Variational Bayesian Approach to Movie Rating Prediction

Yew Jin Lim
School of Computing
National University of Singapore
limyewji@comp.nus.edu.sg

Yee Whye Teh
Gatsby Computational Neuroscience Unit
University College London
ywteh@gatsby.ucl.ac.uk

ABSTRACT

Singular value decomposition (SVD) is a matrix decomposition algorithm that returns the optimal (in the sense of squared error) low-rank decomposition of a matrix. SVD has found widespread use across a variety of machine learning applications, where its output is interpreted as compact and informative representations of data. The Netflix Prize challenge, and collaborative filtering in general, is an ideal application for SVD, since the data is a matrix of ratings given by users to movies. It is thus not surprising to observe that most currently successful teams use SVD, either with an extension, or to interpolate with results returned by other algorithms. Unfortunately SVD can easily overfit due to the extreme data sparsity of the matrix in the Netflix Prize challenge, and care must be taken to regularize properly.

In this paper, we propose a Bayesian approach to alleviate overfitting in SVD, where priors are introduced and all parameters are integrated out using variational inference. We show experimentally that this gives significantly improved results over vanilla SVD. For truncated SVDs of rank 5, 10, 20, and 30, our proposed Bayesian approach achieves 2.2% improvement over a naïve approach, 1.6% improvement over a gradient descent approach dealing with unobserved entries properly, and 0.9% improvement over a maximum a posteriori (MAP) approach.

Categories and Subject Descriptors

I.2.6 [**Machine Learning**]: Engineering applications—*applications of techniques*

General Terms

Experimentation, Algorithms

Keywords

Netflix Prize, machine learning, SVD, Variational Inference

1. INTRODUCTION

The Netflix Prize [2] is a competition organized by Netflix, an on-line movie subscription rental service, to create a recommender system that predicts movie preferences based on user rating history. For the competition, Netflix provides a large movie rating dataset consisting of over 100 million ratings (and their dates) from approximately 480,000 randomly-chosen users and 18,000 movies. The data were collected between October, 1998 and December, 2005 and represent the distribution of all ratings Netflix obtained during this time period. Given this dataset, the task is to predict the actual ratings of over 3 million unseen ratings from these same users over the same set of movies.

One common machine learning approach to this collaborative filtering problem is to represent the data as a sparse $I \times J$ matrix M consisting of ratings given by the I users to the J movies [5]. A low-rank decomposition of M is then found, $M \approx UV^\top$, where $U \in \mathbb{R}^{I\times n}$, $V \in \mathbb{R}^{J\times n}$ with $n \ll I, J$. The matrices U and V can be viewed as compact and informative representations of the users and movies respectively, and the predicted rating given by a user to a movie is given by the corresponding entry in $UV^\top$.

Singular value decomposition (SVD) is a popular matrix decomposition algorithm that returns such a low-rank decomposition which is optimal in the sense of squared error. Unfortunately SVD only works for fully observed matrices, while extensions of SVD which can handle partially observed matrices[1] can easily overfit due to the extreme data sparsity of the matrix in the Netflix Prize competition, and care must be taken to regularize properly.

In this paper, we propose a Bayesian approach to alleviate overfitting in SVD. Priors are introduced and all parameters are integrated out using Variational Bayesian inference [1]. Our technique can be efficiently implemented and has comparable run time and memory requirements to an efficiently implemented SVD. We show experimentally that this gives significantly improved results over vanilla SVD.

The outline of this paper is as follows: Section 2 describes the Netflix Prize and outlines the standard SVD-based matrix decomposition techniques currently used in collaborative filtering problems. Next, we present a Bayesian formulation for matrix decomposition and a maximum a posteriori solution in Section 3. In Section 4, we derive and explain our proposed Variational Bayesian approach. We then show and discuss experimental results in Section 5. Finally, we discuss a few related models in 6 and conclude in Section 7.

[1]In this paper we refer to these extensions for partially observed matrices as SVD algorithms as well.

2. MATRIX DECOMPOSITION

Netflix provides a partially observed rating matrix M with observed entries having values between 1 and 5, inclusive. M has $I = 480,189$ rows, corresponding to users, and $J = 17,770$ columns, corresponding to movies. There are $K = 100,480,507$ observed entries in M. As part of the training data, Netflix designated a set of 1,408,395 ratings to be validation data. For all algorithms tested in this paper, we withheld this validation set during training and used the remaining data consisting of $99,072,112$ entries as training data. We then tested the RMSE based on predictions for the validation data. In Section 7 we report our results on the official test set. As a baseline, Netflix's own system, Cinematch, achieved a RMSE of 0.9474 on the validation data.

We are interested in the problem of finding a low-rank approximation to M. That is, find two matrices $U \in \mathbb{R}^{I\times n}$ and $V \in \mathbb{R}^{J\times n}$, where n is small, such that $M \approx UV^\top$. Using the squared loss, we formulate this problem as one of minimizing the objective function:

$$f(U,V) = \sum_{(ij)} (u_i v_j^\top - m_{ij})^2 \tag{1}$$

where (ij) ranges over pairs of user/movie indices such that the user rated that movie in the training set, u_i, v_j are rows of U and V, and m_{ij} is the i,j-th entry of M.

