Large-Scale Parallel Collaborative Filtering for the Netflix Prize

Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan

HP Labs, 1501 Page Mill Rd, Palo Alto, CA, 94304
{yunhong.zhou,dennis.wilkinson,rob.schreiber,rong.pan}@hp.com

Abstract. Many recommendation systems suggest items to users by utilizing the techniques of *collaborative filtering* (CF) based on historical records of items that the users have viewed, purchased, or rated. Two major problems that most CF approaches have to contend with are scalability and sparseness of the user profiles. To tackle these issues, in this paper, we describe a CF algorithm *alternating-least-squares with weighted-λ-regularization* (ALS-WR), which is implemented on a parallel Matlab platform. We show empirically that the performance of ALS-WR (in terms of *root mean squared error* (RMSE)) monotonically improves with both the number of features and the number of ALS iterations. We applied the ALS-WR algorithm on a large-scale CF problem, the Netflix Challenge, with 1000 hidden features and obtained a RMSE score of 0.8985, which is one of the best results based on a pure method. In addition, combining with the parallel version of other known methods, we achieved a performance improvement of 5.91% over Netflix's own CineMatch recommendation system. Our method is simple and scales well to very large datasets.

1 Introduction

Recommendation systems try to recommend items (movies, music, webpages, products, etc) to interested potential customers, based on the information available. A successful recommendation system can significantly improve the revenue of e-commerce companies or facilitate the interaction of users in online communities. Among recommendation systems, *content-based* approaches analyze the content (e.g., texts, meta-data, features) of the items to identify related items, while *collaborative filtering* uses the aggregated behavior/taste of a large number of users to suggest relevant items to specific users. Collaborative filtering is popular and widely deployed in Internet companies like Amazon [16], Netflix [2], Google News [7], and others.

The Netflix Prize is a large-scale data mining competition held by Netflix for the best recommendation system algorithm for predicting user ratings on movies, based on a training set of more than 100 million ratings given by over 480,000 users to 17,700 movies. Each training data point consists of a quadruple (user, movie, date, rating) where rating is an integer from 1 to 5. The test dataset consists of 2.8 million data points with the ratings hidden. The goal is to minimize the RMSE (root mean squared error) when predicting the ratings on the test dataset. Netflix's own recommendation system (CineMatch) scores 0.9514 on the test dataset, and the grand challenge is to improve it by 10%.

The Netflix problem presents a number of practical challenges. (Which is perhaps why, as yet, the prize has not been won.) First, the size of the dataset is 100 times larger than previous benchmark datasets, resulting in much longer model training time and much larger system memory requirements. Second, only about 1% of the user-movie matrix has been observed, with the majority of (potential) ratings missing. This is, of course, an essential aspect of collaborative filetering in general. Third, there is noise in both the training and test dataset, due to human behavior – we cannot expect people to be completely predictable, at least where their feelings about ephemera like movies is concerned. Fourth, the distribution of ratings per user in the training and test datasets are different, as the training dataset spans many years (1995-2005) while the testing dataset was drawn from recent ratings (year 2006). In particular, users with few ratings in the training set are well represented in the test set. Intuitively, it is hard to predict the ratings of a user who is sparsely represented in the training set.

In this paper, we introduce the problem in detail. Then we describe a parallel algorithm, alternating-least-squares with weighted-λ-regularization. We use parallel Matlab on a Linux cluster as the experimental platform, and our core algorithm is parallelized and optimized to scale up well with large, sparse data. When we apply the proposed method to the Netflix Prize problem, we achieve a performance improvement of 5.91% over Netflix's own CineMatch system.

The rest of the paper is organized as follows: in Section 2 we introduce the problem formulation. In Section 3 we describe our novel parallel Alternative-Least-Squares algorithm. Section 4 describes experiments that show the effectiveness of our approach. Section 5 discusses related work and Section 6 concludes with some future directions.

