Social Link Prediction
  - KDD Cup 2012 Tencent Weibo
- Web Search, Image Tagging, ……

# Feature-based Matrix Factorization

- The model
  - $\hat{Y}_{ij} = \boldsymbol{U_i^T V_j}$
  - $\boldsymbol{U_i = P X_i}$
  - $\boldsymbol{V_j = Q Z_j}$




  - $\boldsymbol{P}$ and $\boldsymbol{Q}$ are parameters to learn
  - Plain MF is a special case of this model

# Objective and Regularization

- Any differentiable convex loss for $l(Y_{ij}, \hat{Y}_{ij})$
  - Square
  - Logistic
- Elastic Net
  - Both L1 and L2
  - In experiments, we fix L1 = 0.1 and L2 = 1

# Acceleration

- Key idea
  - Avoid repeated calculations
  - Introduce auxiliary variables

# Acceleration

**Algorithm 1:** Coordinate Descent Algorithm for Learning to Match

randomly initialize $\mathbf{P}, \mathbf{Q}$;
$\mathbf{U} \leftarrow \mathbf{PX}$, $\mathbf{V} \leftarrow \mathbf{QZ}$, $\hat{\mathbf{Y}} \leftarrow \mathbf{U}^T\mathbf{V}$ {calculate $\mathbf{U}$, $\mathbf{V}$, $\hat{\mathbf{Y}}$}
**while** *not converge* **do**
  **for** $k = 1$ *to* $d$ **do**
    **for** $s = 1$ *to* $n$ **do**
      $x \leftarrow \sum_{i,j \in \mathcal{O}} g_{ij} \mathbf{V}_{kj} \mathbf{X}_{is}$
      $y \leftarrow \beta \sum_{i,j \in \mathcal{O}} \mathbf{V}_{kj}^2 \mathbf{X}_{is}^2$
      $\Delta \mathbf{P}_{ks} \leftarrow T(x, y, \mathbf{P}_{ks}, \alpha, \lambda)$
      $\mathbf{P}_{ks} \leftarrow \mathbf{P}_{ks} + \Delta \mathbf{P}_{ks}$
    **end**
  **end**
  $\mathbf{U} = \mathbf{PX}$ {recalculate buffered $\mathbf{U}$}
  update $\mathbf{Q}$ in the same way as $\mathbf{P}$
**end**

**Algorithm 2:** Efficient Algorithm for Learning to Match

randomly initialize $\mathbf{P}, \mathbf{Q}$
$\mathbf{U} \leftarrow \mathbf{PX}$, $\mathbf{V} \leftarrow \mathbf{QZ}$, $\hat{\mathbf{Y}} \leftarrow \mathbf{U}^T\mathbf{V}$ {calculate $\mathbf{U}$, $\mathbf{V}$, $\hat{\mathbf{Y}}$}
**while** *not converge* **do**
  **for** $k = 1$ *to* $d$ **do**
    **for** $i = 1$ *to* $q$ **do**
      $G_{ki} \leftarrow \sum_{j \in \mathcal{O}_i} g_{ij} \mathbf{V}_{kj}$
      $H_{ki} \leftarrow \beta \sum_{j \in \mathcal{O}_i} \mathbf{V}_{kj}^2$
    **end**
    **for** $s = 1$ *to* $n$ **do**
      $x \leftarrow \sum_i G_{ki} \mathbf{X}_{is}$
      $y \leftarrow \sum_i H_{ki} \mathbf{X}_{is}^2$
      $\Delta \mathbf{P}_{ks} \leftarrow T(x, y, \mathbf{P}_{ks}, \alpha, \lambda)$
      $\mathbf{P}_{ks} \leftarrow \mathbf{P}_{ks} + \Delta \mathbf{P}_{ks}$
      **for** $i = 1$ *to* $q$ **do**
        $G_{ki} \leftarrow G_{ki} + \mathbf{X}_{is} \Delta \mathbf{P}_{ks} H_{ki}$
      **end**
    **end**
  **end**
  $\mathbf{U} = \mathbf{PX}$ {recalculate buffered $\mathbf{U}$}
  update $\mathbf{Q}$ in the same way as $\mathbf{P}$
**end**