2.1 Singular Value Decomposition

Singular value decomposition (SVD) is an efficient method of finding the optimal solution to (1), for the case when the rating matrix M is fully observed [8]. SVD works by factorizing the fully observed matrix M into $M = \tilde{U}\tilde{\Sigma}\tilde{V}^T$, where $\tilde{U}$ is an $I \times I$ orthonormal matrix, $\tilde{\Sigma}$ is an $I \times J$ diagonal matrix with non-negative and decreasing diagonal entries and $\tilde{V}$ is a $J \times J$ orthonormal matrix. The non-negative diagonal entries of $\tilde{\Sigma}$ are called the singular values of M. Such a decomposition into $\tilde{U}$, $\tilde{\Sigma}$ and $\tilde{V}$ is unique (up to rotations of the corresponding columns of $\tilde{U}$ and $\tilde{V}$ if there are identical singular values), and is called the singular value decomposition of M. The singular values can be viewed as the importance of the corresponding features (columns of $\tilde{U}$ and $\tilde{V}$). A low-rank approximation to M can be obtained by keeping only the top n singular values along with first n columns in $\tilde{U}$ and $\tilde{V}$. Specifically, let $U = \tilde{U}_{1:I,1:n}\tilde{\Sigma}_{1:n,1:n}$ and $V = \tilde{V}_{1:J,1:n}$. Then it can be shown that the low-rank approximation $M \approx UV^\top$ minimizes the objective (1).

The time complexity of the standard algorithm to compute SVD is $O(\max(I^3, J^3))$, which makes using it computationally infeasible with the size of the Netflix Prize dataset. Fortunately, it is possible to approximate the rank-n SVD with successive rank-1 updates in $O(I \times J \times n)$ time [3]. As rank-1 updates involve incrementally learning one column c of M at a time, this also reduces space requirements to storing a single column of M in main memory (as opposed to the whole matrix M) when computing the SVD. Thus an optimal low-rank approximation can be efficiently computed if the matrix M is fully observed.

The main issue with using such a vanilla SVD is that it assumes a fully observed matrix M. In the next two subsection we review two baseline approaches to dealing with a sparsely observed M, one based on EM, and another based on directly optimizing (1) in a greedy fashion.

2.2 Expectation Maximization

As the matrix M contains missing entries, standard SVD needs to be adapted to handle missing values appropriately. One method is to use a simple expectation-maximization (EM) algorithm that fills in the missing values with predictions from the low-rank reconstruction from the previous iteration [10].

Assume that we have an algorithm to compute optimal (in terms of minimizing (1)) rank n decomposition of a completely observed matrix, say one which uses the rank n truncation of an SVD. Call this algorithm SVD_n. The EM approach uses SVD_n to impute successively better estimates of the missing matrix entries during training. Formally, we introduce a $I \times J$ binary matrix W, where w_{ij} has value 1 if m_{ij} is observed, and value 0 otherwise. The EM procedure then iterates between the following until convergence:

$$\text{E-Step}: \quad X = W \cdot M + (1 - W) \cdot \tilde{M} \tag{2}$$

$$\text{M-Step}: \quad [U,V] = \mathrm{SVD}_n(X)$$

$$\tilde{M} = UV^\top \tag{3}$$

where $\cdot$ is the *Hadamard product* given by $(A \cdot B)_{ij} = a_{ij}b_{ij}$.

2.3 Greedy Residual Fitting

Another popular approach which approximately minimizes (1) when M is partially observed is to greedily fit columns of U and V iteratively. Specifically, if U', V' is a rank $n-1$ approximation to M, we obtain a rank n approximation by fitting a rank 1 approximation to the residual matrix $M - U'V'^\top$, and appending the new columns to U' and V' respectively. In this paper we find the rank 1 approximation by minimizing (1) (plus a L_2 regularization) by gradient descent:

$$\delta_{ij} := m_{ij} - u'_i v'^\top_j \tag{4}$$

$$u_{in} := u_{in} + \alpha(\delta_{ij} v_{jn} - \lambda u_{in}) \tag{5}$$

$$v_{jn} := v_{jn} + \alpha(\delta_{ij} u_{in} - \lambda v_{jn}) \tag{6}$$

where α is the step size and λ is the regularization parameter. We iterate through (5) and (6) until a desired threshold for convergence is reached. Then the newly found columns u_n and v_n are appended to U' and V' respectively. We call this method greedy residual fitting (GRF).

3. BAYESIAN FORMULATION

We turn to statistical machine learning methods to perform regularization by introducing prior knowledge into our model. In particular, we shall recast our objective function (1) using a probabilistic model with entries of M being observations and U, V being parameters, and introduce regularization via placing priors on the parameters. Specifically, we place the following likelihood model on M:

$$m_{ij}|U,V \sim \mathcal{N}(u_i v_j^\top, \tau^2) \tag{7}$$

for each observed entry (i,j) of M, where τ^2 is variance of observation noise about the mean $u_i v_j^\top$.

