Scalable Machine Learning - Final Project Report: Collaborative Ranking from Pairwise Comparisons

Dohyung Park and Jin Zhang

December 13, 2014

Abstract

We consider the problem of ranking multiple items for each of multiple users when a set of pairwise comparisons of the users is given. To formulate the problem, we estimate a low-rank matrix that can represent the ranking. We propose to use two algorithms, stochastic gradient descent and alternating rank-SVM. We also provide parallel implementations for these two algorithms, which use either random sampling or graph partitioning. Performance and scalability of the implementations are evaluated for the comparison sets extracted from the public datasets with user-item ratings.

1 Introduction

Recommendation systems have been one of the main problems in machine learning. As it has become popular through the well known *netflix prize* and the mathmatical problem of matrix completion [1], the standard formulation has been such that one estimate the numerical ratings of items by users when the users have rated a subset of the items. However, the fundamental goal of recommendation systems is to estimate *which items would be more preferred to the others for each user*. In this sense, ranking items for each user from given information can be a more appropriate formulation of recommendation systems.

Moreover, a user's preferences over items are often observed *implicitly* in many applications of recommendation systems. For example, when a user selects an item from the list of items relevant to the user's query, this implies that the user prefers the item to the others. The number of clicks on an item can imply how the user is interested in the item. While most academic work has studied recommendation using explicit feedback, e.g., rating data, implicit feedback is predominant in practice.

In this project, we estimate the rankings for each of multiple users from given a set of pairwise comparisons, which is a natural form of implicit feedback. We formulate an empirical risk minimization framework with the matrix factorization approach. We propose to use two natual algorithms for this setting, stochastic gradient descent and alternating rank-SVM. We discuss how to parallelize the two algorithms, and test the implementations on real-world datasets.

2 Problem formulation

Suppose there are n users and m items. We are given a set of $|\Omega|$ pairwise comparisons, $\Omega \in ([n]\times[m]\times[m])^{|\Omega|}$, where user i prefers item j_1 to item j_2 if $(i, j_1, j_2) \in \Omega$. We denote by Ω_i the set of comparisons of user i, i.e., $\Omega_i = \{(i, j_1, j_2) : (i, j_1, j_2) \in \Omega\}$. We also denote by Ω^j the set of comparisons associated with item j, i.e., $\Omega^j = \{(i, j, j_2) : (i, j, j_2) \in \Omega\} \cup \{(i, j_1, j) : (i, j_1, j) \in \Omega\}$

We want to estimate the ranking of each user based on the low-rank matrix model $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{V}^\top$ ($\boldsymbol{U} \in \mathbb{R}^{n\times r}, \boldsymbol{V} \in \mathbb{R}^{m\times r}$) in which $X_{ij_1} > X_{ij_2}$ implies that user i prefers item j_1 to item j_2. We denote by $\boldsymbol{u}_i^\top$ and $\boldsymbol{v}_i^\top$ the ith rows of $\boldsymbol{U}$ and $\boldsymbol{V}$, respectively.

2.1 Matrix factorization approach

Similarly to the collaborative rating, this problem can be stated as

$$\underset{\boldsymbol{U}\in\mathbb{R}^{n\times r},\boldsymbol{V}\in\mathbb{R}^{m\times r}}{\text{minimize}} \sum_{(i,j_1,j_2)\in\Omega} \mathcal{L}(\boldsymbol{u}_i^\top(\boldsymbol{v}_{j_1}-\boldsymbol{v}_{j_2})) + \frac{\lambda}{2}(\|\boldsymbol{U}\|_F^2+\|\boldsymbol{V}\|_F^2) \tag{1}$$

where $\mathcal{L}(\cdot)$ is a monotonically nondecreasing loss function which induces positivity (e.g., hinge loss, logistic regression loss, etc.) For the rest of the report, we mainly take the squared hinge loss as our loss function, i.e., $\mathcal{L}(x)=\max(1-x,0)^2$.

3 Stochastic Gradient Descent (SGD)

As the loss function for each training example (one pairwise comparison of a user) is a sparse cost function, stochastic gradient descent can be a good option for solving (1). First we can write (1) as

$$\underset{\boldsymbol{U}\in\mathbb{R}^{n\times r},\boldsymbol{V}\in\mathbb{R}^{m\times r}}{\text{minimize}} \sum_{(i,j_1,j_2)\in\Omega} \left\{\mathcal{L}(\boldsymbol{u}_i^\top(\boldsymbol{v}_{j_1}-\boldsymbol{v}_{j_2})) + \frac{\lambda}{2}\left(\frac{1}{|\Omega_i|}\|\boldsymbol{u}_i\|_2^2+\frac{1}{|\Omega^{j_1}|}\|\boldsymbol{v}_{j_1}\|_2^2+\frac{1}{|\Omega^{j_2}|}\|\boldsymbol{v}_{j_2}\|_2^2\right)\right\}$$

Since each component function is associated with $\boldsymbol{u}_i, \boldsymbol{v}_{j_1}, \boldsymbol{v}_{j_2}$ for some $(i,j_1,j_2)\in\Omega$, one SGD step for the component will update only those parts.

