Predicting User Ratings with Matrix Factorization

Jiunn Haur Lim
Carnegie Mellon University
Pittsburgh, PA 15289
jiunnhal@cmu.edu

Abstract

With e-commerce becoming more popular, users are often faced with a tremendous number of choices, and recommender systems have become necessary to help them navigate through the sea of products. Collaborative filtering is a popular method used to aggregate user preferences and predict product ratings. I will explore different variations of Alternating Least Squares, a matrix factorization algorithm that decomposes the ratings matrix into latent factor matrices. ALS allows latent factors to be solved in parallel, which makes it easily scalable to large datasets. When implemented on GraphLab and scaled up to the full dataset of 100 million ratings, I was able to get a best RMSE of 0.9288, which represents a 2% improvement over Netflix's Cinematch algorithm.

1 Introduction

As the World Wide Web becomes more ubiquitous and e-commerce becomes more popular, barriers to entry for several markets are reduced and there is an explosion in the variety of goods and services available for the Web user. Consumers are frequently overwhelmed by the large number of choices. An effective recommender system not only improves the overall user experience but can also increase sales revenues significantly. *Collaborative filtering* is a popular technique used by recommender systems to aggregate the preferences of a large number of users and make recommendations to a specific user based on the similarity of his preferences to those of the other users [1]. The input to such an algorithm is usually the set of ratings that users have explicitly assigned to items, but it may also include users' purchase histories and other profile information. In this report, I will study different variations of a particular collaborative filtering algorithm that uses matrix factorization to predict the user ratings. A user recommender system may use these predicted ratings to generate recommendation lists.

1.1 The Netflix Prize

This study will be conducted on the Netflix dataset released during the Netflix Prize competition [2]. The dataset provides 100 million ratings that 480,189 users gave to 17,770 films. Participants were challenged to use these ratings to develop collaborative filtering algorithms that predict user ratings for films. Submitted algorithms were tested against a qualifying set of 2,817,131 ratings, and each algorithm was scored using the root mean squared error (RMSE) of its predictions. Netflix also identified 1,408,395 ratings in the dataset that has the same distribution of ratings as the qualifying set. I will refer to this set of ratings as the *probe set*.

The goal of the competition was to beat Netflix's Cinematch algorithm, which scored an RMSE of 0.9514 on the qualifying set. Since the competition has ended and I do not have access to the qualifying set, I will separate the probe set from the training data and attempt to beat Cinematch using the probe set.

Zhou, et. al. identified the following four challenges [3]:

1. The large number of ratings, users, and movies will require long model training times and large system memory.
2. The user-movie matrix, $\mathbf{R}$, is incomplete - only about 1% of the ratings are known.
3. Human behavior is highly unpredictable, which contributes to significant noise in the training set and quiz set.
4. The training set and qualifying set may have different distributions of ratings.

In this report, I will first describe the matrix factorization algorithm that I used to tackle the Netflix problem, and how it may be implemented in parallel and scaled up on large datasets. Then, I will discuss the results of my experiments and an analysis of the prediction errors. When tested against the full probe set of 1,408,395 ratings, my implementation scored an RMSE of 0.9288, which is approximately a 2% improvement over Netflix's Cinematch algorithm.

2 Problem definition

Let u be the number of users, m be the number of films, and $\mathbf{R}$ be the $m \times u$ ratings matrix, where r_{ij} is the rating assigned by user j to film i. From the description of the Netflix problem in the previous section, we know that $\mathbf{R}$ is a incomplete matrix, and only about 1% of the ratings are known. The goal of a collaborative algorithm is then to use the known values to predict an unobserved value in $\mathbf{R}$ [3].

To estimate $\mathbf{R}$, we begin with a *latent factor model* [4] that attempts to identify features in films and users' preferences for these features. Using this model, we take the dot product of the features vector $\mathbf{m}_i$ and the preferences vector $\mathbf{u}_j$ to get an estimate of r_{ij}. Intuitively, if film i has a high value for a particular feature f that user j has a high preference for, then $m_{if} \times u_{jf}$ will contribute to a larger dot product, and hence give a higher estimate for r_{ij}. Note that, unlike content filtering, collaborative filtering techniques are domain-free and do not attempt to explicitly name these features [1].