The probability density at m_{ij} is:

$$p(m_{ij}|U,V) = \frac{1}{\sqrt{2\pi\tau^2}} \exp\left(-\frac{1}{2}\frac{(m_{ij} - u_i v_j^\top)^2}{\tau^2}\right) \tag{8}$$

We place independent priors on each entry in U and V, say, $u_{il} \sim \mathcal{N}(0, \sigma_l^2)$ and $v_{jl} \sim \mathcal{N}(0, \rho_l^2)$. so the densities of

U and V are:

$$p(U) = \prod_{i=1}^{I}\prod_{l=1}^{n} \frac{1}{\sqrt{2\pi\sigma_l^2}} \exp\left(-\frac{1}{2}\frac{u_{il}^2}{\sigma_l^2}\right) \tag{9}$$

$$p(V) = \prod_{j=1}^{J}\prod_{l=1}^{n} \frac{1}{\sqrt{2\pi\rho_l^2}} \exp\left(-\frac{1}{2}\frac{v_{jl}^2}{\rho_l^2}\right) \tag{10}$$

Note that the prior variances σ_l^2, ρ_l^2 depends on the column l in U and V respectively. We will show how to optimize these prior variances in a later section.

This completes the model, and we now need to compute or approximate the posterior:

$$p(U,V|M) = \frac{p(M|U,V)p(U)p(V)}{p(M)} \tag{11}$$

3.1 Maximum A Posteriori

The method of maximum a posteriori (MAP) estimation can be used to obtain a point estimate of the posterior (11) based on observed data. It is similar to maximum likelihood (ML), but as the model incorporates a prior distribution, MAP estimation can be seen as a regularization of ML estimation. MAP methods approximate the posterior by its *mode*:

$$p(U,V|M) \approx \arg\max_{U,V} \frac{p(M|U,V)p(U,V)}{p(M)} \tag{12}$$

In this paper, we simplify the approximation by assuming that U and V are independent in the posterior before solving (12). The eventual derivations turns out to be similar to our derivations for VB inference (refer to Section 5.4).

4. VARIATIONAL BAYES

The variational free energy [1] of the model introduced in Section 3 is defined as:

$$\mathcal{F}(Q(U,V)) = \mathbb{E}_{Q(U,V)}[\log p(M,U,V) - \log Q(U,V)] \tag{13}$$

The variational free energy can be shown to be a lower bound on the log likelihood $p(M)$ for all distributions $Q(U,V)$, since we have

$$\mathcal{F}(Q(U,V)) = \mathbb{E}_{Q(U,V)}[\log p(M) + \log p(U,V|M) - \log Q(U,V)] \tag{14}$$

$$= \log p(M) - \mathrm{KL}(Q(U,V)\|p(U,V|M)) \leq \log p(M) \tag{15}$$

where $\mathbb{E}_q[f(x)]$ is the expectation of the function $f(x)$ with respect to the distribution $q(x)$ and $KL(q\|p) = \int q(x)\log\frac{q(x)}{p(x)}\,dx$ is the Kullback-Leibler divergence from distribution q to distribution p and is always non-negative, being zero exactly when $q(x) = p(x)$ almost surely.

Maximizing the lower bound on $\log p(M)$, note that the optimum is achieved at $Q(U,V) = p(U,V|M)$ the posterior. To see this, simply differentiate (13) with respect to $Q(U,V)$ and solve subject to $\int Q(U,V)\,dU\,dV = 1$. Noting that this is intractable in practice, we maximize $\mathcal{F}(Q(U,V))$ subject to the variational approximation $Q(U,V) = Q(U)Q(V)$. This gives the variational approximate inference procedure.

We therefore see that while MAP estimation incorporates a prior distribution, MAP estimates are only point estimates, and does not account for variance and uncertainty observed in the empirical data. On the other hand, VB methods strive to avoid overfitting by taking into account the whole posterior distribution.

Plugging in the model for $p(M,U,V)$, we have

$$\begin{aligned}
&\mathcal{F}(Q(U)Q(V)) \\
=&E_{Q(U)Q(V)}\Bigg[-\frac{1}{2}\left(\sum_{i=1}^{I}\sum_{l=1}^{n}\log(2\pi\sigma_l^2) + \frac{u_{il}^2}{\sigma_l^2}\right) \\
&-\frac{1}{2}\left(\sum_{j=1}^{J}\sum_{l=1}^{n}\log(2\pi\rho_l^2) + \frac{v_{jl}^2}{\rho_l^2}\right) \\
&-\frac{1}{2}\left(\sum_{(ij)}\log(2\pi\tau^2) + \frac{(m_{ij} - u_i^\top v_j)^2}{\tau^2}\right) \\
&-\log Q(U) - \log Q(V)\Bigg] \\
=&-\frac{K}{2}\log(2\pi\tau^2) - \frac{I}{2}\sum_{l=1}^{n}\log(2\pi\sigma_l^2) - \frac{J}{2}\sum_{l=1}^{n}\log(2\pi\rho_l^2) \\
&-\frac{1}{2}\sum_{l=1}^{n}\left(\frac{\sum_{i=1}^{I}\mathbb{E}_{Q(U)}[u_{il}^2]}{\sigma_l^2} + \frac{\sum_{j=1}^{J}\mathbb{E}_{Q(V)}[v_{jl}^2]}{\rho_l^2}\right) \\
&-\frac{1}{2}\sum_{(ij)}\frac{\mathbb{E}_{Q(U)Q(V)}[(m_{ij} - u_i v_j^\top)^2]}{\tau^2}
\end{aligned} \tag{16}$$