2 Problem Formulation

Let $R = \{r_{ij}\}_{n_u \times n_m}$ denote the user-movie matrix, where each element r_{ij} represents the rating score of movie j rated by user i with its value either being a real number or missing, n_u designates the number of users, and n_m indicates the number of movies. As in most recommendation systems our task is to estimate some of the missing values in R based on the known values. (The Netflix dataset consists of $n_m = 17770$ movies, $n_u = 488000$ users, and $n_r \approx 100$ million known ratings.)

We start with a low-rank approximation of the ratings matrix R. This approach models both users and movies by giving them coordinates in a low dimensional feature space. Each user and each movie has a feature vector, and each rating (known or unknown) of a movie by a user is modeled as the inner product of the corresponding user and movie feature vectors. More specifically, let $U = [\mathbf{u_i}]$ be the user feature matrix, where $\mathbf{u_i} \in \mathbb{R}^{n_f}$, $i = 1 \ldots n_u$, denotes the i^{th}column of U, and let $M = [\mathbf{m_j}]$ be the movie feature matrix, where

$\mathbf{m_j} \in \mathbb{R}^{n_f},\ j = 1 \ldots n_m$, is the j^{th}column of M. Here n_f is the dimension of the feature space, that is, the number of hidden variables in the model. It is a system parameter that can be determined by a hold-out dataset or cross-validation. If user ratings were fully predictable and n_f sufficiently large, we could expect that $r_{ij} = < \mathbf{u_i}, \mathbf{m_j} >$, $\forall\ i, j$. In practice, however, we minimize a loss function (of U and M) to obtain the matrices U and M. In this paper, we study the mean-square loss function. The loss due to a single rating is defined as the squared error:

$$\mathcal{L}^2(r, \mathbf{u}, \mathbf{m}) = (r- < \mathbf{u}, \mathbf{m} >)^2. \tag{1}$$

Then we can define the empirical, total loss (for a given pair U and M) as the summation of loss on all the known ratings in Eq. (2).

$$\mathcal{L}^{emp}(R, U, M) = \frac{1}{n} \sum_{(i,j) \in I} \mathcal{L}^2(r_{ij}, \mathbf{u_i}, \mathbf{m_j}), \tag{2}$$

where I is the index set of the known ratings and n is the size of I.

We can formulate the low-rank approximation problem as follows.

$$(U, M) = \arg \min_{(U,M)} \mathcal{L}^{emp}(R, U, M). \tag{3}$$

where U and M are real, have n_f rows, but are otherwise unconstrained.

In this problem, (Eq. (3)), there are $(n_u + n_m) \times n_f$ free parameters to be determined. Our results show that allowing n_f to be quite large, 1000 in our tests, improves the quality of the results. Thus, for Netflix we have over $480,000,000$ model parameters, more data than there are points in the training set I. For this reason, problem 3 is underdetermined. Indeed, very few users will have rated n_f movies when n_f is greater than a few hundred, yet we must assign each user a point in an n_f dimensional feature space.

Thus, when n_f is relatively large, solving the problem Eq. (3) overfits the data. To avoid overfitting, a common method appends a Tikhonov regularization [22] term to the empirical risk function:

$$\mathcal{L}^{reg}_{\lambda}(R, U, M) = \mathcal{L}^{emp}(R, U, M) + \lambda(\|U\Gamma_U\|^2 + \|M\Gamma_M\|^2), \tag{4}$$

for a certain suitably selected Tikhonov matrices Γ_U and Γ_M.[1] We will discuss the details in the next section.

3 Our Approaches

In this section, we describe an iterative algorithm, alternating-least-squares with weighted-λ-regularization (ALS-WR), to solve the low rank approximation problem. Then we develop a parallel implementation of ALS-WR based on a parallel Matlab platform.

[1] Throughout the paper, $\|X\|$ denotes the Frobenius norm of the matrix X.

3.1 ALS with Weighted-λ-Regularization

As the rating matrix contains both signals and noise, it is important to remove noise and use the recovered signal to predict missing ratings. *Singular Value Decomposition* (SVD) is a natural approach that approximates the original user-movie rating matrix R by the product of two rank-k matrices $\tilde{R} = U^T \times M$. The solution given by the SVD minimizes the Frobenious norm of $R - \tilde{R}$, which is equivalent to minimizing the RMSE over all elements of R. However, as there are many missing elements in the rating matrix R, standard SVD algorithms cannot find U and M.