With this model, we may decompose $\mathbf{R}$ as follows:

$$\mathbf{M} \times \mathbf{U} = \mathbf{R} \tag{1}$$

where $\mathbf{M}$ is the $m \times n_f$ features matrix, $\mathbf{U}$ is the $n_f \times u$ preferences matrix, and n_f is the number of features. Row i in $\mathbf{M}$ corresponds to the features vector $\mathbf{m}_i$, and column j in $\mathbf{U}$ corresponds to the preferences vector $\mathbf{u}_j$. The loss function is defined as the squared error [3]:

$$L(r, \mathbf{m}, \mathbf{u}) = (r - \mathbf{m}^\intercal \mathbf{u})^2 \tag{2}$$

The learning objective is to minimize the sum of the squared errors:

$$(\hat{\mathbf{M}}, \hat{\mathbf{U}}) = \text{argmin}_{\mathbf{M},\mathbf{U}} \sum_{(i,j) \in K} L(r_{ij}, \mathbf{m}_i, \mathbf{u}_j) \tag{3}$$

where K is the set of known ratings in $\mathbf{R}$. This also minimizes the RMSE, which may be found by taking the square root of the mean of the loss function over K.

3 Approach

To estimate $\mathbf{M}$ and $\mathbf{U}$, I used the Alternating Least Squares (ALS) algorithm. A basic version of the algorithm is as follows [4]:

1. Initialize $\mathbf{M}_0$ with random values between 0 and 1
2. $t \leftarrow 0$
3. Hold $\mathbf{M}_t$ constant, and find the maximum likelihood estimate of $\mathbf{U}_{t+1}$
4. Hold $\mathbf{U}_{t+1}$ constant, and find the maximum likelihood estimate of $\mathbf{M}_{t+1}$
5. $t \leftarrow t + 1$
6. Repeat from Step 3 until objective function converges

Note that in Step 3, when $\mathbf{M}_t$ is held constant, the maximum likelihood estimate of $\mathbf{U}_{t+1}$ becomes

$$\mathbf{U}_{t+1} = \text{argmin}_{\mathbf{U}} \sum_{(i,j)\in K} (r_{ij} - \mathbf{m}_i{}^\intercal \mathbf{u}_j)^2$$

which is just linear regression. The same concept is used in Step 4, where the roles of $\mathbf{M}$ and $\mathbf{U}$ in the linear regression are swapped. (A detailed proof may be found in other works [3].) On further inspection, it is clear that each column of $\mathbf{U}$ and each row of $\mathbf{M}$ may be solved in parallel. An implementation that exploits this parallelism will be described in a subsequent section.

3.1 Regularization

To avoid over-fitting, a regularization term may be included in the minimization [3]:

$$(\hat{\mathbf{M}}, \hat{\mathbf{U}}) = \text{argmin}_{\mathbf{M},\mathbf{U}} \sum_{(i,j)\in K} (r_{ij} - \mathbf{m}_i^\intercal \mathbf{u}_j)^2 + \left(\sum_j \|\Gamma_{u_j} \mathbf{u}_j\|_2^2 + \sum_i \|\Gamma_{m_i} \mathbf{m}_i\|_2^2 \right) \tag{4}$$

for some suitably chosen Tikhonov matrices $\{\Gamma_{u_j}\}$ and $\{\Gamma_{m_i}\}$. I experimented with the following methods:

$$\Gamma_{u_j} = \lambda \cdot \mathsf{diag}(\sqrt{n_{u_j}}), \quad \Gamma_{m_j} = \lambda \cdot \mathsf{diag}(\sqrt{n_{m_j}}) \tag{5}$$

$$\Gamma_{u_j} = \lambda \cdot \mathsf{diag}(\sqrt[4]{n_{u_j}}), \quad \Gamma_{m_j} = \lambda \cdot \mathsf{diag}(\sqrt[4]{n_{m_j}}) \tag{6}$$

where n_{u_j} and n_{m_i} are the number of known ratings for user j and film i respectively. In both cases, the regularization penalizes large parameters, but depends on the number of relevant known ratings [5].

Using Tikhonov matrices from (5) and Step 3 of the ALS algorithm as an example, the regularized linear regression solution becomes [5]:

$$\begin{aligned} \mathbf{u}_j &= \left[\mathbf{X_j}^\intercal \mathbf{X_j} + \Gamma_{u_j}{}^\intercal \Gamma_{u_j}\right]^{-1} \mathbf{X}_j{}^\intercal \mathbf{y} \\ &= \left[\mathbf{X_j}^\intercal \mathbf{X_j} + \lambda \cdot n_{u_j} \cdot \mathbf{I}\right]^{-1} \mathbf{X}_j{}^\intercal \mathbf{y} \end{aligned} \tag{7}$$

where $\mathbf{X_j}$ is the $n_{u_j} \times n_f$ matrix containing the feature vectors of the films rated by user j. The regularized linear regression solution for Step 4 is symmetrical.