To maximize $\mathcal{F}(Q(U)Q(V))$ we optimize one keeping the other fixed, and iterate until convergence. To maximize with respect to $Q(U)$ with $Q(V)$ held fixed, we differentiate $\mathcal{F}(Q(U)Q(V))$ with respect to $Q(U)$ and solve for $Q(U)$ by setting the derivatives to 0 subject to the constraint that $\int Q(U)\,dU = 1$. This gives,

$$Q(U) \propto \prod_{i=1}^{I}\exp\left(-\frac{1}{2}(u_i - \overline{u_i})^\top\Phi_i^{-1}(u_i - \overline{u_i})\right)$$

$$\Phi_i = \left(\begin{pmatrix} \frac{1}{\sigma_1^2} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{\sigma_n^2}\end{pmatrix} + \sum_{j\in N(i)}\frac{\Psi_j + \overline{v_j}\overline{v_j}^\top}{\tau^2}\right)^{-1} \tag{17}$$

$$\overline{u_i} = \Phi_i\left(\sum_{j\in N(i)}\frac{m_{ij}\overline{v_j}}{\tau^2}\right) \tag{18}$$

where $N(i)$ is the set of j's such that m_{ij} is observed, $\Phi_i$is the covariance of u_i in $Q(U)$, $\overline{u_i}$ is the mean of u_i, Ψ_j is the covariance of v_j in $Q(V)$, and $\overline{v_j}$ of v_j.

Thus the optimal $Q(U)$ with $Q(V)$ fixed is independent across i, with $Q(u_i)$ being Gaussian with covariance Φ_i and mean $\overline{u_i}$. Similarly one derives that:

$$Q(V) \propto \prod_{j=1}^{J}\exp\left(-\frac{1}{2}(v_j - \overline{v_j})^\top\Psi_j^{-1}(v_j - \overline{v_j})\right) \tag{19}$$

decomposes as independent distributions $Q(v_j)$ across j, with $Q(v_j)$ being Gaussian with covariance and mean given by:

$$\Psi_j = \left(\begin{pmatrix} \frac{1}{\rho_1^2} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{\rho_n^2} \end{pmatrix} + \sum_{i \in N(j)} \frac{\Phi_i + \overline{u_i} \overline{u_i}^\top}{\tau^2} \right)^{-1} \tag{20}$$

$$\overline{v_j} = \Psi_j \left(\sum_{i \in N(j)} \frac{m_{ij} \overline{u_i}}{\tau^2} \right) \tag{21}$$

This completes the derivation of the variational algorithm, which simply iterates the updates (17), (18), (20) and (21) until convergence.

4.1 Further Implementational Details

In summary, the variational algorithm iterates the following until convergence:

1. Update $Q(u_i)$ for $i = 1, \ldots, I$ using (17) and (18).
2. Update $Q(v_j)$ for $j = 1, \ldots, J$ using (20) and (21).

The computation costs are as follows. For each step we need to go through the observed ratings once. For each user and each movie the dominating computation cost are to collect the values of the matrix to be inverted, followed by the matrix inversion itself. The time complexity is $O(K + I \times n^3 + J \times n^3)$, where K is the number of observed entries. For small values of n, this is much more efficient than SVD which takes $O(I \times J \times n)$ time.

The information that needs to be stored from iteration to iteration are $\Phi_i, \overline{u_i}, \Psi_j$ and $\overline{v_j}$ for each i and j. Note that the storage size of Φ_i is unacceptable since there are $> 400,000$ users, and each Φ_i is of size $n \times n$. To mitigate this, we instead opt to not store Φ_i at all. The idea is that as soon as Φ_i and $\overline{u_i}$ are computed, these are added to the matrix needed to be inverted to compute Ψ_j and $\overline{v_j}$. After we add this in, we need not store Φ_i anymore and can discard it. The final algorithm keeps track of $\Psi_j, \overline{v_j}$ and $\overline{u_i}$ ($\overline{u_i}$ actually need not be kept tracked of, but is useful when we want to predict new ratings), and iterates through the following until convergence:

1. Initialize S_j and t_j for $j = 1, \ldots, J$:

$$S_j \leftarrow \begin{pmatrix} \frac{1}{\rho_1^2} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{\rho_n^2} \end{pmatrix} \qquad t_j \leftarrow 0$$

2. Update $Q(u_i)$ for $i = 1, \ldots, I$:

   (a) Compute Φ_i and $\overline{u_i}$:

$$\Phi_i \leftarrow \left(\begin{pmatrix} \frac{1}{\sigma_1^2} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{\sigma_n^2} \end{pmatrix} + \sum_{j \in N(i)} \frac{\Psi_j + \overline{v_j} \overline{v_j}^\top}{\tau^2} \right)^{-1}$$

$$\overline{u_i} \leftarrow \Phi_i \left(\sum_{j \in N(i)} \frac{m_{ij} \overline{v_j}}{\tau^2} \right)$$

   (b) Update S_j and t_j for $j \in N(i)$, and discard Φ_i:

$$S_j \leftarrow S_j + \frac{\Phi_i + \overline{u_i} \overline{u_i}^\top}{\tau^2} \qquad t_j \leftarrow t_j + \frac{m_{ij} \overline{u_i}}{\tau^2}$$

3. Update $Q(v_j)$ for $j = 1, \ldots, J$:

$$\Psi_j \leftarrow S_j^{-1} \qquad \overline{v_j} \leftarrow \Psi_j t_j$$

Finally, at test time, we predict a previously unobserved entry m_{ij} simply by $\overline{u_i} \overline{v_j}^\top$.

