Bayesian Approaches to Collaborative Filtering on Netflix Data

Hyun Jik Kim
Department of Pure Mathematics and Mathematical Statistics
University of Cambridge
Mathematical Tripos Part III
hjk42@cam.ac.uk

Abstract

Collaborative filtering defines a branch of techniques for tackling the following supervised learning problem: making predictions(filtering) about the preferences of a user, based on information regarding the preference of many users (collaborating). Having obtained such predictions, it is then a simple task to build a recommender system for users; we simply recommend the items with highest preference by a given user. In this paper, we first introduce the three most successful Bayesian algorithms for collaborative filtering on the Netflix dataset. These are **Probabilistic Matrix Factorization** (PMF) [1], its variant **Bayesian PMF** (BayesPMF) [2], and **Variational Bayes** (VB) [3]. We describe their mechanisms in full, fleshing out the missing details in the original papers. We then give results of authentic numerical experiments for direct comparison of their empirical performances and running time. Moreover we show results of experiments using combinations of these algorithms as an attempt to achieve better performance. Finally, we suggest extensions of these models and algorithms for the Netflix data which are worthy of interest. The hidden but perhaps most important purpose of this paper is to describe, explain, and provide diverse interpretations of various key ideas in Bayesian inference, including model construction with latent variables, extension to hierarchical models, MAP estimation, batch learning, Monte Carlo inference, Variational inference for models with latent variables and its relation to the EM algorithm. By presenting these ideas and inference methods in the context of collaborative filtering, the paper aims to provide helpful insight for the reader's understanding of Bayesian modelling and inference.

1 Introduction

Prediction of user ratings is relevant in various contexts, such as online videos, books and other commercial products. In this paper, we will focus on movie ratings by users provided by the Netflix dataset, the largest publicly available movie rating data. This was publicised during the Netflix Prize in 2009, a competition organised by Netflix for the most accurate recommender system that predicts user ratings, given the user rating history [4]. Netflix provided just over 100,000,000 ratings from around 480,000 randomly chosen users and around 18,000 movies which they have rated, each with an integer between 1 and 5, collected between 1998 and 2005. The task is to predict a pre-specified set of 3 million unobserved ratings (quiz set) from these users over these movies. The metric for error is the root mean-squared error (RMSE) over the quiz set.

Let us introduce some notation and terminology to begin with. Let there be N users with M movies. Each user u_i rates movie m_j with rating $R_{ij} \in \{1, ..., K\}$. Hence the rating matrix R is incomplete. In fact, it is incredibly sparse as most users will only have rated a small fraction of the total set of movies. Collaborative filtering aims to fill R by only using the already filled entries (observed

ratings) of R. The underlying assumption for this matrix completion task is that if user A has a similar rating as user B on a movie, then A is more likely to share B's rating on a different movie than another user selected at random.

This approach is contrary to content-based filtering, which predicts user ratings based on attributes of movies such as genre, number of viewings, director, cast etc. There are several drawbacks of content-based filtering, such as limited content analysis and over-specialisation, but the main difficulty in applying content-based filtering to a dataset as large as the Netflix data is that it requires building user profiles; this highly memory intensive task is unsuitable for large data sets. Also it is not so clear as to how detailed the profiles should be made and how the weights should be given to different attributes of films. This is why the vast majority of approaches to the Netflix problem are based on collaborative filtering, which deals with a problem of reduced complexity, and can thus be seen as a step towards automation.

Collaborative filtering can further be classified into two approaches: memory-based and model-based. The former uses the rating data to calculate similarities between users or between movies, and makes predictions based on these similar users/movies, called neighbours. Then a certain function of the similarity and observed ratings is used to compute the prediction. A popular example would be k-NN, where the prediction is a weighted average of the k nearest neighbours' ratings on a movie is the prediction. The similarity measure used is normally not a metric as the triangle inequality does not hold (two users with preferences dissimilar to a third user may have similar preferences). This implies that we need a similarity measure between all pairs of users, the number of which is $O(N^2)$ hence giving rise to scalability issues for big data such as the Netflix dataset ($N > 480,000$). Moreover the sparsity of the data implies that there could only be a handful of truely similar users, severely limiting the set of movies for which a sensible prediction can be made. A more fundamental problem is that there is no explicit statistical model, so very little insight into the data is gained when creating a memory-based prediction system.

On the other hand, model-based collaborative filtering methods use the rating data to train a statistical model, which can then be used to make predictions. There have been various models in the literature even before the Netflix Prize, notably cluster based models [5] and probabilistic models such as [6]. These algorithms seem to show impressive results on standard datasets such as MovieLens and EachMovie [1]. However we must bear in mind that these datasets have had all infrequent users, with fewer ratings than a lower threshold removed; these are intuitively the most difficult cases since there is least information about the users. The Netflix data, on the other hand, is very imbalanced: there coexist infrequent users with less than 5 ratings and frequent users with more than 10,000 ratings. Moreover the previously mentioned data sets are orders of magnitudes smaller than the Netflix data in size, and it has become apparent that scalability is also one of the main issues with these existing models.

The breakthrough for scalable model-based approaches was the clever idea of applying matrix factorisation to the rating matrix. We factorise the $N \times M$ rating matrix R as a product of two matrices U and V^T, where U and V are $N \times D$ and $M \times D$ respectively, where D is a positive integer to be chosen. They are referred to as the user matrix and the movie matrix. Instead of learning R directly, we build a model for R in terms of U and V, and try to learn U and V so that the error on the observed ratings is minimised. Then we simply fill in the blanks of R by multiplying out U and V^T. The intuition behind this model is that there are only a few factors which affect the user's rating of a movie. So each row of V can be interpreted as the D features of a movie which affect the rating of users, and then each row of U would correspond to the user's weights given to each feature. Note here that the choice of a suitable dimension D is important. If D is large, we will always be able to choose U and V such that they give perfect predictions on observed ratings, the training data. This therefore leads to overfitting, as well as being computationally infeasible. So we would like D to be small, but not too small that it fails to capture all the different features responsible for a rating.

From a mathematical point of view, we note that if $R = UV^T$, then $rank(R) \leq D$ since each column of R is a linear combinationa of the columns of U. Hence finding U and V is equivalent to finding the best rank D approximation to R in the squared error sense. It is a standard result in linear algebra that this can easily be obtained from the *Singular Value Decomposition* (SVD) of R, commonly referred to as the *Matrix Approximation Lemma* [7].

[1] See http://www.grouplens.org/

Theorem (Singular Value Decomposition)**.** *Suppose R is a real $N \times M$ matrix. Then there exists a factorisation of the form $R = U\Sigma V^T$ where U and V are $N \times N$ and $M \times M$ orthogonal matrices and Σ is a $N \times M$ diagonal matrix with the diagonal entries in descending order.*

Lemma (Matrix Approximation Lemma)**.** *Let $U\Sigma V^T$ be the SVD of a real matrix R, with the diagonal entries of Σ being $\sigma_1.\sigma_2, ...$ Then for $D \leq min(M, N)$, $\Sigma_D = diag(\sigma_1, ..., \sigma_D)$, $U\Sigma_D V^T$ is the best rank D approximation to R in Frobenius norm.*

However, the problem with our rating matrix is that it is incomplete-some entries are missing. The above result holds for R complete. This seemingly small modification results in a non-convex optimisation problem, and an approach different to standard SVD computation needs to be employed. Nonetheless there do exist SVD extensions which deal successfully with this problem, such as the use of the EM algorithm where the unobserved entries are modelled as latent variables [8], gradient boosting with successive rank 1 updates to U and V, and many other variants which have shown to be successful on the Netflix data [3]. However, these procedures have also been shown to overfit easily due to the extreme sparsity of the rating matrix, and require careful tuning of parameters such as the number of dimensions D for proper regularisation.

This paper describes and evaluates Bayesian models which are less prone to problems of overfitting, as well as being superior in performance to the aforementioned SVD algorithms. In section 2 we introduce the Probabilistic Matrix Factorization (PMF) algorithm, which uses MAP estimation on the simplest Bayesian model with independent Gaussian distributions on each entry of R along with independent Gaussian priors on the rows of U and V. In section 3 we give an extension of PMF called Bayesian PMF, which uses Monte Carlo estimation on a hierarchical Bayesian model. In section 4, we describe the Variational Bayesian approach (VB) which replaces MAP estimates of U,V in PMF with their expectations, and uses variational inference to approximate these values. In section 5, we compare their empirical performances: their RMSE on the test set, the RMSE for ratings of users grouped by number of ratings, and the RMSE for different combinations of these algorithms. We also compare the running times for these algorithms, and comment on computational expenses throughout the paper. Wherever possible we evaluate the results and offer possible explanations to provide insight for each algorithm. Finally in section 6, we discuss extensions of the models/algorithms, providing scope for further research.