3.2 Bias

The above model may also be extended in several ways. One extension that I tried was to add biases. Some films, such as *Titanic* (1997), may be generally more popular than other films, and some users may be generally more critical than average. These effects are independent of the film's characteristics and the user's preferences, and are therefore not captured by $\mathbf{m_i}^\intercal \mathbf{u_j}$. For example, a user A may have the exact same preferences as another user B and thus rank all films in the same order, but the average rating assigned by A to all films may not be equal to the average rating assigned by B to all films. Ideally, the recommender should capture this information and learn the same preference vectors for A and B while noting the biases. The prediction function becomes [4]:

$$\hat{r}_{ij} = \mu + b_{m_i} + b_{u_j} + \mathbf{m}_i{}^\intercal \mathbf{u}_j \tag{8}$$

where

- μ is the global mean rating
- b_{m_i} is the bias for film i ($\mathbf{b}_m$ is the vector of film biases)
- b_{u_j} is the bias for user j ($\mathbf{b}_u$ is the vector of user biases)

We may rewrite the loss function as

$$L^{bias}(r, \mathbf{m}, \mathbf{u}, b_m, b_u) = (r_{ij} - \mu - b_m - b_u - \mathbf{m}^\intercal \mathbf{u})^2 \tag{9}$$

Using (5) and (6), the objective functions become

$$\sum_{(i,j)\in K} L^{bias}(r, \mathbf{m}_i, \mathbf{u}_j, b_{m_i}, b_{u_j}) + \lambda \left(\sum_j n_{u_j} (\|\mathbf{u}_j\|_2^2 + {b_{u_j}}^2) + \sum_i n_{m_i} (\|\mathbf{m}_i\|_2^2 + {b_{m_i}}^2) \right) \tag{10}$$

$$\sum_{(i,j)\in K} L^{bias}(r, \mathbf{m}_i, \mathbf{u}_j, b_{m_i}, b_{u_j}) + \lambda \left(\sum_j \sqrt{n_{u_j}} (\|\mathbf{u}_j\|_2^2 + {b_{u_j}}^2) + \sum_i \sqrt{n_{m_i}} (\|\mathbf{m}_i\|_2^2 + {b_{m_i}}^2) \right) \tag{11}$$

Again using Step 3 of the ALS algorithm as an example, the maximum likelihood of the bias for user j may be found by prepending $\mathbf{X}_j$ with a column containing 1's.

$$\begin{bmatrix} 1 & m_{1,1} & m_{1,2} & \cdots & m_{1,n_f} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & m_{n_{u_j},1} & m_{n_{u_j},2} & \cdots & m_{n_{u_j},n_f} \end{bmatrix} \begin{bmatrix} b_{u_j} \\ u_{j,1} \\ u_{j,2} \\ \vdots \\ u_{j,n_f} \end{bmatrix} = \begin{bmatrix} r_{1,j} \\ \vdots \\ r_{n_{u_j},j} \end{bmatrix}$$

Figure 1: Solving for $\mathbf{u}_j$ and b_{u_j}

4 Experiments

I implemented the ALS algorithm in MatLab on a smaller dataset and scaled it up using GraphLab, a new parallel framework for machine learning applications [6].

4.1 Pre-processing

The probe set was separated from the training set and used for testing. Moreover, a smaller dataset was obtained by a uniform sampling of the ratings of the first 3000 users and the first 3000 films. This yielded data sizes shown in Table 1.

	Small Dataset	Full Dataset
Training	99,072,112	232,619
Probe	1,408,395	21,755

Table 1: Number of ratings in each set

4.2 Post-processing

Since the Netflix ratings are known to be between 1 and 5, an easy way to reduce the RMSE is to truncate all predicted ratings to $[1, 5]$.

Zhou, et. al. also suggested in their work that it may be beneficial to shift all predicted values by the difference between the mean of the training set and the mean of the test set [3]. This helps to correct for the possible difference in ratings distribution between the two sets.

4.3 MatLab implementation

Using MatLab, ALS was implemented with and without biases using a simple `parfor` loop that solves each column of $\mathbf{U}$ in parallel, and each row of $\mathbf{M}$ in parallel. in each case, the function returns

a predictor that was tested against the probe set. Figure 2 shows that the training RMSE converges in about 20 to 30 iterations. Similarly, the objective functions from 4, 10, and 11 converges in about 30 to 40 iterations.