In this paper, we use *alternating-least-squares* (ALS) to solve the low-rank matrix factorization problem as follows:

Step 1. Initialize matrix M by assigning the average rating for that movie as the first row, and small random numbers for the remaining entries.
Step 2. Fix M, Solve for U by minimizing the objective function (4);
Step 3. Fix U, solve for M by minimizing the objective function similarly;
Step 4. Repeat Steps 2 and 3 until a stopping criterion is satisfied.

Observe that when the regularization matrices $\Gamma_{(U,M)}$ are nonsingular, each of the problems of Steps 2 and 3 of the algorithm has a unique solution, which we derive below. Note that the sequence of achieved errors $\mathcal{L}_\lambda^{reg}(R, U, M)$ is monotone nonincreasing and bounded below, hence this sequence converges.

Rather than going all the way to convergence, we use a stopping criterion based on the observed RMSE on the probe dataset. The probe dataset is provided by Netflix, and it has the same distribution as the hidden test dataset. After one round of updating both U and M, if the change in RMSE on the probe dataset is less than 1 bps[2], the iteration stops and we use the obtained U, M to make final predictions on the test dataset.

As we mention in Section 2, there are many free parameters. Without regularization, ALS might lead to overfitting. A common fix is to use Tikhonov regularization, which penalizes large parameters. We tried various regularization matrices, and eventually found the following weighted-λ-regularization to work the best, as it never overfits the test data (empirically) when we increase the number n_f of features or the number of ALS iterations.

$$f(U, M) = \sum_{(i,j) \in I} (r_{ij} - \mathbf{u}_i^T \mathbf{m}_j)^2 + \lambda \left(\sum_i n_{u_i} \|\mathbf{u}_i\|^2 + \sum_j n_{m_j} \|\mathbf{m}_j\|^2 \right), \quad (5)$$

where n_{u_i} and n_{m_j} denote the number of ratings of user i and movie j respectively. This corresponds to Tikhonov regularization (4), where $\Gamma_U = \text{diag}(\sqrt{n_{u_i}})$ and $\Gamma_M = \text{diag}(\sqrt{n_{m_j}})$. [3]

[2] 1 bps equals 0.0001.

[3] The same objective function was used previously by Salakhutdinov et al. [20] and solved using gradient descent. We will discuss more on their approach in Section 5.2.

Now we demonstrate how to find the matrix U when M is given. Let I_i^U denote the set of indices of movies that user i rated, so that n_{u_i} is the cardinality of I_i^U; similarly I_j^M denotes the set of indices of users who rated movie j, and n_{m_j} is the cardinality of I_j^M. A given column of U, say u_i, is determined by solving a regularized linear least squares problem involving the known ratings of user i, and the feature vectors $\mathbf{m}_j$ of the movies $j \in I_i^U$ that user i has rated.

$$\begin{aligned}
&\frac{1}{2}\frac{\partial f}{\partial u_{ki}} = 0, \quad \forall i, k \\
\Rightarrow &\sum_{j \in I_i^U} (\mathbf{u}_i^T \mathbf{m}_j - r_{ij}) m_{kj} + \lambda n_{u_i} u_{ki} = 0, \quad \forall i, k \\
\Rightarrow &\sum_{j \in I_i^U} m_{kj} \mathbf{m}_j^T \mathbf{u}_i + \lambda n_{u_i} u_{ki} = \sum_{j \in I_i^U} m_{kj} r_{ij}, \quad \forall i, k \\
\Rightarrow &\left(M_{I_i^U} M_{I_i^U}^T + \lambda n_{u_i} E \right) \mathbf{u}_i = M_{I_i^U} \; R^T(i, I_i^U), \quad \forall i \\
\Rightarrow &\,\mathbf{u}_i = A_i^{-1} V_i, \quad \forall i
\end{aligned}$$

where $A_i = M_{I_i^U} M_{I_i^U}^T + \lambda n_{u_i} E$, $V_i = M_{I_i^U} \; R^T(i, I_i^U)$, and E is the $n_f \times n_f$ identity matrix. $M_{I_i^U}$ denotes the sub-matrix of M where columns $j \in I_i^U$ are selected, and $R(i, I_i^U)$ is the row vector where columns $j \in I_i^U$ of the i-th row of R are selected.