2 Probabilistic Matrix Factorization (PMF)

This section refers to [1] unless otherwise stated.

We use the notation in the Introduction, along with U_i for row i of U and V_j for row j of V. Since we want $U_i V_j^T$ to estimate R_{ij}, the most natural Bayesian model would begin with the following conditional distribution on each R_{ij}:

$$R_{ij}|U_i, V_j \sim N(U_i V_j^T, \sigma^2) \tag{1}$$

Also for simplicity in calculation, we assume that R_{ij}, conditioned on the values of U and V, are independent random variables. Hence:

$$p(R|U, V) = \prod_{(ij)} \mathcal{N}(R_{ij}|U_i V_j^T, \sigma^2) \tag{2}$$

where the product ranges over pairs of user/movie indices in the training set, and $\mathcal{N}(x|\mu, \sigma^2)$ is the probability density of the Gaussian distribution with mean μ and variance σ^2

[1] does not mention that although R_{ij} is in fact a discrete, integer-valued random variable, it is modelled by a continuous distribution. This model is, however, justified as there is no requirement that the prediction must also be integer-valued. It is in fact beneficial to model R_{ij} as a continuous random variable, since then non-integer part of the prediction would be helpful in distinguishing the top recommendations from the decent ones.

Moreover we set U and V to be random variables, on which we put Gaussian priors with zero mean and spherical covariance (a multiple of the identity matrix):

$$p(U) = \prod_{i=1}^{N} \mathcal{N}(U_i|0, \sigma_U^2 I) \qquad p(V) = \prod_{j=1}^{M} \mathcal{N}(V_j|0, \sigma_V^2 I) \tag{3}$$

These priors can be interpreted as a natural regularisation induced by our Bayesian model, which can help reduce overfitting. We will see below how the values of parameters can be interpreted as the degree of regularisation. Here one could also raise the issue whether it is sensible to use a spherical covariance as opposed to, say, a diagonal covariance or even a full covariance matrix. [1] confirms that empirically, this generalisation does not give much of an improvement in performance, hence the paper adheres to a spherical variance for the sake of simplicity.

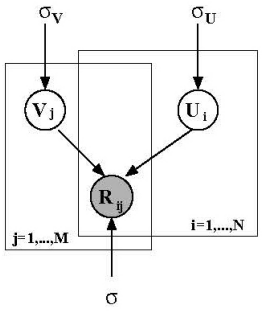


Figure 1: Graphical model for PMF. Source: [1]

By Bayes' Theorem, the log posterior of U and V can be calculated as:

$$\ln p(U, V|R) = -\frac{1}{2\sigma^2} \sum_{(ij)} (R_{ij} - U_i V_j^T)^2 - \frac{1}{2\sigma_U^2} \sum_{i=1}^{N} \|U_i\|_2^2 - \frac{1}{2\sigma_V^2} \sum_{j=1}^{M} \|V_j\|_2^2 + C \tag{4}$$

where C is some constant independent of U, V, R. Maximising the log-posterior over U and V is then equivalent to minimising the sum of squared errors with quadratic regularisation terms:

$$E = \frac{1}{2} \sum_{(ij)} (R_{ij} - U_i V_j^T)^2 + \frac{\lambda_U^2}{2} \sum_{i=1}^{N} \|U_i\|_2^2 + \frac{\lambda_V^2}{2} \sum_{j=1}^{M} \|V_j\|_2^2 \tag{5}$$

where $\lambda_U = \sigma^2/\sigma_U^2, \lambda_V = \sigma^2/\sigma_V^2$. Although unmentioned in [1], this form of quadratic regularisation, as in ridge regression, is intuitively the type of regularisation we would want in order to avoid overfitting; suppose we optimise U and V to minimise error on the observed entries of R. If these values of U, V have large entries, predictions for unobserved entries can become extreme. Due to the extreme sparsity of R this will become a major issue when trying to predict many ratings, as is the case for the Netflix Prize. So it is sensible to have U and V small by penalising their L^2 norms. This demonstrates that the use of a Gaussian prior on U and V, and using a *maximum a posteriori* (MAP) estimate is advantageous for three reasons. First, it automatically provides intuitive regularisation. Secondly, this approach to regularisation is much more flexible than simply penalising the L^2 norms of U and V, since we may use priors with diagonal or even a full covariance matrix with adjustable means. And thirdly, the parameters whose values we need to fix is reduced from three (σ) to two (λ), making easier the tuning parameter selection and hence effective regularisation.

Now the objective in (5) is minimised with respect to U and V by gradient descent, hence U, V converge to a stationary point:

$$U_i^{(n+1)} = U_i^{(n)} - \epsilon \frac{\partial E}{\partial U_i} + p(U_i^{(n)} - U_i^{(n-1)}) \tag{6}$$

$$V_j^{(n+1)} = V_j^{(n)} - \epsilon \frac{\partial E}{\partial V_j} + p(V_j^{(n)} - V_j^{(n-1)}) \tag{7}$$

where ϵ is the learning rate and p is the momentum term, which is used to avoid slow convergence due to heavy zig-zagging behaviour. Note that E is not convex in U nor V, so gradient descent does not guarantee convergence to the global minimum. This implies that the initial values of U, V affect the performance of the algorithm.

Instead of updating U and V after each epoch (one sweep through the entire training set), we divide the data into mini-batches of size 100,000 and update U, V after each batch in order to speed up training. Also since we are updating by going through the training set in order, it is important that we randomly permute the data after each epoch to avoid selection bias.

Through extensive experimentation, [1] claims that the use of values $\epsilon = 0.005, p = 0.9, \lambda_U = 0.01, \lambda_V = 0.001$ gives rise to best results. We use these values for the numerical experiments, the results of which are shown in section 5.

Algorithm 1: Probabilistic Matrix Factorization

Input: training_data,test_data,$T, D, U, V, \Delta U, \Delta V$

Initialise $\epsilon, \lambda_U, \lambda_V, p, numbatches$;
if $U, V, \Delta U, \Delta V$ *missing* **then**
    Initialise each entry of U, V to $N(0, 0.1^2)$;
    Initialise each entry of $\Delta U, \Delta V$ to 0 ;
end
for *t=1:T* **do**
    permute training_data ;
    for *batch=1:numbatches* **do**
        $\frac{\partial E}{\partial U_i}, \frac{\partial E}{\partial V_j} \leftarrow 0 \; \forall i, j$;
        Update $\frac{\partial E}{\partial U_i}, \frac{\partial E}{\partial V_j}$ for user/movie pairs $\{i, j\}$ in batch, by adding on terms contributing to derivatives with respect to U_i and V_j respectively. ;
        for *i=1:N* **do**
            $\Delta U_i \leftarrow p\Delta U_i - \epsilon \frac{\partial E}{\partial U_i}$;
            $U_i \leftarrow U_i + \Delta U_i$;
        end
        for *j=1:M* **do**
            $\Delta V_j \leftarrow p\Delta V_j - \epsilon \frac{\partial E}{\partial V_j}$;
            $V_j \leftarrow V_j + \Delta V_j$;
        end
    end
end

The time complexity for each epoch is $O(L + MD + ND)$ since the computation of $\frac{\partial E}{\partial U_i}$ is $O(D)$ assuming that the number of movies watched by a user is negligible compared to M on average, and a similar assumption for each movie. The space complexity is $O(MD + ND)$ since we need to store $\frac{\partial E}{\partial U_i}, \frac{\partial E}{\partial V_j}$.

[1] suggests various modifications to the model. First it raises the point that with zero mean, diagonal covariance Gaussian priors on U_i and V_j, the expected value of $U_i V_j^T$ is zero *a priori*, whereas we would like a number between 1 and K. Hence the paper suggests first mapping the ratings linearly to $[0, 1]$ by $t(x) = \frac{x-1}{K-1}$ and modelling the mean of $R_{ij}|U_i, V_j$ as $sigmoid(U_i V_j^T)$, so that the expected value is 1/2 [2] *a priori*. Although the paper claims this modification renders an improved RMSE, we later show in section 5 through various experiments that not only is this computationally more expensive but it also shows poor performance, followed by a possible explanation of why this

[2] One can show that a priori $U_i V_j^T$ has a symmetric distribution about 0, and for f an odd function about the line $y = a$, X a symmetric random variable, $\mathbb{E}(f(X)) = a$

is. Furthermore, we discuss in section 6 an approach which tries to learn the function that should be applied to $U_i V_j^T$, using the non-parametric approach of Gaussian Processes.