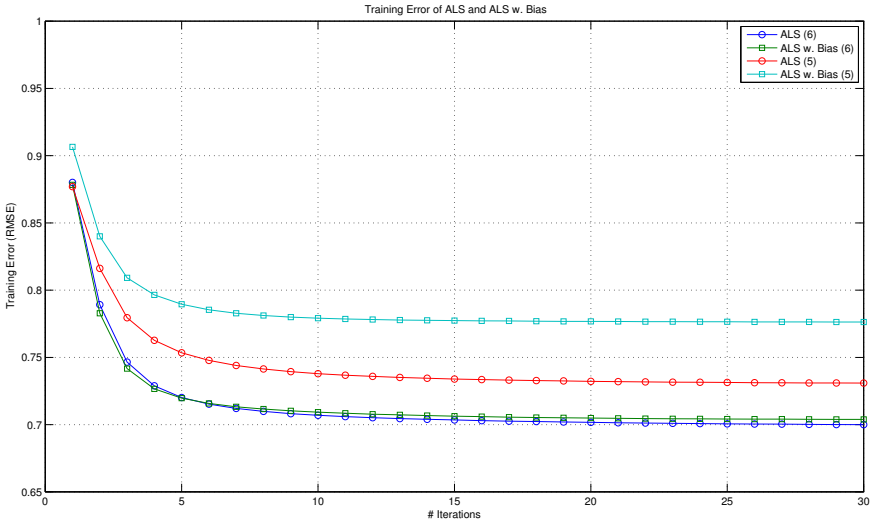

Figure 2: Convergence using training RMSE w. $n_f = 20, \lambda = 1$. (5) and (6) refer to the type of regularization used.

Table 2 shows the values for λ that were selected using 5-fold cross validation. The cross-validation results are shown in Figure 3 and 4.

	Reg. (5)	Reg. (6)
ALS	0.10	1.20
ALS w. Bias	0.12	1.65

Table 2: Values selected for λ using 5-fold CV

Using regularized ALS, I was unable to get any significant improvement when the number of features was varied from 20 to 40. Using $n_f = 20$, the RMSE obtained on the probe set for each configuration may be found in Table 3.

	Reg. (5)	Reg. (6)
ALS	0.9254	0.9505
ALS w. Global Mean Correction	<u>0.9157</u>	0.9305
ALS Bias	0.9258	<u>0.9211</u>
ALS Bias w. Global Mean Correction	0.9470	0.9360

Table 3: ALS RMSE on the small probe set. The best two are underlined.

4.4 GraphLab implementation

To scale up to a 100 million ratings, I reimplemented the ALS algorithm using GraphLab, a new parallel framework for machine learning applications [6]. The sparse ratings matrix $\mathbf{R}$ is represented as a graph that contains users and films as vertices. A user vertex u is adjacent to a film vertex m if and only if u has assigned a rating to film m, and the undirected edge (u, m) contains the corresponding rating. At each iteration, the scheduler executes an update function on each vertex which

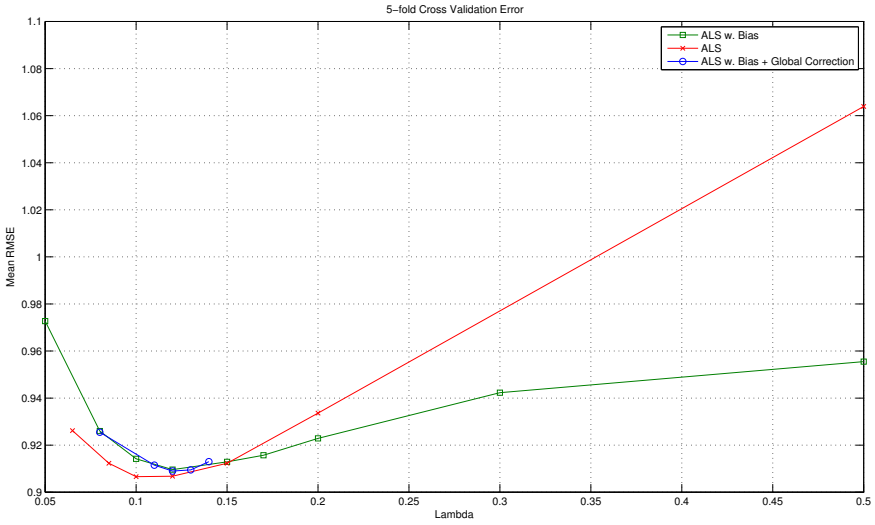

Figure 3: Average RMSE from 5-fold cross validation using (5)