4.2 Learning the Variances

It is also important to try to learning the variances σ_l^2, ρ_l^2 and τ^2. This turns out to be easy. We simply differentiate (16) with respect to these variances, set the derivatives to zero, and solve for the optimal σ_l^2, ρ_l^2 and τ^2. These updates are:

$$\sigma_l^2 = \frac{1}{I-1} \sum_{i=1}^{I} (\Phi_i)_{ll} + \overline{u_{il}}^2 \tag{22}$$

$$\rho_l^2 = \frac{1}{J-1} \sum_{j=1}^{J} (\Psi_j)_{ll} + \overline{v_{jl}}^2 \tag{23}$$

$$\begin{aligned} \tau^2 = \frac{1}{K-1} \sum_{(ij)} & m_{ij}^2 - 2 m_{ij} \overline{u_i} \overline{v_j}^\top \\ & + \mathrm{Tr}[(\Phi_i + \overline{u_i} \overline{u_i}^\top)(\Psi_j + \overline{v_j} \overline{v_j}^\top)] \end{aligned} \tag{24}$$

where Tr is the trace operator and $\mathrm{Tr}(AB) = \sum_{ij} A_{ij} B_{ij}$ for two symmetric matrices A and B.

Note that there is a degree of redundancy between U and V. If we multiply a column of U by a factor a, and divide the corresponding column of V by a, the result will be identical. Thus to remove this degree of redundancy, we keep ρ_l^2 fixed with values $\rho_l^2 = \frac{1}{n}$ while σ_l^2 is initialized at $\sigma_l^2 = 1$ and allowed to be learned using (22). The reason for starting with these initial values is because if we draw U and V from the prior, then the variance of $u_i v_j^\top$ will be exactly 1. This puts the parameters of the model in a regime where it will behave reasonably. Similarly τ^2 should be initialized to 1.

It is possible that as the algorithm runs, the value of σ_l^2 for some l might become very small. This should happen when the algorithm decides that it does not have use for the lth column of U, thus it sets it to zero. In such a case, we can actually remove the lth column of U and V. This is called automatic relevance determination (ARD) [7] and provides computational savings - however, in our own usage of the algorithm for low-rank matrix decompositions of up to rank 100, this effect never materialized.

5. EXPERIMENTS

In our actual entry to the Netflix Prize, we used VB inference to compute rank 100 matrix decompositions. However, each of these decompositions required several days of computation on a PowerMac Dual 1.8 Ghz, and we did not have the necessary resources to repeat the same rank 100 matrix decomposition using the other algorithms. In order to compare the performance of the algorithms, we decided to perform low-rank matrix decompositions on the Netflix Prize dataset.

5.1 Implementation Details

For the Netflix Prize, we implemented the techniques mentioned in earlier sections using `Python` and `SciPy/Numpy`. The `scipy.linalg.svd` function was used to compute SVD of matrices when needed.

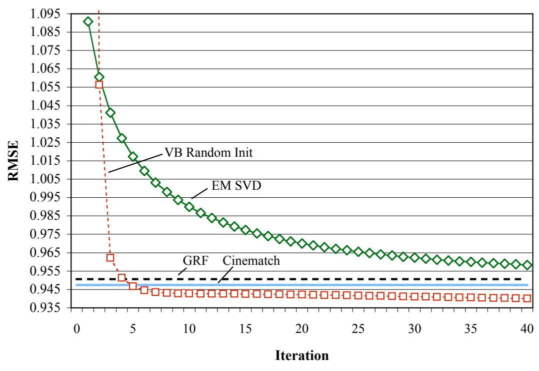

Figure 1: Performance of EM SVD vs VB Random Init, GRF and Cinematch with rank 5 approximation. The x-axis shows the number of iterations the algorithm was ran for, and the y-axis shows the RMSE on validation data (lower is better).

For GRF, the feature values are initialized to 0.1, weights are updated using a learning rate of 0.001 and the regularization parameter λ is set to 0.015. We used publicly available source code[2] that implements GRF. The algorithm updates each set of feature values using (5) and (6) for at least 100 epochs, or the number of passes through the training dataset. These updates continue until the improvement on RMSE from the last iteration falls below 0.0001. The current set of feature values are then fixed and the algorithm continues to the next set of feature values.