- $g_{ij} = \frac{\partial}{\partial \hat{Y}_{ij}} l(\hat{Y}_{ij}, Y_{ij})$
- $\beta = \sup_y \frac{\partial^2}{\partial y^2} l(y, Y_{ij})$
- $G_{ki} = \sum_{j \in \mathcal{O}_i} g_{ij} \boldsymbol{V}_{kj}$
- $H_{ki} = \beta \sum_{j \in \mathcal{O}_i} V_{kj}^2$

# Acceleration

$$\begin{aligned}\text{speedup} =& \frac{O(d|\mathcal{O}|(\frac{\|\mathbf{X}\|_0}{q} + \frac{\|\mathbf{Z}\|_0}{p}))}{O(d(\|\mathbf{X}\|_0 + \|\mathbf{Z}\|_0 + |\mathcal{O}|))} \\ \geq & min\{O(\frac{|\mathcal{O}|}{q}), O(\frac{|\mathcal{O}|}{p})\} \\ = & O(|\mathcal{O}|/(q+p))\end{aligned}$$

- In application tasks, this can be at level of 10 to 100
- When $|\boldsymbol{X}|_0 + |\boldsymbol{Z}|_0$ is close to (or smaller than) $|\mathcal{O}|$ , it is as fast as plain matrix factorization even though it uses extra features

# Acceleration

- Experimental Results:
  - Less time per round
  - Less round to achieve best performance

**Table 4: PL2M versus Hogwild! (8 threads)**

| Dataset | Method | sec/round | rounds | test error |
|---|---|---|---|---|
| Tencent Weibo (MAP@1) | Hogwild! | 104.3 | 14 | 24.96% |
| | PL2M | 70.1 | 5 | 25.52% |
| Flickr (MAP) | Hogwild! | 1117.0 | 59 | 7.59% |
| | PL2M | 155.0 | 33 | 7.59% |
| Movielens-10M (RMSE) | Hogwild! | 162.2 | 20 | 0.8756 |
| | PL2M | 18.5 | 7 | 0.8666 |

# Parallelization

- Key ideas
  - Iteratively relax the objective function to an upper bound
  - $abxy \leq \frac{1}{2}|xy|(a^2 + b^2)$
- Theoretical guarantee on convergence
- $K$ threads
  - speedup = $O(K)$

# Parallelization

- Update $\boldsymbol{P}_{ks}$ for $s \in S$ in parallel
  - Non-zero cross terms $X_{it}X_{is}$ will lead to conflict writes

$$\tilde{L}(\Delta \mathbf{P}_{k,:}) = \sum_{i,j \in \mathcal{O}} l(\hat{\mathbf{Y}}_{ij}, \mathbf{Y}_{ij})$$

$$+ \sum_{s \in S} \left( \sum_{i} G_{ki} \mathbf{X}_{is} \Delta \mathbf{P}_{ks} + \frac{1}{2} \sum_{i} H_{ki} \mathbf{X}_{is}^2 \Delta \mathbf{P}_{ks}^2 \right)$$

$$+ \sum_{s \in S} \sum_{t \in S, t \neq s} \sum_{i} H_{ki} \mathbf{X}_{it} \mathbf{X}_{is} \Delta \mathbf{P}_{kt} \Delta \mathbf{P}_{ks}$$

# Parallelization

- Relax it to an upper bound

$$\tilde{L}(\Delta\mathbf{P}_{k,:}) \leq c + \sum_{s\in S}(\sum_{i} G_{ki}\mathbf{X}_{is}\Delta\mathbf{P}_{ks})$$

$$+ \sum_{s\in S}(\frac{1}{2}\sum_{i} H_{ki}|\mathbf{X}_{is}|(\sum_{t\in S}|\mathbf{X}_{it}|)\Delta\mathbf{P}_{ks}^2)$$