Note that regularisation is ocurring in two different ways in our algorithm. Firstly, our choice of D is crucial in controlling overfitting, since as mentioned in the Introduction, given sufficiently many factors the model can have arbitrarily small error on the training data. However the trouble with the Netflix data is that it is very imbalanced; the number of observations are significantly different over the rows and columns. Hence any single D will be too high for some rows but too low for others, so there is a limit as to how effective this form of regularisation can be. So our second form of regularisation also plays an important role, which is through choosing the parameters λ_U and λ_V. The simplest way to find suitable parameters would be to train each model on a set of candidate parameters, and choose the pair of values which gives the lowest RMSE. However this is computationally expensive as numerous models have to be trained. So the paper suggests a procedure called Adaptive Priors PMF, whereby priors are placed on $\sigma, \sigma_U, \sigma_V$, creating a hierarchial Bayesian model and so regularisation is now controlled by hyperparameters. More generally, we could place priors on the mean and variance parameters Θ_U, Θ_V for U, V. Then an MAP estimate of $U, V, \sigma, \Theta_U, \Theta_V$ are obtained again by gradient descent on the following posterior:

$$\ln p(U, V, \Theta_U, \Theta_V | R) = \ln p(R|U, V) + \ln p(U|\Theta_U) + \ln p(V|\Theta_V) + \ln p(\Theta_U) + \ln p(\Theta_V) + C \quad (8)$$

where C is a constant that does not depend on the parameters nor the hyperparameters. We will further discuss this model in the next section on BayesPMF.

3 Bayesian Probabilistic Factorization using Markov Chain Monte Carlo (BayesPMF)

This section refers to [2] unless otherwise stated.

For the Bayesian PMF model, we build on the same conditional distribution for R as in the PMF model:

$$p(R|U, V) = \prod_{(ij)} \mathcal{N}(R_{ij}|U_i V_j^T, \alpha^{-1}) \quad (9)$$

One problem of the PMF model described in the previous section is the need for manual setting of the regularisation parameters λ_U and λ_V, which is essential in controlling overfitting. So as suggested in the Adaptive Priors PMF model, it would be sensible to create a further level of parametrisation, by placing prior distributions over the parameters of U and V, so that they are controlled automatically by hyperparameters. In the BayesPMF model, however, the priors on U and V are now non-centered Gaussians, with further priors on both the mean and the precision, the inverse of the covariance matrix:

$$p(U) = \prod_{i=1}^{N} \mathcal{N}(U_i|\mu_U, \Lambda_U^{-1}) \qquad p(V) = \prod_{j=1}^{M} \mathcal{N}(V_j|\mu_V, \Lambda_V^{-1}) \quad (10)$$

$$\begin{aligned} p(\Theta_U|\Theta_0) &= p(\mu_U|\Lambda_U)p(\Lambda_U) \\ &= \mathcal{N}(\mu_U|\mu_0, (\beta_0\Lambda_U)^{-1})\mathcal{W}(\Lambda_U|W_0, \nu_0) \end{aligned} \quad (11)$$

$$\begin{aligned} p(\Theta_V|\Theta_0) &= p(\mu_V|\Lambda_V)p(\Lambda_V) \\ &= \mathcal{N}(\mu_V|\mu_0, (\beta_0\Lambda_V)^{-1})\mathcal{W}(\Lambda_U|W_0, \nu_0) \end{aligned} \quad (12)$$

Note that we have placed Gaussian-Wishart(G-W) priors on $\Theta_U = \{\mu_U, \Lambda_U\}$ and $\Theta_V = \{\mu_V, \Lambda_V\}$. This is because the G-W distribution is the conjugate prior to the Gaussian likelihood with unknown

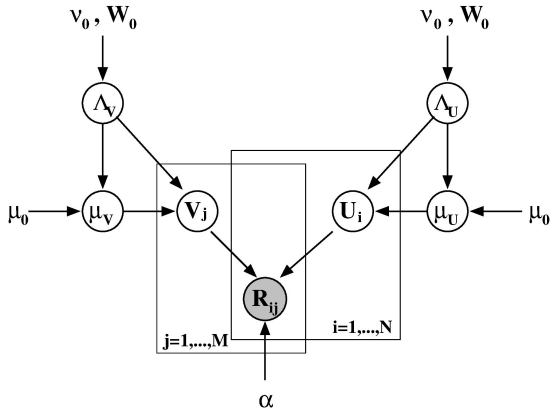


Figure 2: Graphical model for BayesPMF[2]

mean and unknown precision. In other words, this choice of prior ensures that the posterior of Θ_U and Θ_V are also G-W, making it easy to sample from these conditional distributions as we shall see later on. We define the set of hyperparameters as $\Theta_0 = \{\mu_0, \nu_0, W_0, \beta_0\}$, and use the values $\mu_0 = 0, \nu_0 = D, W_0 = I, \beta_0 = 2, \alpha = 2$ as in the paper.

Another potential problem of the PMF algorithm is that we use MAP estimates for U and V, a method which is prone to overfitting as it finds a single point estimate of U and V; it does not take into account all the information in the posterior distribution of U and V. Hence although the model is Baysesian, the method of inference is similar to maximum-likelihood and not truely Bayesian. Instead, it would be beneficial to put a distribution on the parameters Θ_U and Θ_V, and integrate out all random variables but R from the joint distribution to obtain its expected value.This is precisely what the paper attempts to do; the predictive distribution of a new rating R^*_{ij} is obtained by integrating out U, V and the model parameters from the joint:

$$p(R^*_{ij}|R, \Theta_0) = \iint p(R^*_{ij}|U_i, V_j)p(U, V|R, \Theta_U, \Theta_V)p(\Theta_U, \Theta_V|\Theta_0)dU\,dV\,d\Theta_U\,d\Theta_V \tag{13}$$

Exact evaluation of this distribution is analytically intractable, hence we must rely on approximate inference using a Monte Carlo approximation:

$$p(R^*_{ij}|R, \Theta_0) \approx \frac{1}{T}\sum_{t=1}^{T} p(R^*_{ij}|U_i^{(t)}, V_j^{(t)}) \tag{14}$$

where T is the number of epochs for our algorithm. Here each sample $\{U_i^{(t)}, V_j^{(t)}\}$ is generated by running a Markov chain with stationary distribution equal to the posterior of $\{U, V, \Theta_U, \Theta_V\}$. We use Gibbs sampling with a given number of jumps J for each sample $\{U_i^{(t)}, V_j^{(t)}\}$. MCMC methods are rarely used for large-scale learning as they are considered too computationally demanding. Nonetheless in this model, the use of G-W priors, which gives closed form posteriors and thus allows fast sampling from the posterior, gives rise to a computationally feasible implementation of Gibbs sampling.

The conditional distribution of U (and analogously for V) given all other random variables and hyperparameters is:

$$\begin{aligned} p(U_i|R, V, \Theta_U, \alpha) &= \mathcal{N}(U_i|\mu_i^*, [\Lambda_i^*]^{-1}) \\ &\sim \prod_{j\in N(i)} \mathcal{N}(R_{ij}|U_i V_j^T, \alpha^{-1})p(U_i|\mu_U, \Lambda_U) \end{aligned} \tag{15}$$

where

$$\Lambda_i^* = \Lambda_U + \alpha \sum_{j\in N(i)} V_j^T V_j \tag{16}$$

$$\mu_i^* = (\mu_U \Lambda_U + \alpha \sum_{j \in N(i)} V_j R_{ij})[\Lambda_i^{*T}]^{-1} \tag{17}$$

with a similar conditional distribution for V. Note that the conditional distribution over U factorises into a product of conditional distributions for each row U_i, and hence the sampler can easily be speeded up by sampling in parallel. This will indeed be a significant improvement since we must invert a $D \times D$ matrix for each sample of U_i, an $O(D^3)$ operation, and since there is a large number of users, this is the main bottleneck in reducing running-time. So we see how the independence assumptions for each entry of R and each row of U and V are beneficial for computational reasons.

The conditional distribution of the parameters Θ_U (and analogously for Θ_V) is given by the G-W distribution:

$$p(\mu_U, \Lambda_U | U, \Theta_0) = \mathcal{N}(\mu_U | \mu_0^*, (\beta_0^* \Lambda_U)^{-1}) \mathcal{W}(\Lambda_U | W_0^*, \nu_0^*) \tag{18}$$

where

$$\mu_0^* = \frac{\beta_0 \mu_0 + N\overline{U}}{\beta_0 + N} \qquad \beta_0^* = \beta_0 + N \qquad \nu_0^* = \nu_0 + N \tag{19}$$

$$[W_0^*]^{-1} = W_0^{-1} + N\overline{S} + \frac{\beta_0 N}{\beta_0 + N}(\mu_0 - \overline{U})^T(\mu_0 - \overline{U}) \tag{20}$$

$$\overline{U} = \frac{1}{N}\sum_{i=1}^{N} U_i \qquad \overline{S} = \frac{1}{N}\sum_{i=1}^{N}(U_i - \overline{U})^T(U_i - \overline{U}) \tag{21}$$

Algorithm 2: Bayesian Probabilistic Matrix Factorization using MCMC

Input: training_data,test_data,T, D, J, U, V (Use U,V obtained from PMF)

Initialise $\alpha, \mu_U, \mu_V, \Lambda_U, \Lambda_V, W_0, \beta_0, \nu_0, \mu_0$;
Initialise prediction = prediction of user/movie pairs in test_data for initial U V ;
for *t=1:T* **do**
  sample $\Theta_U^t = \{\mu_U^{(t)}, \Lambda_U^{(t)}\}$ from the G-W posterior distribution $p(\Theta_U | U, \Theta_0)$;
  sample $\Theta_V^t = \{\mu_V^{(t)}, \Lambda_V^{(t)}\}$ from the G-W posterior distribution $p(\Theta_V | V, \Theta_0)$;
  for *n=1:J* **do**
    for *i=1:N* **do**
      sample $U_i^{(t+1)}$ from the the posterior $p(U_i | R, V^{(t)}, \Theta_V^{(t)})$; `// parallelisable`
    end
    for *j=1:M* **do**
      sample $V_j^{(t+1)}$ from the the posterior $p(V_j | R, U^{(t+1)}, \Theta_V^{(t)})$; `// parallelisable`
    end
  end
  prediction ← prediction + prediction from the update of U, V ;
end
prediction ← prediction/$(T+1)$;

The time complexity for each epoch is $O((M+N)DJ)$ since sampling $U_i^{(t+1)}$ and $V_j^{(t+1)}$ are both $O(D)$ operations. The space complexity is $O(D(N+M))$ since we need to store $U_i^{(t)} \forall i$ and $V_j^{(t)} \forall j$.

Note also that we provide the output of PMF as the initial values for U and V in our algorithm. There is scope for experimentation by initialising U and V differently.

4 Variational Bayesian Approach to Movie Rating Prediction (VB)

This section refers to [3] unless otherwise stated. The model for this new approach is precisely the model in PMF with diagonal covariance priors on U and V. Note the change of notation:

$$p(R|U,V) = \prod_{(ij)} \mathcal{N}(R_{ij}|U_i V_j^T, \tau) \tag{22}$$

$$p(U) = \prod_{i=1}^{N} \mathcal{N}(U_i|0,\Sigma) \qquad p(V) = \prod_{j=1}^{M} \mathcal{N}(V_j|0,P) \tag{23}$$

where $\Sigma = diag(\sigma_1, ..., \sigma_D), P = diag(\rho_1, ..., \rho_D)$.

The novelty of this approach is that we use a different method for inference. We mentioned in the last section that a problem of the PMF algorithm is that it uses an MAP estimate $argmax_{U,V} p(U,V|R)$ for U and V. These are point estimates, which are prone to overfitting. So in the BayesPMF algorithm, we used MCMC to estimate the expected value of R by integrating out all other random variables from the joint. For the VB approach, we attempt to estimate instead the values $\hat{U} = \mathbb{E}(U|R)$ and $\hat{V} = \mathbb{E}(V|R)$, and use $\hat{U}_i \hat{V}_j^T$ as the prediction for R_{ij}. Now in order to evaluate these expectations, we need to be able to compute:

$$p(U,V|R) = \frac{p(R|U,V)p(U)p(V)}{p(R)} = \frac{p(R|U,V)p(U)p(V)}{\int_{U,V} p(R|U,V)p(U)p(V)} \tag{24}$$

However the denominator is both analytically and computationally intractable due to the high dimensionality of U and V, hence exact inference is not possible. Hence we resort to variational inference where we lower bound the marginal log-likelihood and try to maximise this bound, while at the same time using an approximation $Q(U,V)$ of $p(U,V|R)$ to estimate $\mathbb{E}(U|R)$ and $\mathbb{E}(V|R)$. The *variational free energy* [9] is:

$$\mathcal{F}(Q(U,V)) = \mathbb{E}_{Q(U,V)}[\log p(R,U,V) - \log Q(U,V)] \tag{25}$$

The variational free energy is a lower bound on the log-likelihood $p(R)$ for all distributions $Q(U,V)$:

$$\begin{aligned} \mathcal{F}(Q(U,V)) &= \mathbb{E}_{Q(U,V)}[\log p(R) + \log p(U,V|R) - \log Q(U,V)] \\ &= \log p(R) - KL(Q(U,V)\|p(U,V|R)) \\ &\leq \log p(R) \end{aligned} \tag{26}$$

where $Q(U,V)$ is an arbitrary distribution over U and V, and $KL(q||p) = \int q(x) log \frac{q(x)}{p(x)} dx$ is the *Kullback-Leibler(KL) divergence* from q to p, a measure of information lost when p is used to approximate q. Although this interpretation is not so intuitive and requires understanding of entropy in information theory, it relates to a more intuitive measure-theoretic notion of distance between two measures, called *total variational distance*, by *Pinsker's Inequality*:

Definition (Total Variational Distance)**.** *For P, Q two probability measures on measure sapce $(\chi, \mathcal{A})$ we define the total variational distance $\|P - Q\|_{TV} = \sup_{A \in \mathcal{A}} |P(A) - Q(A)|$*

Lemma (Pinsker's Inequality)**.** *Suppose P, Q have a common dominating measure μ in $(\chi, \mathcal{A})$. Then*

$$\|P - Q\|_{TV} \leq \sqrt{\tfrac{1}{2} KL(P||Q)}$$

Hence we see that maximising $\mathcal{F}(Q(U,V))$ with respect to $Q(U,V)$, and hence minimising the KL divergence from $Q(U,V)$ to $P(U,V|R)$, is sensible when trying to approximate $P(U,V|R)$ with $Q(U,V)$. Now $KL(P\|Q) \geq 0$ for all distributions P, Q, with equality if and only if $P \equiv Q$, known as *Gibb's Inequality* in information theory. Hence the minimiser of the KL divergence from $Q(U,V)$ to $P(U,V|R)$ is $P(U,V|R)$ itself. However we have from above that $P(U,V|R)$ is intractable, hence this global minimiser of the KL divergence is not what we desire. We want instead something similar to a "local minimum" but in the infinite dimensional

space of all distributions. A common practice in Bayesian variational inference when the posterior is intractable, is to constrain the approximator Q to be of a particular form, and minimise the KL divergence with respect to Q subject to this constraint, so that equality is not achieved above [9]. The paper applies the *mean field* approximation where $Q(U, V)$ is assumed to take the form $Q(U)Q(V)$. In other words, the posteriors of U and V are assumed to be independent when searching for the approximation. The reasoning behind using this constraint, apart from being simple and perhaps most natural, is that it makes the minimiser of KL divergence computationally tractable, so that we can find a closed form iterative scheme to approximate the minimising $Q(U, V)$.

$$\begin{aligned}\mathcal{F}(Q(U)Q(V)) = \mathbb{E}_{Q(U)Q(V)}\Bigg[&- \frac{1}{2}\left(\sum_{i=1}^{N}\sum_{l=1}^{D}\log(2\pi\sigma_l^2) + \frac{u_{il}^2}{\sigma_l^2}\right) - \frac{1}{2}\left(\sum_{j=1}^{M}\sum_{l=1}^{D}\log(2\pi\rho_l^2) + \frac{v_{jl}^2}{\rho_l^2}\right) \\ &- \frac{1}{2}\left(\sum_{(ij)}\log(2\pi\tau^2) + \frac{(R_{ij} - U_i V_j^T)^2}{\tau^2}\right) - \log Q(U) - \log Q(V)\Bigg] \\ = &-\frac{L}{2}\log(2\pi\tau^2) - \frac{N}{2}\sum_{l=1}^{D}\log(2\pi\sigma_l^2) - \frac{M}{2}\sum_{l=1}^{D}\log(2\pi\rho_l^2) \\ &- \frac{1}{2}\sum_{l=1}^{D}\left(\frac{\sum_{i=1}^{N}\mathbb{E}_{Q(U)}[U_{il}^2]}{\sigma_l^2} + \frac{\sum_{j=1}^{M}\mathbb{E}_{Q(V)}[V_{jl}^2]}{\rho_l^2}\right) \\ &- \frac{1}{2}\sum_{(ij)}\frac{\mathbb{E}_{Q(U)Q(V)}[(R_{ij} - U_i V_j^T)^2]}{\tau^2} - \mathbb{E}_{Q(U)}\log Q(U) - \mathbb{E}_{Q(V)}\log Q(V)\end{aligned} \tag{27}$$

where L is the total number of observations in the training data. To maximise the above we may simply maximise with respect to $Q(U)$ and $Q(V)$ separately, optimising one keeping the other fixed, and iterating until convergence. For this we use Lagrange Multipliers so that $\mathcal{F}(Q(U)Q(V))$ is maximised in $Q(U)$ subject to $\int Q(U)dU = 1$ and similarly for $Q(V)$. This gives:

$$Q(U) \propto \prod_{i=1}^{N}\exp\left(-\frac{1}{2}(U_i - \overline{U_i})\Phi_i^{-1}(U_i - \overline{U_i})^T\right) \tag{28}$$

where

$$\Phi_i = \left(diag\left(\frac{1}{\sigma_1^2}, ..., \frac{1}{\sigma_D^2}\right) + \sum_{j\in N(i)}\frac{\Psi_j + \overline{V_j}^T\overline{V_j}}{\tau^2}\right)^{-1} \qquad \overline{U_i} = \left(\sum_{j\in N(i)}\frac{R_{ij}\overline{V_j}}{\tau^2}\right)\Phi_i^T \tag{29}$$

where $\overline{V_j} = \mathbb{E}_{Q(V)}[V_j]$, $\Psi_j = \mathrm{Var}_{Q(V)}[V_j]$ and $N(i)$ is the set of j's such that R_{ij} is observed, with an analogous equation for $Q(V)$.

In the algorithm, we approximate $\overline{V_j}$ by the current value of V_j and initialise Ψ_j as $\frac{1}{D}I$ where I is the identity matrix, the reasoning for which we will see below.

However, this assumption that the posterior of U and V are independent is difficult to justify. Due to this crude approximation we may face performance losses on the test set, which is the main drawback of this variational algorithm.

The VB algorithm does not only maximise $\mathcal{F}(Q(U)Q(V))$ with respect to $Q(U), Q(V)$, but also tries to learn the variance parameters $\Theta = \{\tau, \sigma, \rho\}$. This is done by alternating the above maximisation of $\mathcal{F}(Q(U)Q(V))$ with respect to $Q(U)$ and $Q(V)$ with the maximisation (of the same objective) with respect to Θ. By setting each of the derivatives to 0, we get the maximisers:

$$\sigma_l^2 = \frac{1}{N-1}\sum_{i=1}^{N}(\Phi_i)_{ll} + \overline{U_{il}}^2 \tag{30}$$

$$\rho_l^2 = \frac{1}{M-1}\sum_{j=1}^{M}(\Psi_j)_{ll} + \overline{V_{jl}}^2 \tag{31}$$

$$\tau^2 = \frac{1}{L-1}\sum_{(ij)} R_{ij}^2 - 2R_{ij}\overline{U_i}\overline{V_j}^T + tr[(\Phi_i + \overline{U_i}^T\overline{U_i})(\Psi_j + \overline{V_j}^T\overline{V_j})] \tag{32}$$

Here we note that the VB algorithm is a *Generalised Expectation-Maximisation* algorithm applied to the hidden variables $Q(U), Q(V)$ and the set of parameters Θ [9]. For the E-step, we maximise $\mathcal{F}(Q(U)Q(V))$ with respect to $Q(U), Q(V)$ by minimising $KL(Q(U)Q(V)||P(U,V|R))$, and for the M-step we maximise $\mathcal{F}(Q(U)Q(V))$ with respect to Θ. For each step $\mathcal{F}(Q(U)Q(V))$ always increases, with log-likelihood staying the same for the E-step, whereas the KL divergence and log-likelihood may either increase or decrease for the M-step, since both are parametrised by Θ. Figure 3 displays the algorithm visually.

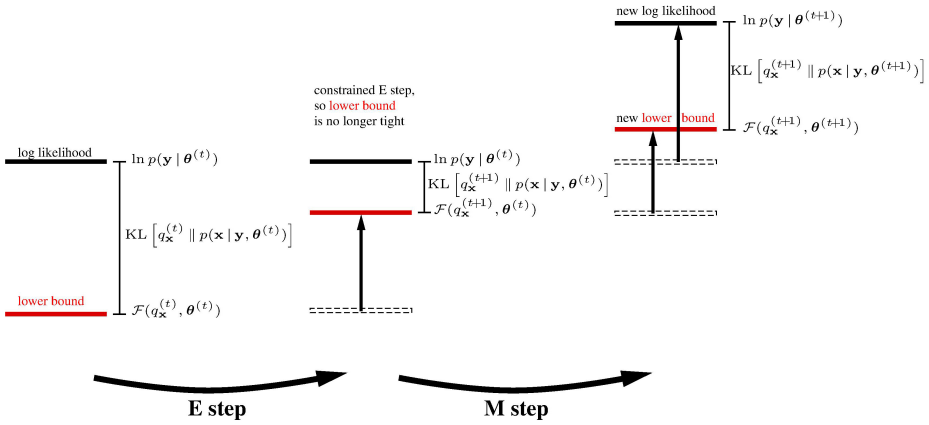


Figure 3: Variational interpretation of constrained EM algorithm. In the E-step the hidden variable posterior $q_{\boldsymbol{x}}^{(t+1)}(x)$ is set to the minimiser of $KL(q_{\boldsymbol{x}}(x)\|p(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{\theta}^{(t)}))$ subject to $q_{\boldsymbol{x}}$ lying in the family of constrained distributions. In the M-step the parameters $\boldsymbol{\theta}^{(t+1)}$ are set to maximise $\mathcal{F}(q_{\boldsymbol{x}}^{(t+1)}, \boldsymbol{\theta})$ Source: [9]

The maximisation of $\mathcal{F}(Q(U)Q(V))$ with respect to $Q(U), Q(V)$ and Θ can be seen as a type of maximum-likelihood approach with penalisation by how much $Q(U)Q(V)$ and $P(U,V|R)$ differ, their KL divergence. By maximising $\mathcal{F}(Q(U,V))$ with respect to both sets of arguments, we are trying to maximise the likelihood such that $KL(Q(U)Q(V)||P(U,V|R))$ does not get too large, which is reasonable since we wish to learn Θ so that it fits the data well but at the same time wish to learn $Q(U)Q(V)$ to be a good approximator for $P(U,V|R)$.

In summary, the variational algorithm runs as follows:

E-step:
1. Initialise S_j and t_j for $j = 1, ..., M$:

$$S_j \leftarrow diag\left(\frac{1}{\sigma_1^2}, ..., \frac{1}{\sigma_D^2}\right) \qquad t_j \leftarrow 0_{1\times D} \tag{33}$$

2. Update $Q(U_i)$ for $i = 1, ..., N$:
a) Compute Φ_i and $\overline{U_i}$:

$$\Phi_i \leftarrow \left(diag\left(\frac{1}{\sigma_1^2}, ..., \frac{1}{\sigma_D^2}\right) + \sum_{j\in N(i)} \frac{\Psi_j + \overline{V_j}^T\overline{V_j}}{\tau^2}\right)^{-1} \tag{34}$$

$$\overline{U_i} \leftarrow \left(\sum_{j\in N(i)} \frac{R_{ij}\overline{V_j}}{\tau^2}\right)\Phi_i^T \tag{35}$$

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

$$S_j \leftarrow S_j + \frac{\Phi_i + \overline{U_i}^T\overline{U_i}}{\tau^2} \qquad t_j \leftarrow t_j + \frac{R_{ij}\overline{V}}{\tau^2} \tag{36}$$

3. Update $Q(V_j)$ for $j = 1, ..., M$:

$$\Psi_j \leftarrow S_j^{-1} \qquad \overline{V_j} \leftarrow t_j\Psi_j^T \tag{37}$$

M-step:
Update σ_l, ρ_l, τ for $l = 1, ..., D$:

$$\sigma_l^2 = \frac{1}{N-1}\sum_{i=1}^{N}(\Phi_i)_{ll} + \overline{U_{il}}^2 \tag{38}$$

$$\rho_l^2 = \frac{1}{M-1}\sum_{j=1}^{M}(\Psi_j)_{ll} + \overline{V_{jl}}^2 \tag{39}$$

$$\tau^2 = \frac{1}{L-1}\sum_{(ij)} R_{ij}^2 - 2R_{ij}\overline{U_i}\overline{V_j}^T + tr[(\Phi_i + \overline{U_i}^T\overline{U_i})(\Psi_j + \overline{V_j}^T\overline{V_j})] \tag{40}$$

The variables that need to be stored for each E-M step are $\{\Phi_i, \overline{U_i}, \Psi_j, \overline{V_j} : \forall i \in \{1, ..., N\}, j \in \{1, ..., M\}\}$. Note however that the number of users $N > 400,000$ and each Φ_i is of size $D \times D$, so we will not have enough memory to store all the Φ_i's. To alleviate this, we instead use Φ_i wherever it is required as soon as it is computed, then we discard it immediately for each $i \in \{1, ..., N\}$. Note however that $\{\Psi_j : \forall j \in \{1, ..., M\}\}$ can be stored, since $M < 20,000$. As we shall see in the pseudocode below, all that is needed to initialise VB is U, V, Ψ, σ and τ. Also, instead of using S_j and t_j we update as we go Ψ and V', a container for the updated V. If initial values of U and V are unspecified, we initialise them such that each entry is sampled independently from the standard Gaussian.