5.2 EM

We used EM SVD approach to obtain rank 5 matrix decompositions. We initialized the missing values of the rating matrix in the first iteration to $\overline{v_j} + \overline{w_i}$, where $\overline{v_j}$ is the average rating for movie j, and $\overline{w_i}$ is the average offset from movie averages that user i had given in the training data.

In Figure 1, we see that EM SVD had not converged after 40 iterations, and achieved a RMSE on the validation set of 0.9583 after 40 iterations. This is higher than the RMSEs achieved by Cinematch and GRF, which are 0.9474 and 0.9506, respectively. In contrast, our proposed method of using VB inference, even when initialized randomly, converged after less than 10 iterations and achieved a RMSE that is better than both Cinematch and GRF. We expect that EM SVD would overfit on the training set in the long run as it performs no regularization.

As the EM SVD approach requires the complete matrix to be filled up, the time complexity $O(I \times J \times n)$ is much higher than the other methods described in this paper which only need to train on observed data. In our rank 5 experiments, EM SVD took 2 days of computing, VB inference required only 2 hours to complete 40 iterations and GRF took less than 30 minutes to complete.

Since the performance and time complexity of EM SVD turned out to be inferior to our other techniques, we only tested EM SVD approach in our rank 5 matrix factorizations experiments, and do not compute EM SVD in subsequent experiments with larger rank matrix decompositions. However, we run EM SVD *once* to obtain an initialization for VB and MAP algorithms in later experiments - These single iteration runs typically took a few hours to complete.

5.3 Initialization

As VB inference uses an approximation of the posterior distribution, we have found that the initialization of U and V prior to running VB inference can affect both the convergence speed and RMSE performance at convergence. We tested the following ways of initializing U and V prior to running VB inference - (1) Random Init - Draw samples of U and V from $u_{il} \sim \mathcal{N}(0,1)$ and $v_{jl} \sim \mathcal{N}(0,1)$. (2) EM Init - Initializing U and V by using the matrix decomposition returned by one iteration of EM SVD. (3) GRF Init - Initialize U and V using the matrix decomposition returned by GRF.

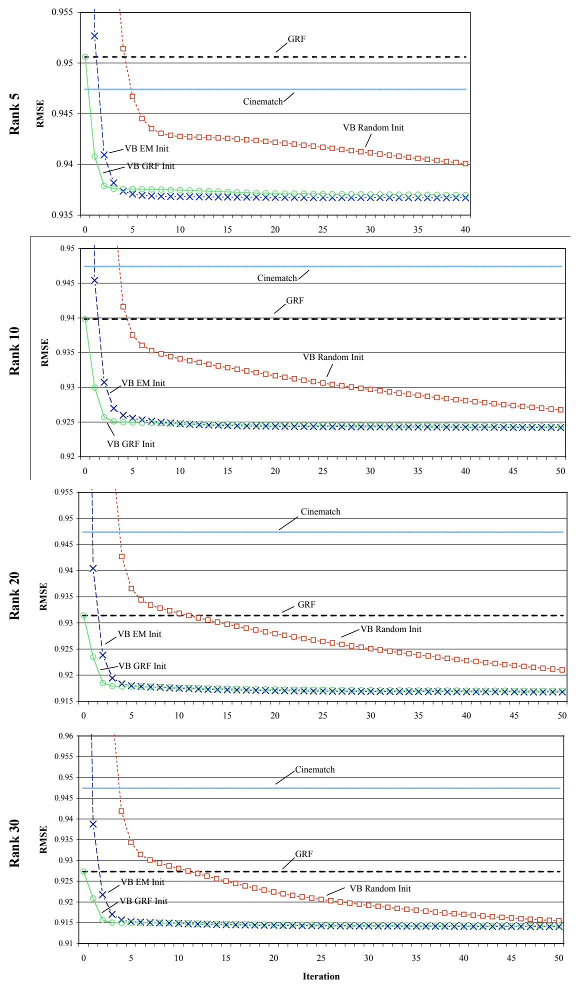

Figure 2: Comparisons of VB between initializing randomly, initializing to EM SVD and initializing to GRF. The x-axis shows the number of iterations the algorithm was ran for, and the y-axis shows the RMSE on validation data (lower is better).

Based on Figure 2, we can make the following observations: (1) All variants are able to achieve a RMSE better than Cinematch and GRF. (2) Randomly initializing U and V does not perform as well as initializing to EM SVD or GRF in terms of convergence speed or RMSE at conver-

[2] `http://www.timelydevelopment.com/Demos/NetflixPrize.htm`

gence. (3) VB inference appears to converge quickly (within 10 iterations) if initialized using EM SVD or GRF and does not exhibit any overfitting. (4) Initializing to GRF appears to be better in terms of convergence speed (for rank 10, 20 and 30 matrix decompositions). (5) Initializing using EM SVD and GRF have the same RMSE performance at convergence (although initializing to EM SVD seems to produce very slightly better RMSE at convergence). Based on these observations, and the fact that GRF is more computationally efficient than one iteration of EM SVD, we recommend initializing VB inference with GRF. However in later experiments we used EM SVD initialization instead; as observed above the resulting RMSE at convergence is virtually indistinguishable from GRF initialization.