For each iteration, one example (i,j_1,j_2) is randomly chosen and update $\boldsymbol{U}$ and $\boldsymbol{V}$ as follows.

$$\begin{aligned}
\boldsymbol{u}_i^+ &\leftarrow \boldsymbol{u}_i - \eta\cdot\left\{g\cdot(\boldsymbol{v}_{j_1}-\boldsymbol{v}_{j_2})+\frac{\lambda}{|\Omega_i|}\boldsymbol{u}_i\right\}\\
\boldsymbol{v}_{j_1}^+ &\leftarrow \boldsymbol{v}_{j_1} - \eta\cdot\left\{g\cdot\boldsymbol{u}_i+\frac{\lambda}{|\Omega^{j_1}|}\boldsymbol{v}_{j_1}\right\}\\
\boldsymbol{v}_{j_2}^+ &\leftarrow \boldsymbol{v}_{j_2} - \eta\cdot\left\{-g\cdot\boldsymbol{u}_i+\frac{\lambda}{|\Omega^{j_2}|}\boldsymbol{v}_{j_2}\right\}
\end{aligned}$$

η is a step size and $g\in\partial\mathcal{L}(\boldsymbol{u}_i^\top(\boldsymbol{v}_{j_1}-\boldsymbol{v}_{j_2}))$. When $\mathcal{L}(\cdot)$ is the squared hinge loss, g is given by

$$g=\begin{cases} -2(1-\boldsymbol{u}_i^\top(\boldsymbol{v}_{j_1}-\boldsymbol{v}_{j_2})) & \text{if } \boldsymbol{u}_i^\top(\boldsymbol{v}_{j_1}-\boldsymbol{v}_{j_2})<1,\\ 0 & \text{if } \boldsymbol{u}_i^\top(\boldsymbol{v}_{j_1}-\boldsymbol{v}_{j_2})\ge 1.\end{cases}$$

4 Alternating RankSVM (AltSVM)

The alternating minimization of (1) can be stated as follows. We alternately fix $\boldsymbol{U}$ and $\boldsymbol{V}$, and minimize (1) over each other. Solving for $\boldsymbol{U}$ (user subproblem) can be written as

$$\boldsymbol{u}_i \leftarrow \arg\min_{\boldsymbol{u}\in\mathbb{R}^r}\left\{\frac{1}{2}\|\boldsymbol{u}\|_2^2 + C\sum_{j_1,j_2:(i,j_1,j_2)\in\Omega}\mathcal{L}(\boldsymbol{u}^\top(\boldsymbol{v}_{j_1}-\boldsymbol{v}_{j_2}))\right\} \tag{2}$$

for every $i\in[n]$ where $C=\frac{1}{\lambda}$. Note that this subproblem can be independently solved for each user. This is exactly the ranking SVM algorithm [4]. Solving for V (item subproblem) can be written as

$$\boldsymbol{V} \leftarrow \arg\min_{\boldsymbol{V}\in\mathbb{R}^{m\times r}}\left\{\frac{1}{2}\|\boldsymbol{V}\|_F^2 + C\sum_{i,j_1,j_2:(i,j_1,j_2)\in\Omega}\mathcal{L}(\langle\boldsymbol{V},\boldsymbol{A}^{(i,j_1,j_2)}\rangle)\right\} \tag{3}$$

where

$$(\boldsymbol{A}^{(i,j_1,j_2)})_j = \begin{cases} \boldsymbol{u}_i^\top, & j = j_1 \\ -\boldsymbol{u}_i^\top, & j = j_2 \\ \boldsymbol{0}, & \text{otherwise} \end{cases}$$

Note that this problem is also a ranking SVM formulation. As the subproblems can be solved using linear SVM, we adopt the dual coordinate descent for the both parts. [3]

4.1 Dual Coordinate Descent for SVM

We breifly describe the dual coordinate descent method for the linear SVM problem. For a given SVM problem (with the squared hinge loss)

$$\begin{aligned} &\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{minimize}} \ \frac{1}{2}\boldsymbol{w}^\top\boldsymbol{w} + C\sum_{i=1}^{n}\xi_i^2 \equiv g(\boldsymbol{w}) \\ &\text{subject to} \ \ y_i\boldsymbol{w_i}^\top\boldsymbol{x_i} \geq 1-\xi_i,\ \xi_i \geq 0,\ \forall i = 1,\ldots,n, \end{aligned}$$

the dual problem is stated as

$$\begin{aligned} &\underset{\boldsymbol{\alpha}\in\mathbb{R}^n}{\text{minimize}} \ \frac{1}{2}\boldsymbol{\alpha}^\top(Q+\frac{1}{2C}I)\boldsymbol{\alpha} - \boldsymbol{e}^\top\boldsymbol{\alpha} \equiv f(\boldsymbol{\alpha}) \\ &\text{subjec to} \ \ \alpha_i \geq 0, \forall i = 1,\ldots,n, \end{aligned}$$

where $Q_{ij} = y_i y_j \boldsymbol{x_i}^T\boldsymbol{x_j}$. If $\boldsymbol{\alpha}^*$ is the dual optimum, then $\boldsymbol{w}^* = \sum_{i=1}^{n}\alpha_i^* y_i \boldsymbol{x_i}$ is the primal optimum.