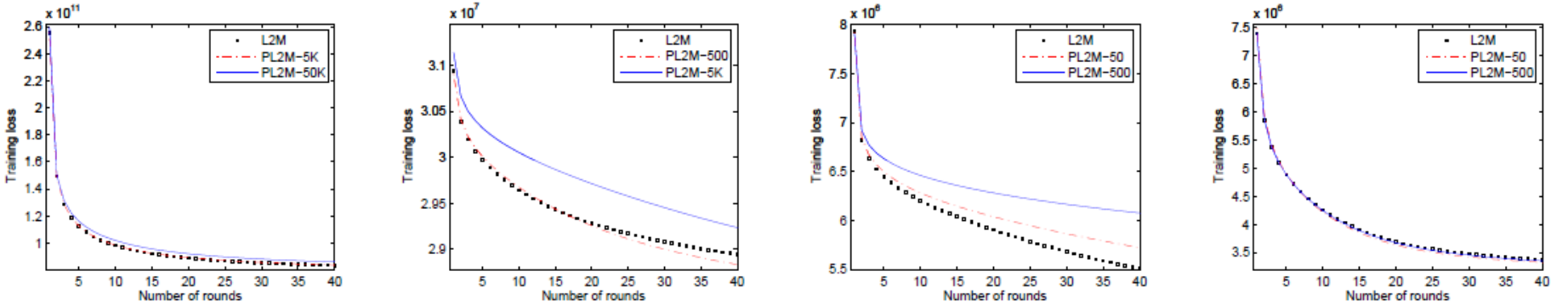
$$\triangleq \tilde{L}^p(\Delta\mathbf{P}_{k,:}).$$

**Algorithm 3:** Parallel Algorithm for Learning to Match

randomly initialize $\mathbf{P}, \mathbf{Q}$
$\mathbf{U} \leftarrow \mathbf{PX}$, $\mathbf{V} \leftarrow \mathbf{QZ}$, $\hat{\mathbf{Y}} \leftarrow \mathbf{U}^T\mathbf{V}$ {calculate $\mathbf{U}$, $\mathbf{V}$, $\hat{\mathbf{Y}}$}
**while** *not converge* **do**
  schedule a partition $P$ of $\{1, 2, \cdots, n\}$
  **for** $k = 1$ *to* $d$ **do**
    **for** $i = 1$ *to* $q$ *in parallel* **do**
      $G_{ki} \leftarrow \sum_{j \in \mathcal{O}_i} g_{ij} \mathbf{V}_{kj}$
      $H_{ki} \leftarrow \beta \sum_{j \in \mathcal{O}_i} \mathbf{V}_{kj}^2$
    **end**
    **for** *each index set* $S \in P$ **do**
      **for** $i = 1$ *to* $q$ *in parallel* **do**
        $C_{ki} \leftarrow \sum_{s \in S} |\mathbf{X}_{is}|$
      **end**
      **for** $s \in S$ *in parallel* **do**
        $x \leftarrow \sum_i G_{ki} \mathbf{X}_{is}$
        $y \leftarrow \sum_i H_{ki} |\mathbf{X}_{is}| C_{ki}$
        $\Delta \mathbf{P}_{ks} \leftarrow T(x, y, \mathbf{P}_{ks}, \alpha, \lambda)$
        $\mathbf{P}_{ks} \leftarrow \mathbf{P}_{ks} + \Delta \mathbf{P}_{ks}$
      **end**
      **for** $i = 1$ *to* $q$ *in parallel* **do**
        $G_{ki} \leftarrow G_{ki} + \sum_{s \in S} \mathbf{X}_{is} \Delta \mathbf{P}_{ks} H_{ki}$
      **end**
    **end**
  **end**
  $\mathbf{U} = \mathbf{PX}$ {recalculate buffered $\mathbf{U}$}
  update $\mathbf{Q}$ in the same way as $\mathbf{P}$
**end**