5.4 MAP

To show that the effectiveness of VB inference is not just due to the priors introduced in our model, we compared it to a method of MAP estimation. In our experiments, we used the same program for VB inference to compute MAP estimates, except that the covariances Ψ_i and Φ_j in (17) and (20), respectively, are set to zero. The hyperparameters σ_l^2, ρ_l^2 and τ^2 are set to the values learnt by VB inference. We had also investigated learning these hyperparameters directly but found that it produced similar results as setting the hyperparameters to the VB learned values.

Similar to our experiments with VB inference, we explored various ways of initializing U and V prior to running MAP - (1) Random Init - Draw samples of U and V from $u_{il} \sim \mathcal{N}(0,1)$ and $v_{jl} \sim \mathcal{N}(0,1)$. (2) EM Init - Initializing U and V by using one iteration of EM SVD. (3) GRF Init - Initialize U and V using the matrix decomposition returned by GRF.

In Figure 3, we see that MAP has much more local optima issues compared to VB inference as different initializations converge to different local optima. For example, when initialized to EM SVD or GRF, VB inference consistently converges to local optima which are better than local optima reached when initialized randomly. On the other hand, MAP initialized to GRF performs better compared to randomly initializing for rank 5 and 10, whereas it performs worse than random initialization for rank 20 and 30. MAP also appears to converge much slower than VB and requires many more iterations before reaching local optima.

Another observation we can draw from Figure 3 is that MAP manifests overfitting problems while VB inference does not. This is because the RMSEs on the validation data do not decrease monotonically. Figure 4 shows RMSE on both training and validation data when using MAP initialized to EM SVD. We clearly see that while RMSE on training data is decreasing, RMSE on validation data does not monotonically decrease. In contrast, when using VB inference initialized to EM SVD, RMSE on both training and validation data decrease monotonically.

5.5 Performance Comparisons

We also compared the performance of the various algorithms at convergence for rank 5, 10, 20 and 30 matrix decompositions. For VB inference, we used the U and V from the last iteration of VB initialized using EM SVD. For MAP inference, we used the U and V which achieved the best RMSE *on the validation set* over all iterations and when initialized randomly, using EM SVD and using GRF. Note that this way of choosing U and V gives unfairly good results for MAP. However we see that even this best possible MAP result is worse than the result we obtained using VB inference at convergence.

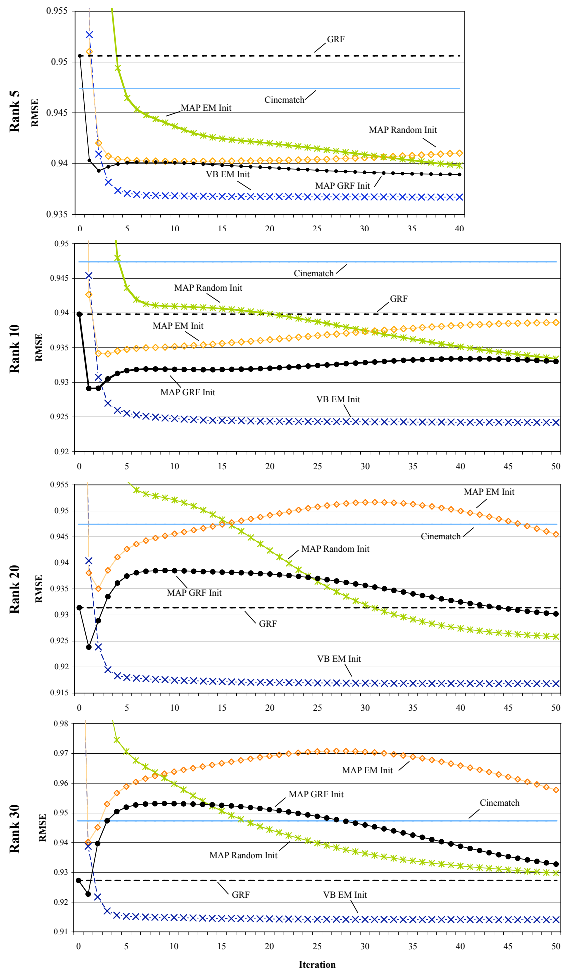

Figure 3: Comparisons of MAP between initializing randomly, initializing to EM SVD and initializing to GRF. The x-axis shows the number of iterations the algorithm was ran for, and the y-axis shows the RMSE on validation data (lower is better).

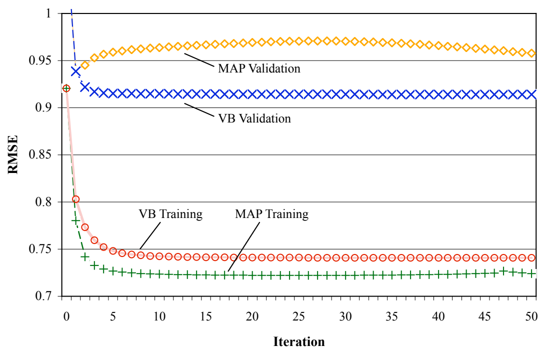

Figure 4: RMSE on training and validation data when using MAP and VB on rank 30 matrix decompositions initialized to EM SVD (lower is better).