Suppose we maintain $\boldsymbol{w} = \sum_{i=1}^{n}\alpha_i y_i \boldsymbol{x}_i$. A dual coodinate descent step for α_i is:

$$\begin{aligned} \delta^* &= \max\left(0, \alpha_i + \frac{1 - \sum\limits_{j=1}^{n}\alpha_j Q_{ij} - \frac{\alpha_i}{2C}}{Q_{ii} + \frac{1}{2C}}\right) - \alpha_i \\ &= \max\left(0, \alpha_i + \frac{1 - y_i\boldsymbol{x}_i^\top\boldsymbol{w} - \frac{\alpha_i}{2C}}{\|\boldsymbol{x}_i\|_2^2 + \frac{1}{2C}}\right) - \alpha_i \\ \alpha_i &\leftarrow \alpha_i + \delta^* \end{aligned}$$

Then the update rule for $\boldsymbol{w}$ is:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \delta^* y_i \boldsymbol{x}_i$$

4.2 User Subproblem (2)

We will solve a SVM problem for each user i. $\boldsymbol{X(i)}$ denotes the difference matrix for user i such that:

$$X(i)_j = \boldsymbol{v}_{j_1} - \boldsymbol{v}_{j_2}$$

in which (i, j_1, j_2) is the jth entry in $\boldsymbol{\Omega_i}$

The SVM problem is:

$$\min_{\boldsymbol{u}\in R^r} \frac{1}{2}\boldsymbol{u}^\top\boldsymbol{u} + C\sum_{j=1}^{|\Omega_i|}\xi_j^2 \equiv g(\boldsymbol{u})$$

subject to $\boldsymbol{u}^\top\boldsymbol{X(i)_j} \geq 1-\xi_j$, $\xi_j \geq 0, \forall j = 1,\ldots,|\Omega_i|$.

The update rule for α_j is ($Q_{jk} = \boldsymbol{X(i)}_{\boldsymbol{j}}^{\top}\boldsymbol{X(i)}_{\boldsymbol{k}}$):

$$\delta^* = \max\left(0, \alpha_j + \frac{1 - u^{\top}X(i)_j - \frac{\alpha_j}{2C}}{\|X(i)_j\|_2^2 + \frac{1}{2C}}\right) - \alpha_j$$

Then

$$\boldsymbol{u}_i \leftarrow \boldsymbol{u}_i + \delta^*(\boldsymbol{v}_{j_1} - \boldsymbol{v}_{j_2}) \tag{4}$$

4.3 Item Subproblem (3)

The dual problem of (3) is written as

$$\begin{aligned} \underset{\boldsymbol{\alpha}\in\mathbb{R}^{|\Omega|}}{\text{minimize}} \quad & \frac{1}{2}\boldsymbol{\alpha}^{\top}(Q + \frac{1}{2C})\boldsymbol{\alpha} - e^{\top}\boldsymbol{\alpha} \\ \text{subject to} \quad & \alpha_{(i,j_1,j_2)} \geq 0, \quad \forall(i,j_1,j_2) \in \Omega \end{aligned} \tag{5}$$

where $\boldsymbol{\alpha}$ is a length-$|\Omega|$ dual vector, and $\alpha_{(i,j_1,j_2)}$ is a component of $\boldsymbol{\alpha}$ corresponding to the sample (i, j_1, j_2). $Q \in \mathbb{R}^{|\Omega|\times|\Omega|}$ is given by

$$\begin{aligned} Q_{(i,j_1,j_2),(i',j_1',j_2')} &= \langle \boldsymbol{A}^{(i,j_1,j_2)}, \boldsymbol{A}^{(i',j_1',j_2')} \rangle \\ &= (\boldsymbol{u}_i^{\top}\boldsymbol{u}_{i'}) \cdot (\mathbb{I}_{j_1=j_1'} + \mathbb{I}_{j_2=j_2'} - \mathbb{I}_{j_1=j_2'} - \mathbb{I}_{j_2=j_1'}) \end{aligned}$$

In the dual coordinate descent, we randomly pick (i, j_1, j_2) and update $\boldsymbol{v}_{j_1}$ and $\boldsymbol{v}_{j_2}$. Following the derivation in Section 3.2, we obtain

$$\begin{aligned} \delta^* &\leftarrow \max\left(0, \alpha_{(i,j_1,j_2)} + \frac{1 - \boldsymbol{u}_i^{\top}(\boldsymbol{v}_{j_1} - \boldsymbol{v}_{j_2})^{\top} - \frac{\alpha_{(i,j_1,j_2)}}{2C}}{2\|\boldsymbol{u}_i\|_2^2 + \frac{1}{2C}}\right) - \alpha_{(i,j_1,j_2)}, \\ \alpha_{(i,j_1,j_2)} &\leftarrow \alpha_{(i,j_1,j_2)} + \delta^*, \\ \boldsymbol{v}_{j_1} &\leftarrow \boldsymbol{v}_{j_1} + \delta^*\boldsymbol{u}_i, \\ \boldsymbol{v}_{j_2} &\leftarrow \boldsymbol{v}_{j_2} - \delta^*\boldsymbol{u}_i. \end{aligned} \tag{6}$$

5 Parallelization

The above two algorithms can be parallelized by giving each processor a subset of the comparison set Ω and letting it update the entries of U and V based on the subset. The scalability of the algorithms depend on how we divide the comparison set properly so that the updates of the processors are as independent as they can. To this end, we consider two strategies: random sampling which simply relies on the problem structure, and graph partitioning.

5.1 Parallel SGD using random sampling

It is demonstrated in [6] that if one SGD step updates a sparse subset of the parameters, then simply running SGD in parallel without locking is effective and scales well. In our problem, one SGD step updates only one of n user vectors and two of the m item vectors. Hence, parallelized SGD without locking will run effectively with few conflicts.