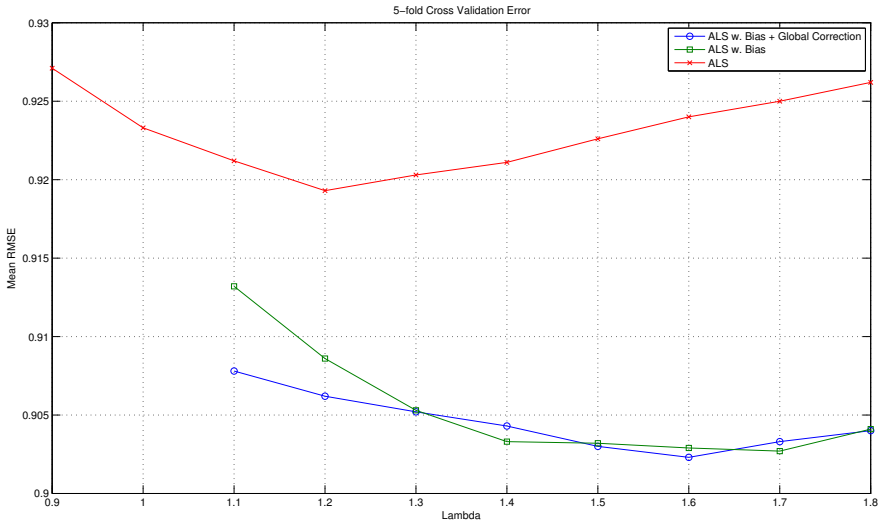

Figure 4: Average RMSE from 5-fold cross validation using (6)

1. gathers the latent factors of the vertex's neighbours,
2. gathers the ratings on the incident edges,
3. calculates the maximum likelihood estimate of the latent factors for the vertex, and
4. reschedules neighbour vertices for updating if convergence conditions are not met

This allows the algorithm to find the latent factors of the vertices in parallel, greatly shortening the training time. Using the best results from the MatLab implementation, I scaled up on GraphLab to the entire dataset of 100 million ratings on a 16-core AMD Opteron with 64Gb of memory. The RMSE scored by my GraphLab implementation of ALS is shown in Table 4 [1]

[1]My implementation was written in Java, using the JNI interface that I developed for GraphLab. Parts of the algorithm were adapted from Gonzalez's C++ implementation [6].

	Reg. (5)	Reg. (6)
ALS w. Global Mean Correction	0.9317	0.9348
ALS w. Bias	0.9444	<u>0.9288</u>

Table 4: ALS RMSE on full probe set. The best RMSE is underlined.

The best RMSE is about a 2% improvement over Cinematch's RMSE.

4.5 RMSE analysis

Figure 5 shows a graph of the RMSE of each film in probe set against the number of known ratings for that film in the training set. Intuitively, we should expect the RMSE to decrease as the number of known ratings increases, but the graph also shows that some films are a lot more predictable than average (and, conversely, some films are a lot harder to predict than average [7].)

If we focus our attention on films with more than 50,000 ratings and look for outliers, a particular cluster of films (marked by the black circle on Figure 5) stands out. These films have, on average, a lower RMSE than any other film with more than 50,000 ratings. Interestingly, all of them belong to the *Lord of the Rings* trilogy: *The Fellowship of the Ring*, *The Two Towers*, and *The Return of the King*. An explanation for such a phenomenon could be as follows: users who watched the third sequel of the trilogy are likely to have watched the first two and enjoyed them, and therefore likely to have rated all three films (which explains the similarity in number of ratings) and given them the same ratings (which explains the similarity in RMSE.) This suggests that the ALS algorithm may be blended with some clustering strategy to exploit such obvious symmetries.

Similarly, we may identify two films that have higher RMSEs than any other film with more than 50,000 ratings. The first film, marked by the purple circle, is *Fahrenheit 9/11*, and the second film, marked by the red circle, is *Napoleon Dynamite*. Slee found similar results in his analysis of the RMSE of the winning entry for the Netflix Prize [7], and noted that there could be a subset of films on Netflix that are politically or culturally polarizing, which makes it difficult for most algorithms, including ALS, to predict accurately. However, it might be beneficial to identify such outliers during training (possibly by selecting films with unusually large training errors, or via some clustering algorithm) and separate them from the probe set during prediction. A more specialized algorithm may then be used to tackle such films.