Similarly, when M is updated, we can compute individual m_j's via a regularized linear least squares solution, using the feature vectors of users who rated movie j, and their ratings of it, as follows:

$$\mathbf{m}_j = A_j^{-1} V_j, \quad \forall j,$$

where $A_j = U_{I_i^M} U_{I_i^M}^T + \lambda n_{m_j} E$ and $V_j = U_{I_i^M} R(I_i^M, j)$. $U_{I_i^M}$ denotes the sub-matrix of U where columns $i \in I_i^M$ are selected, and $R(I_i^M, j)$ is the column vector where rows $i \in I_i^M$ of the j-th column of R is taken.

Let n_r denote the total number of ratings, then we have $n_r \ll n_u n_m$ for a sparse rating matrix. We can easily derive the running time of the above algorithm based on standard matrix operations:

Theorem 1. *For ALS-WR, each step of updating U takes $O\left(n_f^2(n_r + n_f n_u)\right)$ time while each step of updating M takes $O\left(n_f^2(n_r + n_f n_m)\right)$. If ALS-WR takes totally n_t rounds to stop, it runs in time $O\left(n_f^2(n_r + n_f n_u + n_f n_m) n_t\right)$.*

3.2 Parallel ALS with Weighted-λ-Regularization

We parallelize ALS by parallelizing the updates of U and of M. We are using the latest verstion of Matlab, which allows parallel Matlab computation in which several separate copies of Matlab, each with its own private workspace, and each running on its own hardware platform, collaborate and communicate to solve problems. Each such running copy of Matlab is referred to as a "lab", with its own identifier (labindex) and with a static variable (numlabs) telling how many labs there are. Matrices can be private (each lab has its own copy, and their values differ), replicated (private, but with the same value on all labs) or distributed (there is one matrix, but with rows, or columns, partitioned among the labs). Distributed matrices are a convenient way to store and use very large datasets, too large to be stored on one processor and its associated local memory. In our case, we use two distributed copies of the ratings matrix R, one distributed by rows (i.e., by users) and the other by columns (i.e., by movies). We will compute distributed, updated matrices U and M. In computing U we will require a replicated version of M, and *vice versa*. Thus, our labs communicate to make replicated versions of U and of M from the distributed versions that are first computed. Matlab's "gather" function performs the inter-lab communication needed for this.

To update M, we require a replicated copy of U, local to each lab. We use the ratings data distributed by columns (movies). The distribution is by blocks of equal numbers of movies. The lab that stores the ratings of movie j will, naturally, be the one that updates the corresponding column of M, which is movie j's feature vector. Each lab computes $\mathbf{m_j}$ for all movies in the corresponding movie group, in parallel. These values are then "gathered" so that every node has all of M, in a replicated array. To update U, users are partitioned into equal-size user groups and each lab just update user vectors in the corresponding user group, using the ratings data partitioned by rows. The following Matlab snippet implements the procedure of updating M given U:

```
function M = updateM(lAcols, U)
    lamI = lambda * eye(Nf);
    lM = zeros(Nf,Nlm); lM = single(lM);
    for m = 1:Nlm
        users = find(lAcols(:,m));
        Um = U(:, users);
        vector = Um * full(lAcols(users, m));
        matrix = Um * Um' + locWtM(m) * lamI;
        X = matrix \ vector;
        lM(:, m) = X;
    end
    M = gather(darray(lM));
end
```

For the above Matlab code, lAcols is the local copy of R distributed by columns (movies), locWtM is the vector of n_{m_j} for all movies in the partitioned movie group, and Nlm is the number of movies in the movie group. Nf and lambda correspond to n_f and λ, and they are the only tunable parameters of ALS-WR.