Algorithm 1 Parallel Stochastic Gradient Descent with random sampling

```
Input: $\Omega$ and a step size sequence $\{\eta_t\}_{t=1}^{\infty}$
Output: $\boldsymbol{U} \in \mathbb{R}^{n \times r}$, $\boldsymbol{V} \in \mathbb{R}^{m \times r}$
1: Initialize $\boldsymbol{U}, \boldsymbol{V}$ to uniform random variables
2: parfor $t = 1, \dots$ do
3:     Randomly sample $(i, j_1, j_2) \in \Omega$
4:     if $\boldsymbol{u}_i^\top(\boldsymbol{v}_{j_1} - \boldsymbol{v}_{j_2}) < 1$ then
5:         $\boldsymbol{u}_i \leftarrow (1 - \frac{\eta_t \lambda}{|\Omega_i|})\boldsymbol{u}_i - 2\eta_t(1 - \boldsymbol{u}_i^\top(\boldsymbol{v}_{j_1} - \boldsymbol{v}_{j_2}))(\boldsymbol{v}_{j_1} - \boldsymbol{v}_{j_2})$
6:         $\boldsymbol{v}_{j_1} \leftarrow (1 - \frac{\eta_t \lambda}{|\Omega^{j_1}|})\boldsymbol{v}_{j_1} - 2\eta_t(1 - \boldsymbol{u}_i^\top(\boldsymbol{v}_{j_1} - \boldsymbol{v}_{j_2}))\boldsymbol{u}_i$
7:         $\boldsymbol{v}_{j_2} \leftarrow (1 - \frac{\eta_t \lambda}{|\Omega^{j_2}|})\boldsymbol{v}_{j_2} + 2\eta_t(1 - \boldsymbol{u}_i^\top(\boldsymbol{v}_{j_1} - \boldsymbol{v}_{j_2}))\boldsymbol{u}_i$
8:     end if
9: end parfor
```

5.2 Parallel SGD using graph partitioning

In the SGD algorithm, we can think of two graphs from the comparison set.

- A hypergraph with hyperedges

  There are $n + m$ nodes in which n nodes represent the users, and the other m nodes represent the items. As one comparison contains one user and two items, each comparison can be regarded as a hyperedge connecting three correponding nodes. If we can parition the nodes perfectly so that there is no hyperedge between two groups, then the algorithm can be perfectly parallelized since the processors update completely disjoint subsets of parameters. Although there is no perfect partition of the hypergraph, we can minimize the number of hyperedges in the cut.

  Another aspect we need to consider other than the minimum cut is the edge balance in the parition. Since the size of a subset represents the number of comparisons that a processor computes, it is very important for the sizes of the subsets to be almost equal.

  The only public algorithm for partitioning hypergraphs is hMETIS [5]. This algorithm can find a parition with small cut well, but the resulting subsets are not balanced well. This is a critical problem in achieving scalability for our algorithm. Therefore, we did NOT use hypergraph partitioning.

- An item graph

  If we ignore the user indices, then we can simply construct an item graph, where m nodes represent the items, and two nodes are connected if there is a comparison of the corresponding items by some user. For the general graph, we can obtain a more balanced partition using normalized cut [7]. There is public softwares, e.g., Graclus [2], to compute the normalized cut.

  We can now reduce the number of conflicts updating the item vectors, but there is still a possibility of conflicts in user vectors. To address this issue, we use the idea of the NOMAD paper [9]. While the item vectors are moving over the processors in [9], we let user vectors move over the processors each of which contains a subset of the comparisons derived from the graph partition. The algorithm is given in Algorithm 2.

5.3 Parallel AltSVM using graph partitioning

In AltSVM, parallelization of the user part and that of the item part are different. Partitioning the samples by user is straightforward as each sample only contain one unique user. On the other hand, since each sample contain two different items, it is impossible to assign all samples containing the same items to the same thread. In this problem, we view each item j as a node, and two nodes $j1, j2$ will form an edge

Algorithm 2 Parallel Stochastic Gradient Descent with graph partitioning

```
Input: $\Omega$, a step size sequence $\{\eta_t\}_{t=1}^{\infty}$, and the number of processors $p$
Output: $\boldsymbol{U} \in \mathbb{R}^{n \times r}$, $\boldsymbol{V} \in \mathbb{R}^{m \times r}$
 1: Construct an item graph $G = (V, E)$ where $V = \{1, \ldots, m\}$ and $E = \{(j_1, j_2) : \exists i : (i, j_1, j_2) \in \Omega\}$.
 2: Partition the graph into $p$ subsets using normalized cut, and label each item $\pi(j) \in [m]$ by the index of
    the corresponding cluster.
 3: Partition $\Omega$ into $p$ subsets, $\Omega(1), \ldots, \Omega(p)$ such that a comparison $(i, j_1, j_2)$ is in either $\Omega(\pi(j_1))$ or
    $\Omega(\pi(j_2))$.
 4: Initialize $\boldsymbol{U}, \boldsymbol{V}$ to uniform random variables
 5: parfor $t = 1, \ldots, p$ do
 6:     while stop signal is not yet received do
 7:         if queue[$t$] is not empty then
 8:             $i \leftarrow$ queue[$t$].pop()
 9:             for $(i, j_1, j_2) \in \Omega(t)$ do
10:                 if $\boldsymbol{u}_i^\top(\boldsymbol{v}_{j_1} - \boldsymbol{v}_{j_2}) < 1$ then
11:                     $\boldsymbol{u}_i \leftarrow (1 - \frac{\eta\lambda}{|\Omega_i|})\boldsymbol{u}_i - 2\eta(1 - \boldsymbol{u}_i^\top(\boldsymbol{v}_{j_1} - \boldsymbol{v}_{j_2}))(\boldsymbol{v}_{j_1} - \boldsymbol{v}_{j_2})$
12:                     $\boldsymbol{v}_{j_1} \leftarrow (1 - \frac{\eta\lambda}{|\Omega^{j_1}|})\boldsymbol{v}_{j_1} - 2\eta(1 - \boldsymbol{u}_i^\top(\boldsymbol{v}_{j_1} - \boldsymbol{v}_{j_2}))\boldsymbol{u}_i$
13:                     $\boldsymbol{v}_{j_2} \leftarrow (1 - \frac{\eta\lambda}{|\Omega^{j_2}|})\boldsymbol{v}_{j_2} + 2\eta(1 - \boldsymbol{u}_i^\top(\boldsymbol{v}_{j_1} - \boldsymbol{v}_{j_2}))\boldsymbol{u}_i$
14:                 end if
15:             end for
16:             $t' \leftarrow$ next-thread($t$)
17:             queue[$t'$].push($i$)
18:         end if
19:     end while
20: end parfor
```