Rank	Algorithm	RMSE	VB Improvement
Rank 5	Cinematch	0.9474	1.13%
	EM SVD	0.9583	2.26%
	GRF	0.9506	1.46%
	MAP	0.9389	0.24%
	VB	0.9367	-
Rank 10	Cinematch	0.9474	2.45%
	GRF	0.9398	1.66%
	MAP	0.9291	0.53%
	VB	0.9242	-
Rank 20	Cinematch	0.9474	3.23%
	GRF	0.9314	1.57%
	MAP	0.9238	0.76%
	VB	0.9168	-
Rank 30	Cinematch	0.9474	3.52%
	GRF	0.9273	1.43%
	MAP	0.9227	0.94%
	VB	0.9141	-

Table 1: Comparison of RMSEs achieved by all algorithms for matrix decompositions of different ranks.

The results are shown in Table 1. Firstly we see that VB inference consistently outperforms MAP, GRF, EM SVD and Cinematch, with the amount of improvement of VB over the other algorithms increasing as the rank increases. We believe this is because the other algorithms become more prone to overfitting as the rank increases, while as we had seen previously VB is not prone to overfitting. The best improvement VB gives over EM SVD is 2.26% (Rank 5), over GRF is 1.66% (Rank 10), and over MAP is 0.94% (Rank 30). We emphasize that the performance gain of VB inference over MAP is artificially low as we had purposely picked the best possible U and V in hindsight.

6. RELATED WORK

In addition to low-rank decompositions, there are other approaches to collaborative filtering. [5] gives a good overview of a variety of approaches. One popular alternative to solving collaborative filtering problems is to cluster the users and/or movies into groups. Ratings can then be predicted based on how a user group would typically rate a movie group. Such alternatives represent users and/or movies using discrete clusterings. Instead of clustering, [6] represented users and movies using binary vectors instead, where each feature can represent different aspects of movies and whether a user likes that aspect of a movie. This is a distributed representation and can be a much richer representation than clustering. Low-rank decompositions can also be viewed as distributed representations of users and movies, where each feature is a continuous value instead of binary as in [6].

Another approach to low-rank decompositions is maximum margin matrix factorization (MMMF) [9]. MMMF has been shown to give state-of-the-art results for collaborative filtering problems. Though still slower than SVD and related approaches (e.g. our VB inference approach), [4] has described an efficient implementation which appears to be applicable to the Netflix Prize dataset. As future work we would like to implement MMMF, to both compare against our VB inference approach as well as to interpolate the two approaches, hopefully producing better results.

7. CONCLUSION

In this paper, we described and discussed the shortcomings of SVD and various matrix decomposition techniques. We propose a Variational Bayesian Inference technique to alleviate overfitting in SVD, where priors are introduced and all parameters are integrated out using variational inference. We implemented and tested these algorithms on the Netflix Prize dataset to show experimentally that our technique gives significantly improved results over other matrix decomposition techniques. For low-rank matrix decompositions of rank 5, 10, 20 and 30, our proposed Bayesian approach achieves 2.26% improvement over an EM SVD approach, 1.66% improvement over a GRF approach, and 0.94% improvement over a MAP approach. As we expect VB inference to be more robust against overfitting, it should perform significantly better than GRF or MAP for larger rank matrix decompositions.

For our entry in the actual Netflix Prize competition, we used VB inference to compute the rank 100 matrix decomposition of the original rating matrix and a zero-centered rating matrix (by subtracting $\overline{v_j} + \overline{w_i}$, where $\overline{v_j}$ is the average rating for movie j, and $\overline{w_i}$ is the average offset from movie averages that user i had given in the training data). Both methods achieved results approximately 4.5% better than Cinematch, and blending these two results achieves slightly better than 5.5% improvement over Cinematch on the qualifying data.

8. REFERENCES

[1] M. J. Beal. *Variational algorithms for approximate Bayesian Inference.* PhD thesis, University College London, May 2003.

[2] J. Bennett and S. Lanning. The Netflix Prize. In *Proceedings of KDD Cup and Workshop 2007*, San Jose, CA, USA, Aug 2007.

[3] M. Brand. Fast online SVD revisions for lightweight recommender systems. In *Proceedings of the Third SIAM International Conference on Data Mining*, 2003.

[4] D. DeCoste. Collaborative prediction using ensembles of maximum margin matrix factorizations. In *ICML*, pages 249–256. ACM, 2006.

[5] B. Marlin. Collaborative filtering: A machine learning perspective. Master's thesis, University of Toronto, Canada, 2004.

[6] E. Meeds, Z. Ghahramani, R. Neal, and S. Roweis. Modeling dyadic data with binary latent factors. In *Advances in Neural Information Processing Systems*, 2007.

[7] R. M. Neal. Assessing relevance determination methods using DELVE generalization. In *Neural Networks and Machine Learning*, pages 97–129. Springer-Verlag, 1998.

[8] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes in C.* Cambridge University Press, 1992.

[9] J. D. M. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *ICML*, pages 713–719. ACM, 2005.

[10] N. Srebro and T. Jaakkola. Weighted low-rank approximations. In T. Fawcett and N. Mishra, editors, *ICML*, pages 720–727. AAAI Press, 2003.