The gather operation is the only place where we incur communication cost due to using a distributed (as opposed to a shared-memory) algorithm. For our method, it takes less than 5% of the total run time. Therefore, *the parallel* *algorithm achieves a nearly linear speedup.* As an example, for $n_f = 100$, it takes about 2.5 hours to update M and U once with a single processor; with 30 processors, it takes about 5 minutes for one iteration, and the converged solution (with 30 ALS iterations) can be computed in 2.5 hours.

4 Performance for the Netflix Prize Problem

We ran on a 30-processor Linux cluster of HP ProLiant DL380 G4 machines. All processors are Xeon 2.8GHz and every four processors share 6GB of RAM. For each fixed n_f, we run between 10 to 25 rounds of ALS-WF and stop when one round of U, M update improves by less than 1 bps the RMSE score on the probe dataset. The optimal value of λ is determined by trial and error.[4] The test RMSE is obtained by submission to the Netflix prize website[5].

The probe dataset is a subset of the training dataset. It consists of 1,408,395 ratings from the year 2006. In it, users are selected at random with uniform probability, and at most 9 ratings are included for each selected user. The test dataset is hidden by Netflix but the distribution of the test dataset is the same as the distribution of the probe dataset. For model building and parameter tuning, we exclude the probe dataset from the training dataset (this is how we determine the set I, above.) We use the probe for convergence criterion in ALS-WR and for testing.

4.1 Postprocessing

For postprocessing of the prediction results, we first apply a global bias correction technique to the prediction solution. Given a prediction P, if the mean of P is not equal to the mean of the test dataset, we can shift all predicted values by a fixed constant $\tau = \text{mean(test)} - \text{mean}(P)$, thus improving the RMSE. We also use convex combinations of several predictors to obtain a better predictor. For example, given two predictors P_0 and P_1, we can obtain a family of predictors $P_x = (1-x)P_0 + xP_1,\ x \in [0,1]$, and use linear regression to find x^* minimizing $\text{RMSE}(P_x)$. Therefore we obtain P_{x^*} which is at least as good as P_0 or P_1.

4.2 Experimental Results for ALS

The most important discovery we made is that ALS-WR never overfits the data if we either increase the number of iterations or the number of hidden features. As Figure 1 shows, with fixed n_f and λ, each iteration improves the RMSE score of the probe dataset, and it converges after about 20 rounds. Different λ values give different final score, and we normally need to try a small number of

[4] Empirically we found that for fixed n_f, the convergence RMSE score is a convex function of λ, and the optimal value of λ is monotone decreasing with respect to n_f. Based on these observations, we are able to find the best value of λ for each n_f with only 2-3 experiments.

[5] See `http://www.netflixprize.com/rules` for the detailed rules of the competition.

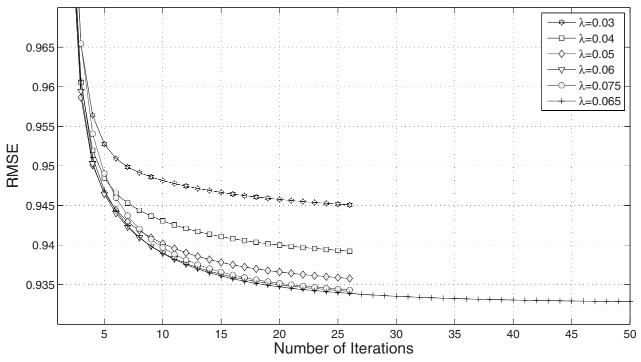

Fig. 1. Comparisons of different λ values for ALS-WR with $n_f = 8$. The best performer with 25 rounds is $\lambda = 0.065$. For this fixed λ, after 50 rounds, the RMSE score still improves but only less than 0.1 bps for each iteration afterwards.