if $\exists i$ s.t. $(i, j1, j2) \in \Omega$. The edge weight is the number of users owning preference including the given item pair. We want to solve the underling graph partitioning problem such that each partition will have minimum communication (edge cuts) with other ones. In this case each thread can work on one partition asynchronously with minimum conflicts (update the same item simultaneously) during the update of item parts.

As the computational time is determined by the slowest thread, or the thread with heaviest workload, we also want to distribute the workload (edge weight in this problem) on different threads equally. Ideally, the workload on each thread will approximately be W/n, in which W is the total wordload and n is the number of threads, and the speedup will be n.

To partition the item graph, we use Graclus [2]. Graclus utilize a multilevel weighted kernel k-means algorithm in partitioning the graph with various metrics including normalized cut, in which the objective is to minimize is the edge cuts across the partitions divided by number of degrees in each partition. The metric takes both communication and workload into consideration and was used in this problem.

While the user subproblem can be easily parallelized as each $\boldsymbol{u}_i$ is solved independently, parallel algorithms to solve the item subproblem are not trivial. Simple random partition of the dual variables will forbid high scalability because the threads may update the same item vector simultaneously.

- Graph partitioning : Graclus was used to partition the item-based graph as described before. The number of clusters was set to be equal to the number of threads used.

Algorithm 3 Dual Co-ordinate Descent algorithm

```
Input: Ω and C
Output: $\boldsymbol{U} \in n \times r$, $\boldsymbol{V} \in m \times r$
1: Initialize U, V to uniform random variables
2: // preprocessing
3: for e = 1, ..., |Ω| do
4:     (i, j1, j2) = $\Omega[e]$
5:     X(i).push($V_{j1} - V_{j2}$)
6: end for
7: for e = 1, ..., |Ω| do
8:     (i, j1, j2) = $\Omega[e]$
9:     if edge(j1, j2) not exist then
10:        edge(j1, j2) = [i]
11:    else
12:        edge(j1, j2).push(i)
13:    end if
14: end for
15: for iter = 1, ... do
16:     // update users
17:     parfor user i = 1, ..., n do
18:         Initialize $\boldsymbol{\alpha}, \boldsymbol{u}$ to zero
19:         choose a random permutation $\pi$ of $\{1, \dots, |\Omega_i|\}$
20:         for s = 1, ..., $|\Omega_i|$ do
21:             j = $\pi$(s)
22:             solve $\delta^*$ for $\alpha_j$
23:             $\alpha_j = \alpha_j + \delta^*$
24:             $\boldsymbol{u} = \boldsymbol{u} + \delta^* * X(i)_j$
25:         end for
26:         $\boldsymbol{U_i} \leftarrow \boldsymbol{u}$
27:     end parfor
28:     // update items
29:     // partition Graph G into $D_1, \dots, D_k$ s.t. $D_1 \cup \dots \cup D_k = G$
30:     parfor partition i = 1, ..., k do
31:         for edge e in $D_i$ do
32:             (i, j1, j2) = $\Omega[e]$
33:             solve $\delta^*$ for $\alpha_{i,j1,j2}$
34:             $\alpha_{i,j1,j2} = \alpha_{i,j1,j2} + \delta^*$
35:             $\boldsymbol{v}_{j1} = \boldsymbol{v}_{j1} + \delta^* * \boldsymbol{u_i}$
36:             $\boldsymbol{v}_{j2} = \boldsymbol{v}_{j2} - \delta^* * \boldsymbol{u_i}$
37:         end for
38:     end parfor
39: end for
```

6 Experiments

There is no available public dataset of pairwise comparisons for collaborative ranking. We instead extract comparisons from the public datasets for collaborative rating. We will use the datasets in Table 1 (Note: the number of perference data is extracted from a subset of the rating data). The comparisons will be extracted in two ways: Taking random pairs among all observed items, and taking every pair from a random subset of items.

	MovieLens10m	Netflix
Users	71567	480000
Items	10681	17000
Ratings	10000054	100000000
Preference	19175000	149566000

Table 1: Datasets to be used for simulation

To evaluate performance and computational time, we measure the following metrics.

- Test comparison error : We take test comparisons and count the number of incorrect estimates (3835000 and 29913200 samples were tested respectively for movielens and netflix data set).
- Computation time : We measure the running time of the proposed algorithms.
- Scalability : We measure how many threads can be run to gain linear speedup.

6.1 Performance comparison

The performance of alt-rankSVM and SGD using one thread was shown in Figure 1 and 2. The nomad SGD has a fastest convergence rate on both datasets, while random SGD converges slightly slower, both of which are much faster than that of alt-rankSVM. One reason of the slow convergence is that the alt-rankSVM requires at least one dual coordinate descent cycle over all of the dual variables. If we can observe some test error in the middle of the first cycle of dual coordinate descent, we believe that the convergence will look faster. On the other hand alt-rankSVM converges at an lower test error rate for both datasets. For choice of the parameters please see section Choice of parameters.

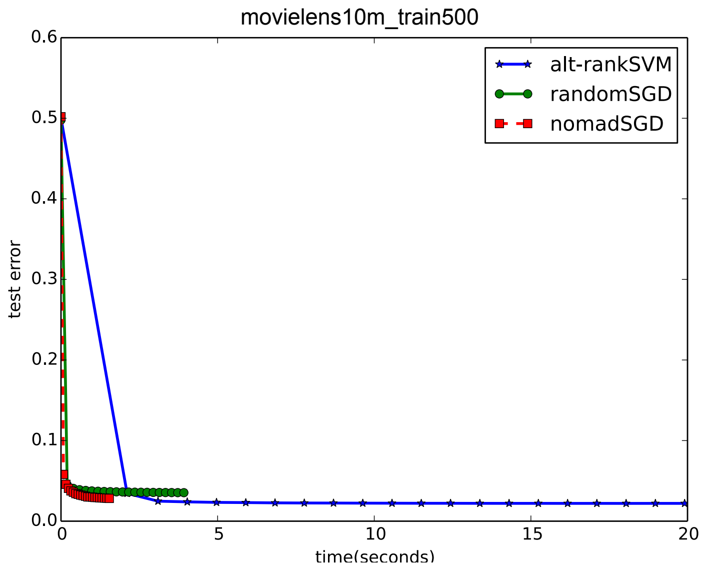

Figure 1: Test error versus time (movielens)

6.2 Scaling in number of cores

The scalability of these algorithms were tested on movielens and netflix data set as well, and the results were shown in Figure 3 and 4. The speedup for nomad SGD was evaluated using time either with or without the

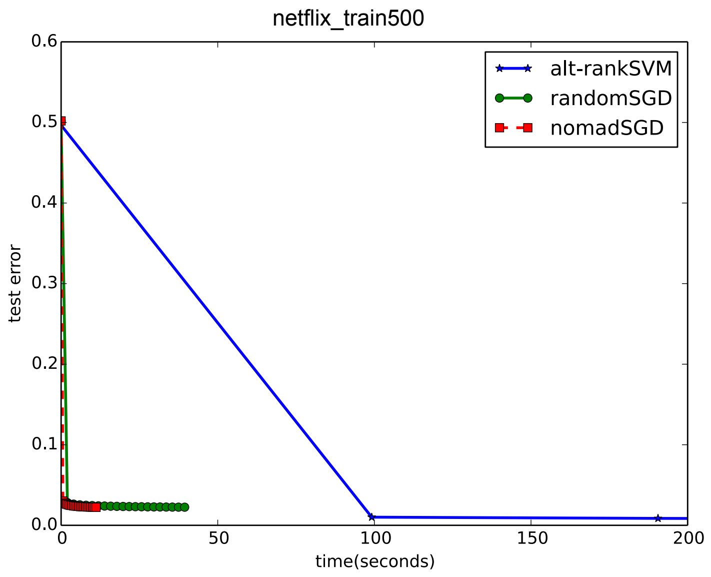

Figure 2: Test error versus time (netflix)

clustering time by Graclus. The proportion of the whole data set to the largest partition by graclus was also plotted, which shows an upbound of the speedup of alt-rankSVM and nomad SGD. Random SGD has a nearly linear speedup on both datasets. The speedup of alt-rankSVM is lower than that of nomad SGD on movielens dataset, and higher on netflix dataset. For choice of the parameters please see section Choice of parameters.

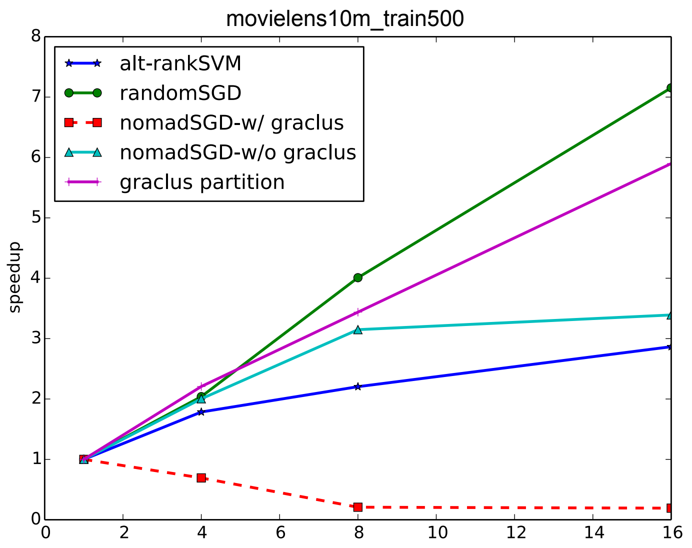

Figure 3: Speedup (movielens)

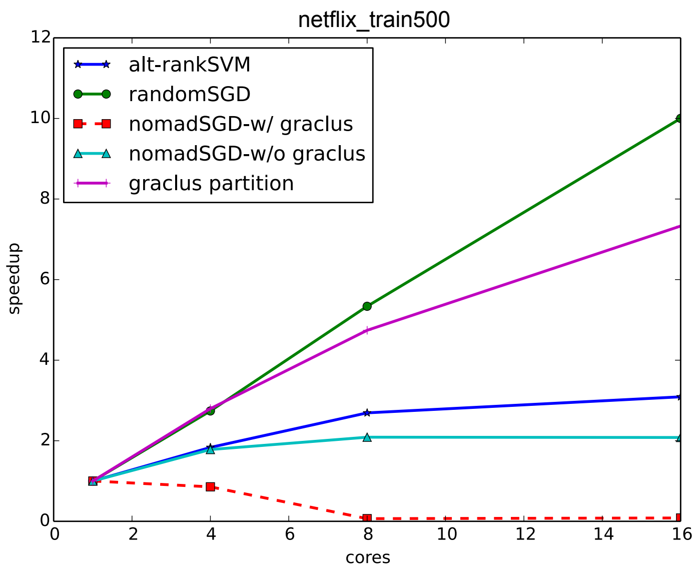

Figure 4: Speedup (netflix)