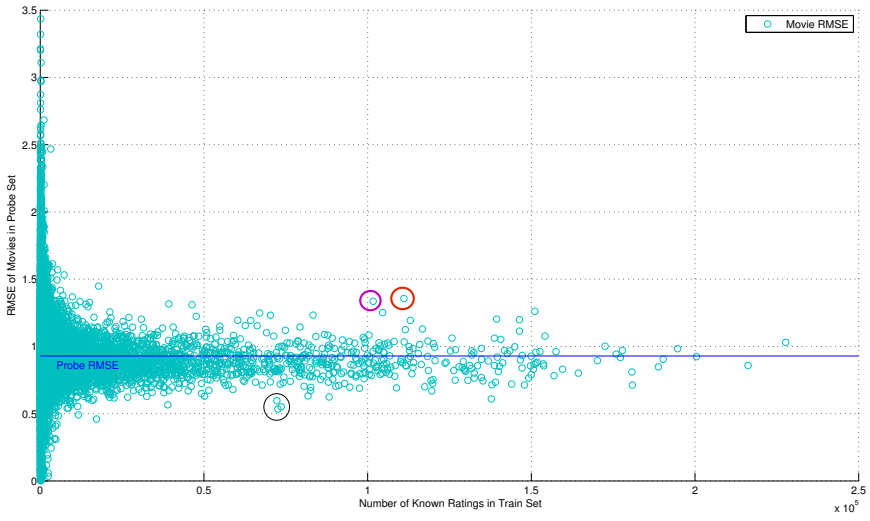

Figure 5: RMSE of films from the probe set versus the number of known ratings in the training set. Black circle contains films from the *Lord of The Rings* trilogy, whereas the purple and red circles contain *Fahrenheit 9/11* and *Napoleon Dynamite* respectively.

Several other works reported positive results from blending matrix factorization algorithms with neighbourhood search algorithms [8]. I attempted a naive variation of the K-Nearest Neighbour algorithm, where a rating is predicted by taking the weighted average of the k most similar films that the user has rated. The distance between two films, m_i and m_j, was defined as the cosine of the angle between the ratings vectors of the films, as follows:

$$\mathsf{dist}(m_i, m_j) = \frac{\mathbf{r}_i \cdot \mathbf{r}_j}{\|\mathbf{r}_i\| \|\mathbf{r}_j\|} \tag{12}$$

The best RMSE I could get with this algorithm on the small dataset was 2.135, which is worse than simply using the mean rating of each film. Since the effectiveness of neighbourhood search algorithms is highly sensitive to the size of the neighbourhood, a larger dataset may have improved my results. However, calculating the distance matrix requires $O(\binom{m}{2})$ time and space, which is hard to scale up to very large datasets.

5 Conclusions

The best RMSE was achieved using my implementation of ALS was 0.9288, which represents a 2% improvement over Netflix's Cinematch algorithm. However, an analysis of the RMSE revealed several opportunities for improvement. A blend with neighbourhood search techniques could be beneficial.

A recommender system may also use the RMSE of predicted ratings for films (possibly calculated using cross-validation) as a measure of confidence. Recommendations may then be selected using both confidence and predicted ratings. For example, outliers (such as cult films and political films) may be avoid using this technique, until more data about the user is collected. An e-commerce site with sufficiently large product base might find this to be more beneficial in the long run, since fans of such products are likely to search for the products on their own, and are thus unaffected by recommendation lists.

References

[1] G. Linden, B. Smith, and J. York. Amazon.com recommendations - item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[2] Netflix. Netflix prize. `http://www.netflixprize.com/`, October 2006.

[3] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Proc. 4th Int'l Conf. Algorithmic Aspects in Information and Management, LNCS 5034*, pages 337–348. Springer, 2008.

[4] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems, 2009.

[5] A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-posed problems*. W.H. Winston, 1977.

[6] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Graphlab: A new framework for parallel machine learning. *CoRR*, abs/1006.4990, 2010.

[7] Tom Slee. Netflix prize: Was the napoleon dynamite problem solved? `http://whimsley.typepad.com/whimsley/2009/10/netflix-prize-was-the-napoleon-dynamite-problem-solved.html`, October 2009.

[8] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM.