Collaborative Ranking from Pairwise Comparisons

Dohyung Park and Jin Zhang

Scalable Machine Learning - Course Project

December 15, 2014

Collaborative Ranking from Pairwise Comparison?

Recommendation systems

- Collect feedback from users on some items, and then recommend.
- Need to find the items *highly ranked* in each user's preference order.

Collaborative ranking?

- Predict each user's preference order (*ranking*) of the items.
- More flexible than rating

Implicit feedback

- Predominant in practice (e.g., Users' choices, Click numbers, etc.)
- One basic form of implicit feedback : Pairwise comparisons

Overview

1. Formulation
2. Stochastic Gradient Descent
   - Parallelization with random sampling
   - Parallelization using NOMAD approach
3. Alternating rankSVM
   - Parallelization via graph partitioning
4. Graph partitioning
5. Experiments

Formulation

Matrix factorization approach

$$\text{minimize}_{U,V} \sum_{(i,j,k)\in\Omega} \mathcal{L}(U_i(V_j - V_k)^\top) + \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2)$$

- $\Omega = \{(i,j,k) : i \in [m], j,k \in [n]\}$: A given set of comparisons
  "*User i prefers item j to item k*"
- $U \in \mathbb{R}^{m\times r}$: (latent) User features
- $V \in \mathbb{R}^{n\times r}$: (latent) Item features
- $\mathcal{L}(U_i(V_j - V_k)^\top)$: Loss function inducing $U_i V_j^\top > U_i V_k^\top$ (e.g., Hinge loss)

Stochastic Gradient Descent

1. **Write the objective function as a sum of (sparse) functions**

$$f(x) = \sum_{c \in \mathcal{C}} f_c(x_c) \quad (c : \text{a subset of the coordinates})$$

2. **Take one sample $c \in \mathcal{C}$, and update x_c.**

$$x_c \leftarrow x_c - \gamma \nabla f_c(x_c)$$

1. Write the objective function as a sum of (sparse) functions

$$\text{minimize}_{U,V} \quad \sum_{(i,j,k)\in\Omega} \mathcal{L}(U_i(V_j - V_k)^\top) + \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2)$$

Equivalently, we can write

$$\text{minimize}_{U,V}$$
$$\sum_{(i,j,k)\in\Omega} \left\{ \mathcal{L}(U_i(V_j - V_k)^\top) + \frac{\lambda}{2|\Omega_i|}\|U_i\|_2^2 + \frac{\lambda}{2|\Omega^j|}\|V_j\|_2^2 + \frac{\lambda}{2|\Omega^k|}\|V_k\|_2^2 \right\}$$

Stochastic Gradient Descent

2. Take one sample $(i, j, k) \in \Omega$, and update U_i, V_j, V_k.

$$\text{minimize}_{U,V} \sum_{(i,j,k)\in\Omega} \left\{ \mathcal{L}(U_i(V_j - V_k)^\top) + \frac{\lambda}{2|\Omega_i|}\|U_i\|_2^2 + \frac{\lambda}{2|\Omega^j|}\|V_j\|_2^2 + \frac{\lambda}{2|\Omega^k|}\|V_k\|_2^2 \right\}$$

An update for (i, j, k) will be

$$U_i \leftarrow U_i - \gamma \cdot \left(\mathcal{L}'(U_i(V_j - V_k)^\top) \cdot (V_j - V_k) + \frac{\lambda}{2|\Omega_i|} U_i \right)$$
$$V_j \leftarrow V_j - \gamma \cdot \left(\mathcal{L}'(U_i(V_j - V_k)^\top) \cdot U_i + \frac{\lambda}{2|\Omega_j|} V_j \right)$$
$$V_k \leftarrow V_k - \gamma \cdot \left(-\mathcal{L}'(U_i(V_j - V_k)^\top) \cdot U_i + \frac{\lambda}{2|\Omega_k|} V_k \right)$$

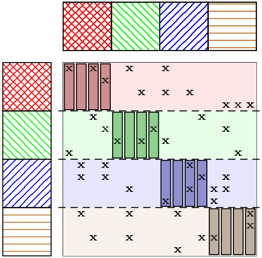
Parallelization with random sampling (and without locking)

In each step, we update only three of the $(m+n)$ feature vectors.

Simple random sampling for each processor will work. [Niu et al, 2011]

$$E[\#\text{conflicts}] = O\left(\frac{p^2}{m+n}\right)$$

Parallelization using NOMAD approach

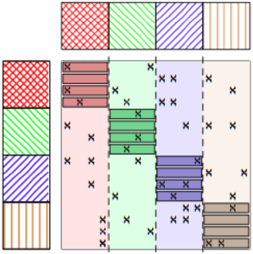


The original NOMAD [Yun et al, 2014]

- Partition the users
- Each item vector cycles over processors

In our problem..