Also note that we have D spare degrees of freedom when choosing U and V, since multiplying a column of U by c and dividing the corresponding column of V by c, the product UV^T is identical. Thus we may keep $\rho_l^2 = \frac{1}{D}$ fixed, and initialise U and V by normalising each column of V so that it has L^2 norm $\frac{1}{\sqrt{D}}$ and multiplying each column of U by an appropriate factor to keep UV^T the same. Then since $diag(\sigma_1, ..., \sigma_D)$ is meant to the covariance of each U_i we initialise $\boldsymbol{\sigma}$ to be the diagonal terms of the sample covariance of the U_i's. τ^2 is initialised to 0.456 through experimentation according to the paper.

Algorithm 3: Variational Bayes

Input: training_data,test_data,T, D

Initialise $U_i, V_j, \Psi_j, \tau, \boldsymbol{\sigma}$ $\forall i \in \{1, ..., N\}, j \in \{1, ..., M\}$ with suitable normalisations applied to columns of U and V ;

for *t=1:T* **do**

  for *j=1:M* **do**

    Initialise V_j' to a row vector of D zeros ; `// container for new` V_j

    Set $\Gamma_j = \Psi_j + V_j V_j^T$;

    $\Psi_j \leftarrow DI_D$;

  end

  Initialise $\boldsymbol{\sigma}_{new}$ to a column vector of D zeros ;

  Initialise τ_{new} to 0 ;

  for *i=1:N* **do**

    Set $\Phi = \left(diag(\boldsymbol{\sigma}) + \sum_{j \in N(i)} \frac{\Gamma_j}{\tau^2}\right)^{-1}$;

    Set $U' = \left(\sum_{j \in N(i)} \frac{R_{ij} V_j}{\tau^2}\right) \Phi^T$; `// container for new` U_i

    Set $\Omega = \Phi + U' U^T$;

    $\sigma_{new} = \boldsymbol{\sigma}_{new} + U'^T .^2 + diag(\Phi)$; `//` $.^2$ `squares vector elementwise`

    for $j \in N(i)$**do**

      $\Psi_j \leftarrow \Psi_j + \frac{\Omega}{\tau}$;

      $V_j' \leftarrow V_j' + \frac{R_{ij} U'}{\tau}$;

      $\tau_{new} \leftarrow \tau_{new} + tr(\Omega \Gamma_j) + R_{ij}^2 - 2R_{ij} V_j U'^T$;

    end

    $U_i \leftarrow U'$;

  end

  for *j=1:M* **do**

    $\Psi_j \leftarrow (\Psi_j)^{-1}$;

    $V_j \leftarrow V_j' \Psi_j$;

  end

  $\boldsymbol{\sigma} \leftarrow \frac{\boldsymbol{\sigma}_{new}}{N-1}$;

  $\tau \leftarrow \frac{\tau_{new}}{L-1}$;

end

Note that the time complexity for each epoch of the algorithm is $O(L + MD^3 + ND^3)$, where the D^3 comes from the inversion of a $D \times D$ matrix. Having to compute this for each user is the bottleneck for the algorithm. The space complexity is $O(MD^2)$ since we need to store each Psi_j.

5 Experimental Results

The Netflix data is authentic user/movie rating data collected between October 1998 and December 2005, consisting of 100,480,507 ratings from 480,189 random users on 17,700 movies. For the actual Netflix Prize competition, Netflix provided a *quiz set* of 2,817,131 user/movie pairs with ratings withheld, and the submitted algorithms were ranked by their RMSE on this set. Netflix also provided the *probe set*, a set of 1,408,395 user/movie/rating triplets which is a subset of the training. This represents the distribution of the *quiz set*, and was given by Netflix so that participants could test their algorithms on the *probe set* prior to submission. The ratings for the *quiz set* still remain unpublished, hence most algorithms in papers on collaborative filtering for the Netflix data have used the Netflix data without the *probe set* (consisting of 99,072,112 ratings) as the training data, and used the *probe set* as the test data for evaluation of performance. We follow this convention for this paper. Netflix's own rating-prediction system Cinematch gives an RMSE of 0.9474 on the test data. We use this as a baseline for our numerical experiments. We first provide the results for each algorithm with comments, and later we compare them and also give results for different combinations. All our experiments were carried out on a quadcore Intel(R) Core(TM) i5-3570 3.40GHz CPU machine with 30 epochs for $D = 30, 60$. We provide running times as well as various measures of performance.

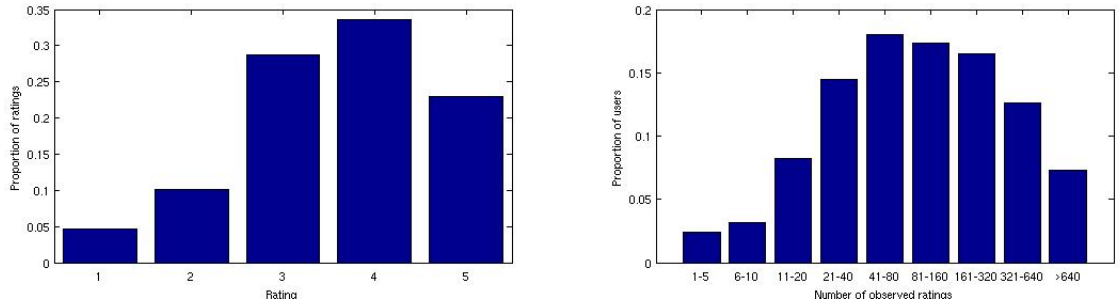


Figure 4: Basic information about the training data. Left panel: Distribution of ratings in the training data. Right panel: Distribution of users by number of ratings given.

All code was written in MATLAB with code for PMF and BayesPMF heavily based on [10] and code for VB self-produced.[3]

5.1 PMF

The code in [10] in fact learns U, V so that $U_i V_j^T$ predicts $R_{ij} - \overline{R}$ where $\overline{R}$ denotes the mean rating over the training data. It turns out that this slight modification leads to a noticeable increase in performance for PMF. We compare the results by referring to learning $R_{ij} - \overline{R}$ as PMF and learning R_{ij} as rawPMF. Moreover, we mentioned in section 2 that the paper uses a learning algorithm for the model $\frac{R_{ij}-1}{4} | U_i, V_j \sim \mathcal{N}(sigmoid(U_i V_j^T), \sigma^2)$, which claims to lead to better results. We refer to this as PMF2, and compare the results for these three variants in figure 4 below:

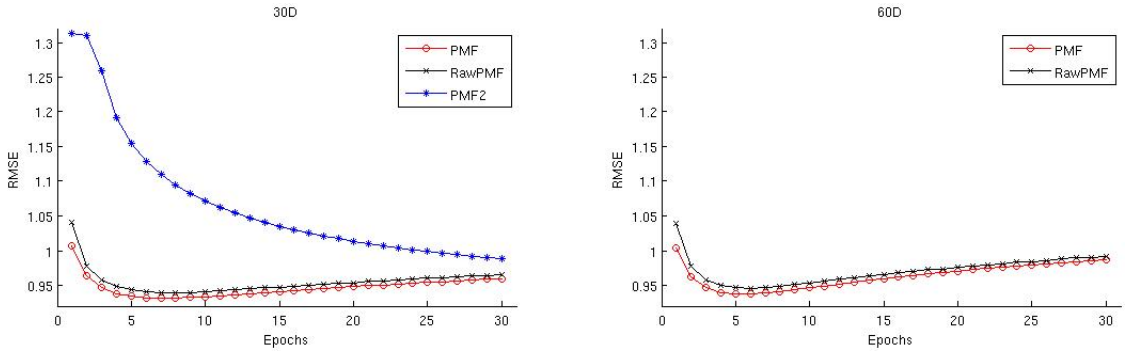


Figure 5: Left panel: RMSE of PMF, rawPMF and PMF2 in 30D as a function of epoch. Right panel: RMSE of PMF, rawPMF in 60D as a function of epoch.