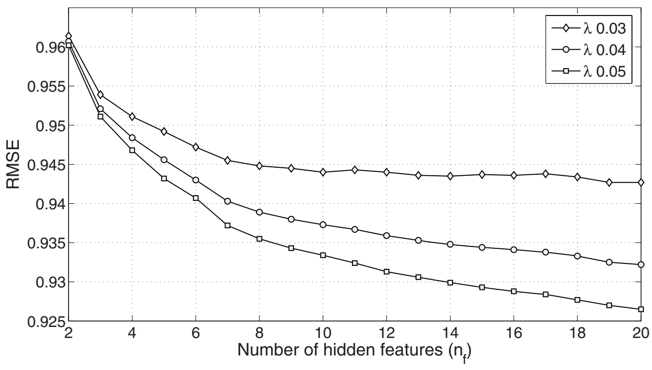

Fig. 2. Performance of ALS-WR with fixed λ and varying n_f

λ values to get a good RMSE score. Figure 2 shows the performance of ALS-WR with fixed λ value and varying number of hidden features (n_f ranges from 2 to 20). For each experiment, ALS-WR iterations continue until the RMSE over the probe dataset improves less than 1 bps. From the figure we can tell that the RMSE monotonically decreases with larger n_f, even though the improvement diminishes gradually.

Next we conduct experiments with real submissions using large values of n_f. For ALS with simple λ regularization ($\Gamma_u = \Gamma_m = E$), we obtain a RMSE of 0.9184. For ALS with weighted-λ-regularization, we obtained a RMSE of 0.9114 with $n_f = 50$, 0.9066 with $n_f = 150$. With $n_f = 300$ and global bias correction, we obtain a RMSE of 0.9017; with $n_f = 400$ and global bias correction, a score of 0.9006 was obtained; with $n_f = 500$ and global bias shift, a score of 0.9000 was obtained. Ultimately, we experimented with $n_f = 1000$ and obtained a RMSE score of 0.8985.[6] Given that 6 bps improvement is obtained from 400 to 500 features, and assuming diminishing (equal-decrease) return with increasing number of features, moving from 500 to 1000 features improves approximately $5 + 4 + 3 + 2 + 1 = 15$ bps. Therefore, 0.8985 is likely the limit we can achieve using ALS with Weighted-λ-Regularization. A RMSE score of 0.8985 translates into a 5.56% improvement over Netflix's CineMatch, and it represents one of the top *single-method* performance according to our knowledge.

4.3 Other Methods and Linear Blending

We also implement parallel versions of two other popular collaborative filtering techniques as described in this section. In each case, the speedup as compared to a single-processor version is roughly a factor of n on a cluster of n processors.

The Restricted Boltzmann Machine (RBM) is a kind of neural network where there are visible states and hidden states, and undirected edges connecting each visible state to each hidden state. There are no connections among visible states or among hidden states, thus the name "restricted." RBM was previously demonstrated to work well for the Netflix challenge [20]. We implemented RBM using Matlab, and converted it to Pmode. For a model with 100 hidden units, it takes about 1 hour for one iteration without Pmode; using Pmode with 30 labs, it takes 3 minutes for one iteration.

The k-nearest neighbor (kNN) method is also popular for prediction. With a properly defined distance metric, for each data point needed for prediction, the weighted average of the ratings of its k closest neighbors is used to predict the rating of this point. Since there are so many user-user pairs for us to handle in reasonable time and space, we use a simplified approach with only movie-movie similarities. Again, we parallelize kNN by partitioning users into user groups so that each lab processes one user group.

For RBM itself, a score of 0.9181 is obtained. For kNN with $k = 21$ and a good similarity function, a RMSE of 0.9270 is obtained. Linear blending of ALS with kNN and RBM yields a RMSE of 0.8952 (ALS + kNN + RBM), and it represents a 5.91% improvement over Netflix's CineMatch system.

5 Related Work

There is a lot of academic and industrial work on recommendation systems, low-rank matrix approximation, and the Netflix prize. In the following we briefly discuss related work most relevant to ours.