- The item vectors can't move independently
- Partition the items (using graph partitioning)
- Each user vector cycles over processors



Alternating rankSVM

Our formulation

$$\text{minimize}_{U,V} \sum_{(i,j,k)\in\Omega} \mathcal{L}(U_i(V_j - V_k)^\top) + \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2)$$

Alternating minimization?

- Solving for U

$$\text{minimize}_U \ \frac{1}{2}\|U\|_F^2 + \frac{1}{\lambda} \sum_{(i,j,k)\in\Omega} \mathcal{L}(U_i(V_j - V_k)^\top)$$

- Solving for V

$$\text{minimize}_V \ \frac{1}{2}\|V\|_F^2 + \frac{1}{\lambda} \sum_{(i,j,k)\in\Omega} \mathcal{L}(U_i(V_j - V_k)^\top)$$

- Both subproblems are linear SVMs
- We can use Dual Coordinate Descent [Hsieh et al, 2008] for each subproblem

Parallelization

Parallelization for the U part is easy

- Can solve for each U_i

$$U_i \leftarrow \operatorname{argmin} \frac{1}{2}\|U_i\|_2^2 + \frac{1}{\lambda} \sum_{j,k:(i,j,k)\in\Omega} \mathcal{L}(U_i(V_j - V_k)^\top)$$

Parallelization for the V part is hard

- Cannot decompose the subproblem

$$\text{minimize}_V \; \frac{1}{2}\|V\|_F^2 + \frac{1}{\lambda} \sum_{(i,j,k)\in\Omega} \mathcal{L}(U_i(V_j - V_k)^\top)$$

A single dual coordinate descent step updates both V_j and V_k

Use graph partitioning

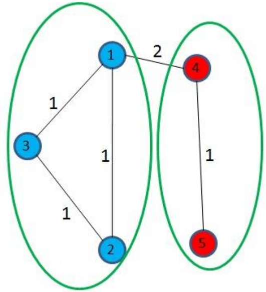
Graph partitioning

Partition the item graph

- Minimize edge cut
- Balance edge weight

Graclus with normalized cut

$$NCut(G) = \min_{\mathcal{V}_1,\ldots,\mathcal{V}_k} \sum_{c=1}^{k} \frac{\mathrm{link}(\mathcal{V}_c, \mathcal{V}/\mathcal{V}_c)}{\mathrm{degree}(\mathcal{V}_c)}$$



(user, item1, item2)
(1, 1, 2)
(1, 3, 1)
(1, 3, 2)
(1, 4, 1)
(1, 5, 4)
(2, 1, 4)

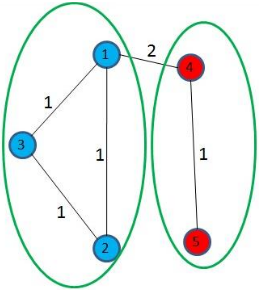
Graph partitioning

Partition the item graph

- Minimize edge cut
- Balance edge weight

Graclus with normalized cut [Dhillon et al, 2007]

$$NCut(G) = \min_{\mathcal{V}_1,\ldots,\mathcal{V}_k} \sum_{c=1}^{k} \frac{\mathrm{link}(\mathcal{V}_c, \mathcal{V}/\mathcal{V}_c)}{\mathrm{degree}(\mathcal{V}_c)}$$



(user, item1, item2)
(1, 1, 2)
(1, 3, 1)
(1, 3, 2)
(1, 4, 1)
(1, 5, 4)
(2, 1, 4)

Experiment

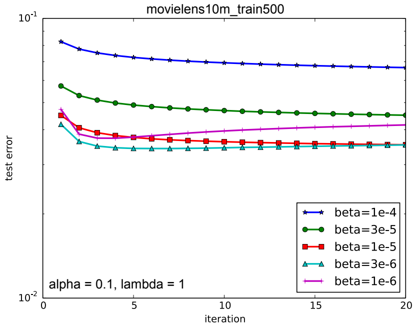
- Dataset

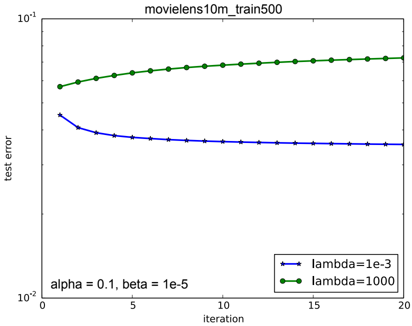
Dataset	MovieLens10m	Netflix
# Users	71k	480k
# Movies	10k	17k
# Ratings	10m	100m
# (Extracted) Comparisons	19m	150m