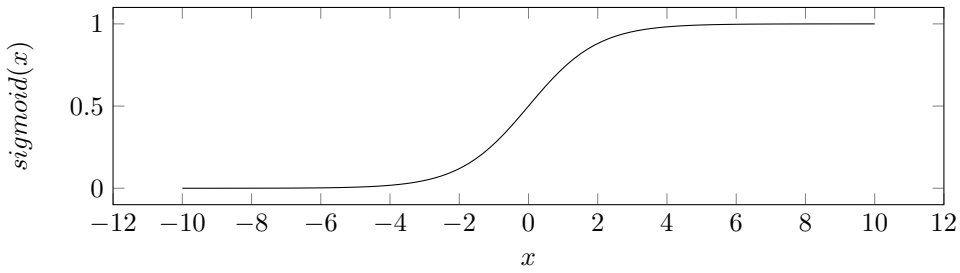
First note that the RMSE begins to rise early on for PMF in both graphs, suggesting that the updated values of U, V start to overfit fairly soon (epoch 7 in 30D and epoch 6 in 60D). This suggests that careful regularisation is very important in PMF. This claim is reinforced by how the 30D PMF model actually performs better than the 60D model, with minimal RMSE being 0.9314 and 0.9375 respectively; in higher dimensions, the presence of more variables implies a greater risk of overfitting. This suggests that we may want to adjust the tuning parameters λ_U, λ_V in a more sophisticated manner, for example by letting them vary with epoch. This provides motivation for BayesPMF and VB, where the parameters are indeed updated after each epoch.

Secondly, note that PMF does indeed perform consistently better than rawPMF, with their minimal RMSE being 0.9314 and 0.9392 respectively. A very rough explanation for this would lie in the way

[3] All code can be found at https://github.com/hyunjik11/PartIIIessay

U, V are initialised for these variants of PMF which, as pointed out in section 3, is important since the minimising objective is not convex so gradient descent converges to a local minimum. For both models, each entry of U, V are initialised by a centred Gaussian with variance 0.1^2. So initially, the model is equally likely to predict positive values as it is to predict negative values for R_{ij}. Since $R_{ij} - \overline{R}$ is positive for some pairs (i, j) and negative for others, we will expect the initial U_i, V_j to be better predictors for $R_{ij} - \overline{R}$ than for R_{ij}, hence the lower RMSE.

Finally, note the very poor performance of PMF2, contrary to what is claimed in the original paper. The RMSE after 30 epochs is 0.9880, much higher than the minimum 0.9314 of PMF. Moreover, since computing the gradients for gradient descent becomes much more expensive with the sigmoid function, PMF2 is also more computationally expensive (see Table 1). The sigmoid function $x \mapsto (1 + \exp(-x))^{-1}$ applied $U_i V_j^T$ in the PMF2 model maps real values to a value between 0 and 1, in order to bound the range of predictions(see graph below). However note that for the model $\frac{R_{ij}-1}{4}|U_i, V_j \sim \mathcal{N}(sigmoid(U_i V_j^T), \sigma^2)$, this is only a bound with high probability, and to ensure that this probability is high enough we need a small value of σ. The problem with having a small σ is that the extreme ratings close to 1 or 5 become very unlikely in our model, as this will require $|U_i V_j^T|$ to be very large; this is unlikely when we initialise each entry by $\mathcal{N}(0, 0.1^2)$.



5.2 BayesPMF

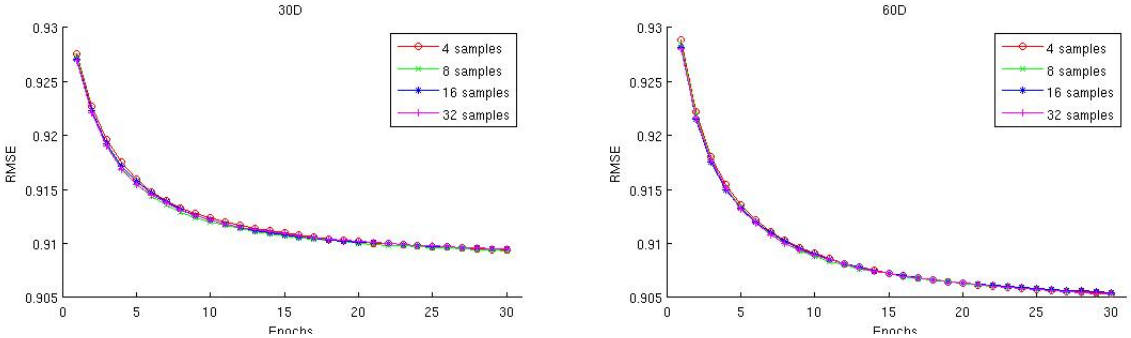


Figure 6: Left panel: RMSE of BayesPMF with different numbers of samples in 30D as a function of epoch. Right panel: Same in 60D

For our experiments on BayesPMF, we first compare the performances for different numbers of jumps J of the Markov Chain for each sample $\{U^{(t)}, V^{(t)}\}$. From the figure above, it is clear that the performances for $J = 4, 8, 16, 32$ are similar for both 30D and 60D. Note that running time grows linearly in J (see Table 1), but performance is slightly worse for $J = 4$ (0.9094 for 30D, 0.9053 for 60D) compared with $J = 8$ (0.9093 for 30D, 0.9052 for 60D). Although the difference is minute, the computing power required for $J = 8$ is still affordable, hence we use $J = 8$ for other variants of BayesPMF and comparisons below as a compromise between performance and computational speed.

Moreover unlike PMF, BayesPMF does not seem to overfit even after 30 epochs, and also performs significantly better in 60D than in 30D. One may infer that in this case, performance increases

steadily as model complexity increases, so higher values of D is likely to yield higher performance. We are however then limited by computing power, since running time scales linearly also with D (see Table 1). The original paper claims that with $D = 150$ and 300, the RMSE falls to 0.8931 and 0.8920 respectively. These results were not confirmed due to computational constraints.

The paper also claims that results could be improved if one includes a burn-in period for the MCMC inference, to allow the Markov Chain to converge to the stationary distribution. The original paper uses a burn-in period of 800 jumps, where the first 800 samples of $\{U, V\}$ are thrown away, and claims that this yields a noticeable improvement. The time taken for this step will be of the order of the time taken for 30 epochs of BayesPMF with $J = 32$, hence due to computational constraints we were unable to testify this claim.

5.3 VB

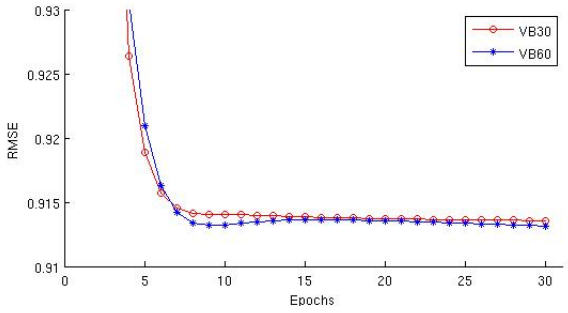


Figure 7: RMSE of VB initialised randomly in 30D and 60D.

For our experiments with VB, we first point out that random initialisation does much better than what is stated in the original paper. The original paper claims that VB in 30D with random initialisation gives an RMSE of around 0.9190 after 30 epochs, whereas our experimental results show that this is in fact 0.9136. Moreover, VB performs better for higher dimensions with an RMSE of 0.9132 for 60D.

Moreover we also tested VB for learning U, V such that $U_i V_j^T$ predicts $R_{ij} - \overline{R}$. However, we soon discovered that VB performs very poorly with this modification: for 30D, the RMSE goes down to 0.9878 at epoch 9 and stays near it. This contrasts with the results for PMF, where it favoured learning $R_{ij} - \overline{R}$. One possible explanation of this phenomenon is that the entries of U and V learned by VB are much larger in absolute value than for PMF, so reducing the absolute value of target ratings by subtracting the mean may not be desirable. This difference in the learned entries of U and V is expected, since for PMF we penalise the L^2 norms of U and V for learning (see equation (5)), so the algorithm favours smaller entries for U and V.

5.4 Combinations and Comparisons

Notice that the performances of PMF, BayesPMF and VB depend on the way the values U, V are initialised. Hence it is natural to use the learned U, V of one algorithm to initialise another, and examine whether we may obtain improved results. Note however that BayesPMF uses not the final values of U, V but the average of predictions across all epochs, so it is unsuitable for initialising VB. So the only feasible combinations we can test out is rawPMF initialised by VB, VB initialised by rawPMF, and BayesPMF initialised by VB. Note however that we must modify the BayesPMF algorithm so that we learn U, V such that $U_i V_j^T$ predicts R_{ij} instead of $R_{ij} - \overline{R}$. We refer to these combinations as rawPMF_VBinit, VB_rawPMFinit and BayesPMF_VBinit respectively. However the results we obtain in 30D for the first two combinations do significantly worse than their randomly initialised counterparts: for rawPMF_VBinit we get an RMSE of 0.9931 as the best RMSE over 30 epochs, and for VB_rawPMFinit we get 0.9225. These are both higher than 0.9392 and 0.9136, the respective results for random initialisation. Nonetheless BayesPMF_VBinit seems to show a comparable performance to BayesPMF, hence we include this in our comparison.