# Parallelization

- Training Loss Convergence

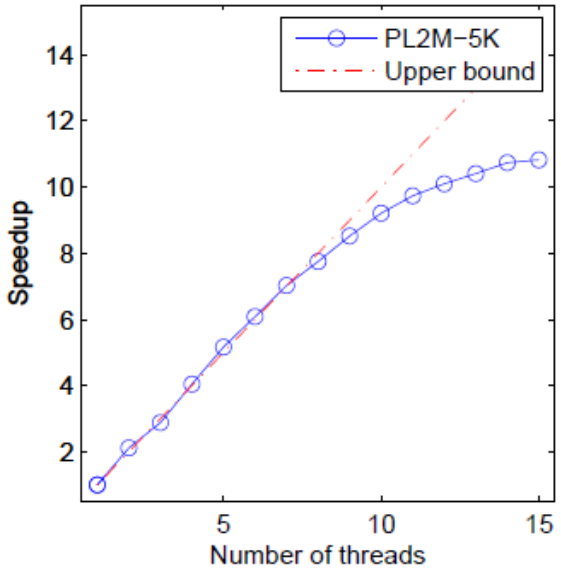



(a) TLC on Yahoo! Music (b) TLC on Tencent Weibo (c) TLC on Flickr (d) TLC on Movielens-10M

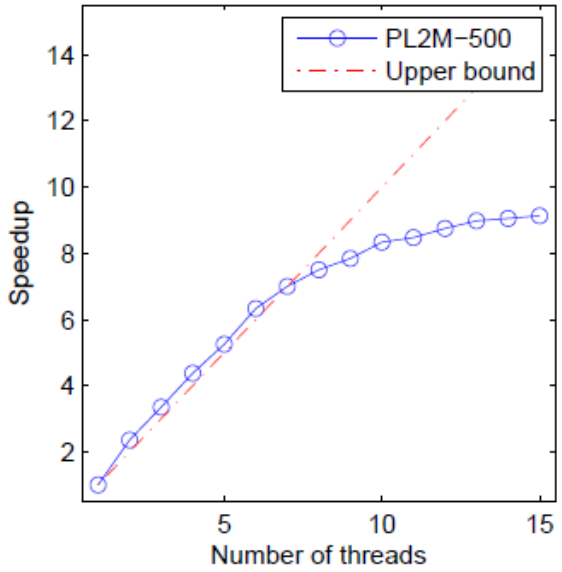
**Figure 2: Training Loss Convergence(TLC) on Four Datasets**

# Parallelization

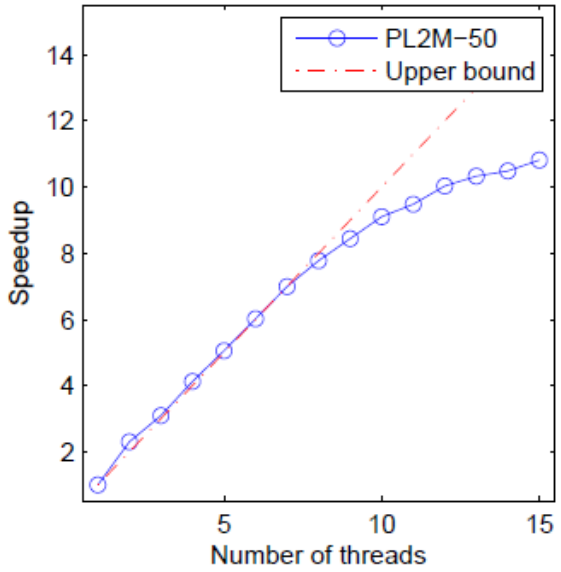
- Speedup Curves
  - 8 cores, 16 threads



(a) on Yahoo! Music



(b) on Tencent Weibo



(c) on Flickr

# Toolkit

- On my github
  - ID: shangjingbo1226
  - REPO: PL2M
- Keep update
  - tested on linux (ubuntu)
  - will provide support for MacOS and windows

# Future Work

- Scheduling methods for the feature set $S$
  - Currently, pure random shuffle
- Distributed Version

# Thank you!

Q&A