- Metrics
  - Test error (3.8m and 30m extracted comparisons)
  - Computational time
  - Scalability

Experiment

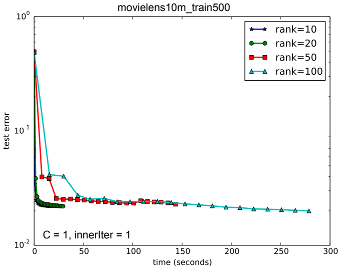
- Parameter tuning for SGD

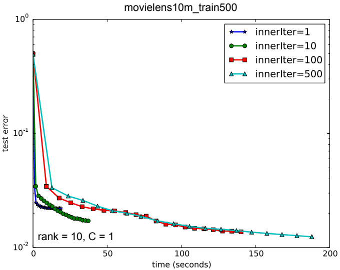


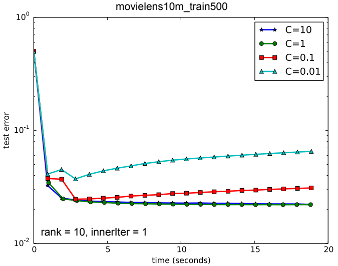


Experiment

- Parameter tuning for Alt-rankSVM

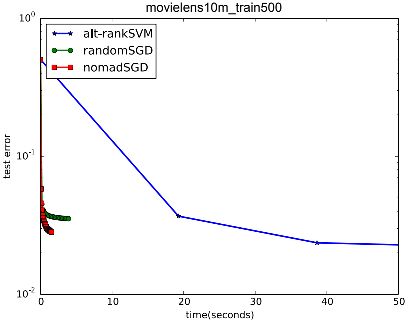


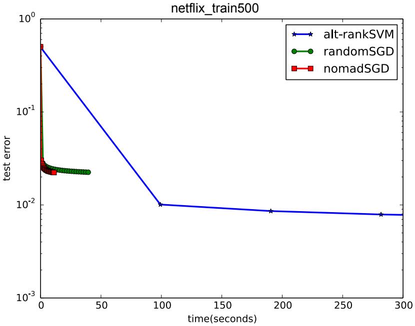




Experiment

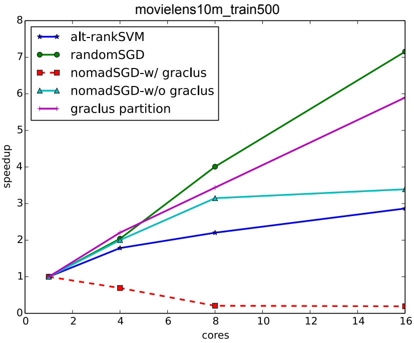
- Convergence speed

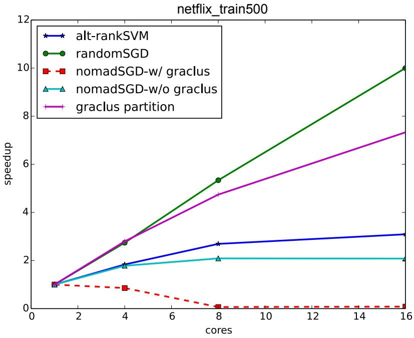




Experiment

- Scalability





Conclusion

- All three algorithms produces low test error (< 0.02)
- SGD converges much faster than Alt-rankSVM
- Random SGD shows linear speedup

Future work

- Improve the parallelization
- How about the other metric? (e.g., NDCG)
- Comparison with matrix completion algorithms (e.g., ALS)
- Kernelization?

References

F. Niu, B. Recht, C. Re, and S. Wright,
Hogwild!: A lock-free approach to parallelized stochatic gradient descent
NIPS, 2011

C.-J. Hsieh, K.-W. Chiang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan,
A Dual Coordinate Descent Method for Large-scale Linear SVM
ICML, 2008

H. Yun, H.-F. Yu, C.-J. Hsieh, S. V. N. Vishwanathan, and I. Dhillon,
NOMAD: Non-locking, stOchastic Multi-machine algorithm for Asynchronous and Decentralized matrix completion
VLDB, 2014

I. Dhillon, Y. Guan, and B. Kulis,
Weighted Graph Cuts without Eigenvectors: A Multilevel Approach
IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 2007

The End