[6] The experiment with $n_f = 1000$ is technically challenging as U takes $2G$ memory with single precision entries for each processor. We managed to run the procedure of updateM in two batches, while in each batch only two processors for each server are active and U is only replicated in these processors. This avoids memory thrashing using the 6G shared memory for each server. And ALS-WR stops in 10 rounds while each rounds takes 1.5 hours.

5.1 Recommendation Systems

Recommendation systems can be mainly categorized into content-based and collaborative filtering, and are well-studied in both academia and industry [16,2,7]. *Content-based* recommendation systems analyze the content (e.g., texts, metadata, features) of the item to identify related items, with exemplary systems InfoFinder [12], NewsWeeder [14]. *Collaborative Filtering* uses aggregated behavior/taste of a large number of users to suggest relevant items to specific users. Recommendations generated by CF are based solely on the user-user and/or item-item similarities, with exemplary systems GroupLens [19] and Bellcore Video Recommender [11]. Efforts to combine both content-based approach and collaborative filtering include the Fab system [3] and unified probabilistic framework [18].

5.2 The Netflix Prize Approaches

For the Netflix prize, Salakhutdinov et al. [20] used Restricted Boltzmann Machines (RBM), obtaining an RMSE score of slightly below 0.91. They also presented a low-rank approximation approach using gradient descent. Their low-rank approximation obtained an RMSE score slightly above 0.91 using between 20-60 hidden features.[7] The objective function of their SVD approach is the same as our ALS-WR, however we use alternating least squares instead of gradient descent to solve the optimization problem, and we are able to use a much large number of features (1000 rather than 40) to obtain significant improvement in RMSE score.

Among many other approaches to the Netflix problem, Bell et al. [5] proposed a neighborhood-based technique which combines k-nearest-neighbor (kNN) and low-rank approximation to obtain significantly better results compared to either technique alone. Their team won the progress prize in October 2007, obtaining an RMSE score on the qualifying dataset of 0.8712, improving the CineMatch score by 8.5%. However, their solution [4] is a linear combination of 107 individual solutions, while multiple solutions are derived by variants of three classes of solutions (ALS, RBM, and kNN). For ALS alone, their best result was obtained using 128 hidden features with an RMSE score above 0.9000. For a comprehensive treatment of various approaches for the Netflix prize, see the individual papers presented in KDD Cup & Workshop 2007 [21,17,15,13,23].

5.3 Low-Rank Approximation

When a fully specified matrix is to be approximated by a low-rank matrix factorization, variants of singular value decomposition are used, for example in information retrieval (where SVD techniques are known as latent semantic indexing [9]). Other matrix factoring methods, for example nonnegative matrix factorization and maximum margin matrix factorization have also been proposed for the Netflix prize [23].

[7] We obtained similar results and got 0.9114 with $n_f = 50$.

For a partially specified matrix, the SVD is not applicable. To minimize the sum of squared differences between the known elements and the corresponding elements of the factored low rank matrix, ALS has proven to be an effective approach. It provides non-orthogonal factors, unlike SVD. The SVD can be computed one column at a time, whereas for the partially specified case, no such recursive formulation holds. An advantage of ALS is its easy parallelization. Like Lanczos for the sparse, fully specified case, ALS preserves the sparse structure of the known matrix elements and is therefore storage-efficient.

6 Concluding Remarks

We introduced a simple parallel algorithm for large-scale collaborative filtering which, in the case of the Netflix prize, performed as well as any single method reported in the literature. Our algorithm is designed to be scalable to very large datasets. Moderately better scores can be obtained by refining the RBM and kNN implementation or using more complicated blending schemes. ALS-WR in particular is able to achieve good results without using date or movie title information. The fast runtime achieved through parallelization is a competitive advantage for model building and parameter tuning in general. It will be interesting to develop a theory to explain why ALS-WR never overfits the data.