6.3 Choice of parameters

Different parameters were tried prior to the run of real experiment, and the test error, computational time was considered in order to select the optimal parameters. For alt-rankSVM rank, number of innerIteration and coefficient C were evaluated. The results were shown in Figure 5, 6 and 7 and the optimal values selected were rank = 10, innerIteration = 1, C = 1. For randomSGD beta and lambda were tried. The results were shown in Figure 8 and 9, and the optimal values selected were $\beta = 1e-5, \lambda = 0.1$.

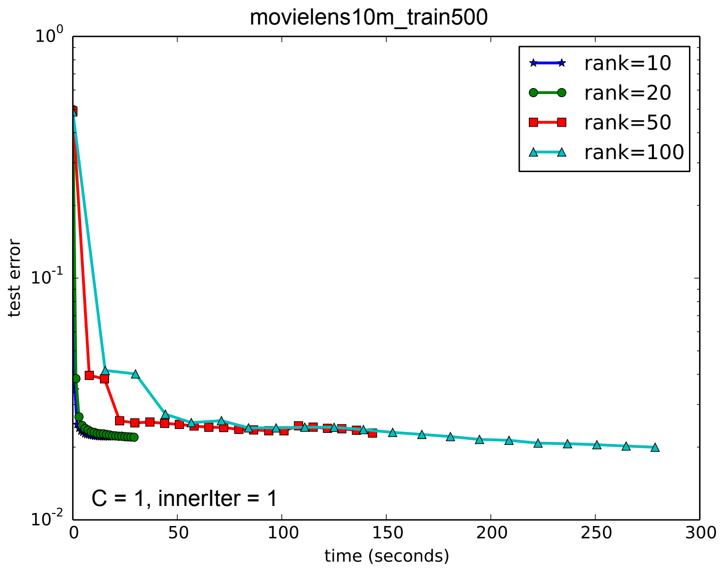

Figure 5: selection of rank

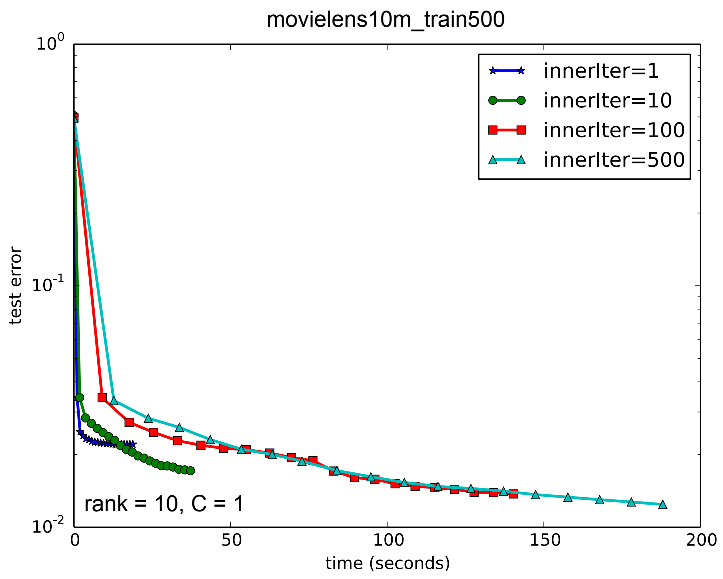

Figure 6: selection of number of innerIteration

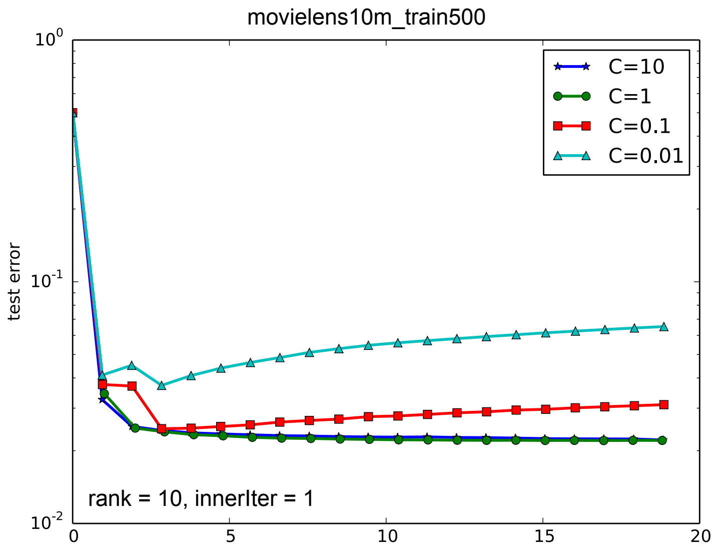

Figure 7: selection of coefficient C

7 Conclusion and Future Work

In this project, we have considered collaborative ranking of multiple items for multiple users from pairwise preference information. We believe that this problem is a more appropriate form of recommendation systems than the usual collaborative filtering for the following two reasons: 1. The ultimate goal of recommendation systems is to provide each user of the items that will be *highly preferred* by the user, and 2. Users' preference information which can be implicit is much more abundant than numerical ratings that are explicit. We

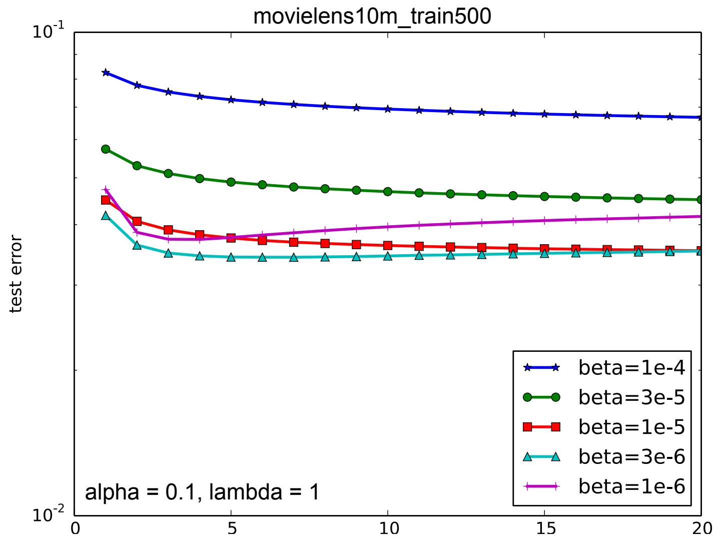

Figure 8: selection of number of innerIteration

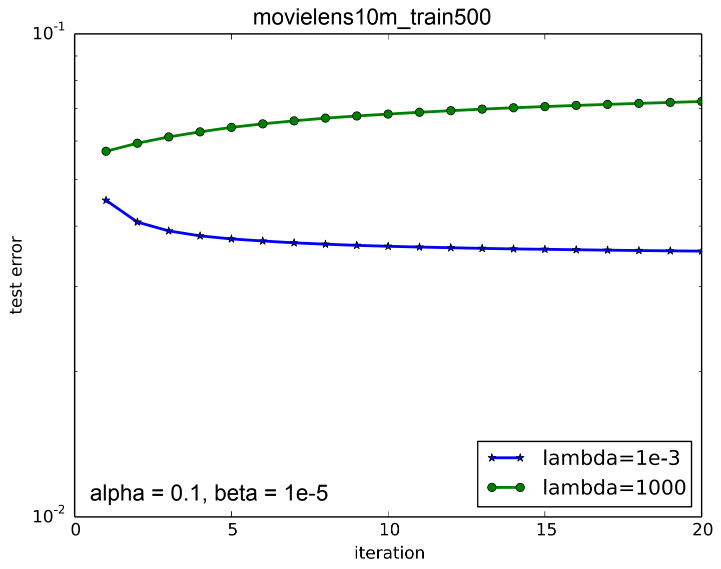

Figure 9: selection of coefficient C

adopted the matrix factorization approach, i.e., the task is to find the underlying low rank matrix that can model the users' preferences. We proposed two algorithms to learn this matrix, stochastic gradient descent and alternating rank-SVM. These algorithms gain linear (random SGD) or nolinear (alt-rankSVM and nomad SGD) speedup with the increase of number of threads.

There are still a few interesting questions remained in this project.

- Implementation optimization

: As shown in Figure 3 and 4, there are gaps between the optimal speedup (purple line) and those from alt-rankSVM and nomad SGD. Further improvement in implementations of these algorithms is necessary to narrow the gaps of performance. The alt-rankSVM also requires a lot of memory and has to be ran on largemem node for the netflix dataset, which is open to optimization as well.

- Comparison with matrix completion algorithms

  : Matrix completion is a slightly different problem from collaborative ranking, but an algorithm of each problem can be used to the other by proper pre- and post-processing. It would be interesting to compare our algorithms with the matrix completion algorithms.

- Can we kernelize this problem?

  : In the binary classification problem, nonlinear kernelized support vector machine (SVM) usually performs better than linear SVM. Can we also kernelize our problem as well? Then would the performance be improved? In particular, since alternating rank-SVM runs two linear SVM alternatively, we can simply replace them by kernel SVMs. However, it is still unknown how it would work.

- How many samples would be sufficient to rank items fairly well?

  : We have empirically observed that the order of 100 comparisons for each user is appropriate. However, how many pairwise comparisons are needed for any low rank matrix? The generalization error bound will answer this question. We believe that the technique in [8] will be useful to obtain the bound.

References

[1] Emmanuel Candes and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 2009.

[2] Inderjit Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 29(11), November 2007.

[3] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *ICML*, 2008.

[4] Thorsten Joachims. Optimizing search engines using clickthrough data. In *SIGKDD*, 2002.

[5] George Karypis, Rajat Aggarwal, and Shashi Shekhar. Multilevel hypergraph partitioning: Applications in vlsi domain. *IEEE Transactions on VLSI Systems*, 7(1), 1999.

[6] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

[7] Jianbo Shi and Jitendra Malik. Normalized cuts and image segementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22(8), 2000.

[8] Nathan Srebro, Noga Alon, and Tommi Jaakkola. Generalization error bounds for collaborative prediction with low-rank matrices. In *NIPS*, 2004.

[9] Hyokun Yun, Hsiang-Fu Yu, Cho-Jui Hsieh, S. V. N. Viswanathan, and Inderjit S. Dhillon. NOMAD: Non-locking, stochastic multi-machine algorithm for asynchronous and descentralized matrix completion. In *VLDB*, 2014.