As the world shifts into Internet computing and web applications, large-scale data intensive computing becomes pervasive. Traditional single-machine, single-thread computing is no longer viable, and there is a paradigm shift in computing models. Parallel and/or distributed computing becomes an essential component for any computing environment. Google, the leading Internet company, is building its own proprietary parallel/distributed computing infrastructure, based on MapReduce [8], Google File System [10], Bigtable [6], etc. Most technology companies do not have the capital and expertise to develop an in-house large-scale parallel/distributed computing infrastructure, and prefer instead to use readily available solutions to solve computing infrastructure problems. Hadoop [1] is an open-source project sponsored by Yahoo!, which tries to replicate the Google computing infrastructure with open-source development. We have found parallel Matlab to be flexible and efficient, and very straightforward to program. Thus, from our experience, it seems to be a strong candidate for widespread, easily scalable parallel/distributed computing.

References

1. The Hadoop Project, `http://lucene.apache.org/hadoop/`
2. Netflix CineMatch, `http://www.netflix.com`
3. Balabanovi, M., Shoham, Y.: Fab: content-based, collaborative recommendation. Communications of the ACM 40(3), 66–72 (1997)
4. Bell, R., Koren, Y., Volinsky, C.: The bellkor solution to the netflix prize. Netflix Prize Progress Award (October 2007), `http://www.netflixprize.com/assets/ProgressPrize2007_KorBell.pdf`

5. Bell, R., Koren, Y., Volinsky, C.: Modeling relationships at multiple scales to improve accuracy of large recommender systems. In: Proc. KDD 2007, pp. 95–104 (2007)
6. Chang, F., et al.: Bigtable: A distributed storage system for structured data. In: Proc. of OSDI 2006, pp. 205–218 (2006)
7. Das, A., Datar, M., Garg, A., Rajaram, S.: Google news personalization: Scalable online collaborative filtering. In: Proc. of WWW 2007, pp. 271–280 (2007)
8. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: Proc. OSDI 2004, San Francisco, pp. 137–150 (2004)
9. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. J. Amer. Soc. Info. Sci. 41(6), 391–407 (1999)
10. Ghemawat, S., Gobioff, H., Leung, S.-T.: The Google File System. In: Proc. of SOSP 2003, pp. 29–43 (2003)
11. Hill, W., Stead, L., Rosenstein, M., Furnas, G.: Recommending and evaluating choices in a virtual community of use. In: Proc. of CHI 1995, Denver (1995)
12. Krulwich, B., Burkey, C.: Learning user information interests through extraction of semantically significant phrases. In: Proc. AAAI Spring Symposium on Machine Learning in Information Access, Stanford, CA (March 1996)
13. Kurucz, M., Benczur, A.A., Csalogany, K.: Methods for large scale SVD with missing values. In: Proc. KDD Cup and Workshop (2007)
14. Lang, K.: NewsWeeder: Learning to filter Netnews. In: Proc. ICML 1995, pp. 331–339 (1995)
15. Lim, Y.J., Teh, Y.W.: Variational bayesian approach to movie rating prediction. In: Proc. KDD Cup and Workshop (2007)
16. Linden, G., Smith, B., York, J.: Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing 7, 76–80 (2003)
17. Paterek, A.: Improving regularized singular value decomposition for collaborative filtering. In: Proc. KDD Cup and Workshop (2007)
18. Popescul, A., Ungar, L., Pennock, D., Lawrence, S.: Probabilistic models for unified collaborative and content-based recommendation in Sparse-Data Environments. In: Proc. UAI, pp. 437–44 (2001)
19. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J.: GroupLens: an open architecture for collaborative filtering of Netnews. In: Proc. the ACM Conference on Computer-Supported Cooperative Work, Chapel Hill, NC (1994)
20. Salakhutdinov, R., Mnih, A., Hinton, G.E.: Restricted boltzmann machines for collaborative filtering. In: Proc. ICML, pp. 791–798 (2007)
21. Takacs, G., Pilaszy, I., Nemeth, B., Tikk, D.: On the gravity recommendation system. In: Proc. KDD Cup and Workshop (2007)
22. Tikhonov, A.N., Arsenin, V.Y.: Solutions of Ill-posed Problems. John Wiley, New York (1977)
23. Wu, M.: Collaborative filtering via ensembles of matrix factorizations. In: Proc. KDD Cup and Workshop (2007)