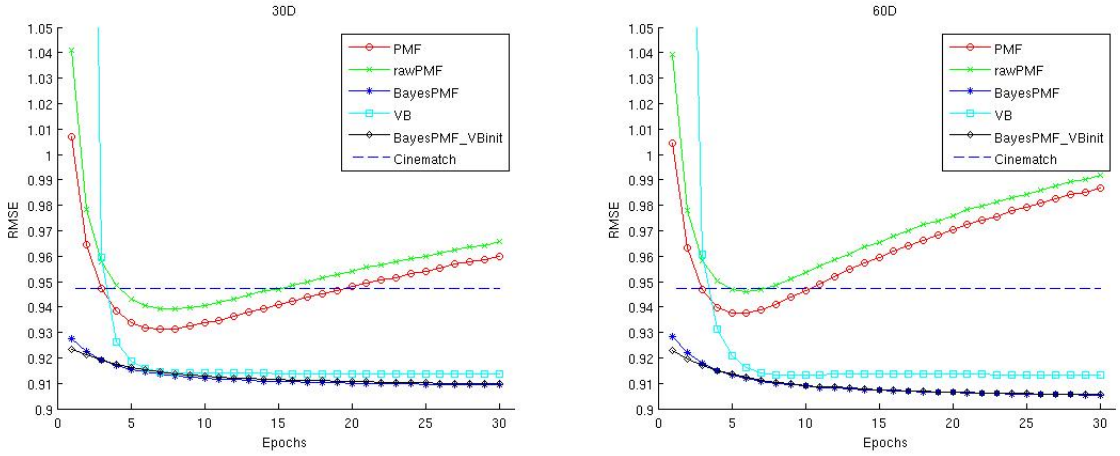


Figure 8: Left panel: RMSE of PMF, rawPMF, BayesPMF, VB and BayesPMF_VBinit in 30D (with predictions optimised over 30 epochs). Right panel: same with 60D.

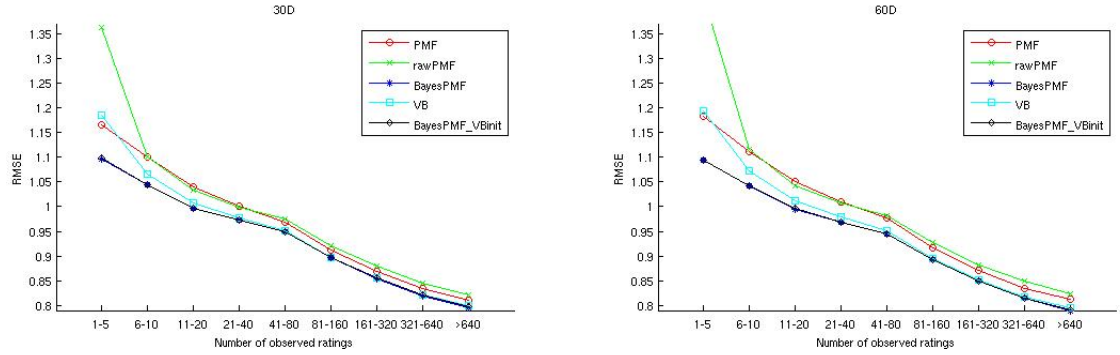


Figure 9: Left panel: RMSE of PMF, rawPMF, BayesPMF, VB and BayesPMF_VBinit in 30D (with predictions optimised over 30 epochs) for users grouped by number of views. Right panel: same with 60D.

It is clear from figure 8 that BayesPMF and its variant BayesPMF_VBinit are superior in performance compared to VB and PMF in both 30D and 60D. Although VB initially starts off with a very high RMSE, it quickly catches up and performs much better than PMF after 4 epochs, with a similar RMSE to BayesPMF by epoch 8. However with more epochs, the difference in performance between VB and BayesPMF becomes more and more noticeable, with a difference of 0.043(=0.9136-0.9093) in 30D and 0.790(=0.9132-0.9053) in 60D by epoch 30.

BayesPMF and BayesPMF_VBinit seem to show very similar performances in all results. They converge to very similar RMSE's in both 30D and 60D after the first two epochs, and also have a similar RMSE distribution among the different groups of users in figure 9. Note that in figure 9 BayesPMF and BayesPMF_VBinit show the best performance for all groups. VB however seems to have poor predictive power for the infrequent users who have not watched more than 5 movies, with an RMSE higher than PMF. On the other hand VB seems to have a similar predictive power to BayesPMF for users how have watched more than 40 movies. Nonetheless VB is computationally more demanding than BayesPMF, with running time more than threefold of BayesPMF with 8 samples.

We conclude that among the approaches discussed above, BayesPMF gives predictions with the lowest RMSE, and initialising it with VB gives results very similar to when it is initialised by PMF. However we note that VB can sometimes be equally effective as BayesPMF, especially with suffi-

ALGORITHM	RUNNING TIME(hrs)	RUNNING TIME (factor of PMF30D)
PMF30D	2.3	1.0
PMF2_30D	11.2	4.9
BayesPMF30D_4	6.1	2.7
BayesPMF30D_8	11.8	5.1
BayesPMF30D_16	23.7	10.3
BayesPMF30D_32	47.3	20.6
VB30D	33.1	14.4
BayesPMF_VBinit30D	12.3	5.3
PMF60D	4.4	1.9
BayesPMF60D_4	14.0	6.1
BayesPMF60D_8	27.5	12.0
BayesPMF60D_16	55.2	24.0
BayesPMF60D_32	112.5	49.0
VB60D	87.9	38.2
BayesPMF_VBinit60D	28.1	12.2

Table 1: Running times

ciently proficient users who have rated more than 40 movies, although VB is generally more computationally expensive than BayesPMF.

6 Extensions

We mention some feasible extensions of the models and inference algorithms covered in this paper, providing scope for further research.

6.1 Variational Bayesian approach to the Hierarchical Bayesian Model

Recall that for VB inference, we used the same model as PMF, where U, V have distributions parametrised by a set of fixed parameters Θ. In BayesPMF, we used a hierarchical Bayesian model, with an extra layer of parametrisation on Θ, placing a distribution on it governed by hyperparameters Θ_0. Hence it is natural to think of applying the variational framework to this hierarchical model. It turns out that the ideas in VB generalise nicely in this new setting, as shown in [9].

As before, we may lower-bound the log-likelihood of R by the variational free energy $\mathcal{F}(Q(U, V, \Theta))$ for an arbitrary distribution Q on U, V, Θ. We again have:

$$\mathcal{F}(Q(U, V, \Theta)) = \log p(R) - KL(Q(U, V, \Theta) \| p(U, V, \Theta | R)) \leq \log p(R) \tag{41}$$

So by maximising $\mathcal{F}(Q(U, V, \Theta))$, we are maximising the likelihood subject to the constraint that $KL(Q(U, V, \Theta) \| p(U, V, \Theta | R))$ does not get too large, ie. that $Q(U, V, \Theta)$ is a good approximation to $p(U, V, \Theta | R)$, the intractable posterior. We again maximise subject to a constraint on Q, as free maximisation simply yields $Q(U, V, \Theta) = p(U, V, \Theta | R)$ which is unhelpful. It is again natural to impose independence constraints $Q(U, V, \Theta) = Q(U)Q(V)Q(\Theta)$ for computational reasons. Then we may maximise $\mathcal{F}(Q(U, V, \Theta))$ by alternating over the E-step, where we optimise with respect to $Q(U)$ and $Q(V)$, and the M-step, where we optimise with respect to $Q(\Theta)$. We must assume however a further constrain on $Q(\Theta)$ to begin with, since we need to know one of $Q(U, V)$ and $Q(\Theta)$ to begin the EM algorithm. Hence constraining $Q(\Theta)$ to be a distribution parametrised by hyperparameters Θ_0, just as we put Gaussian-Wishart priors on both Θ_U and Θ_V in the model for BayesPMF, will be appropriate. It would be interesting to see whether there exist distributions which give rise to closed-form updates in the EM algorithm, and if not whether MCMC can be employed to approximate integrals. It will also be necessary to check whether this algorithm is computationally feasible for the Netflix data.

6.2 Matrix Factorization with Gaussian Processes

Recall from the PMF section that the original paper suggests the model $\frac{R_{ij}-1}{4}|U_i, V_j \sim \mathcal{N}(sigmoid(U_i V_j^T), \sigma^2)$. However through experimentation and explanations in section 5, we concluded that this model is infeasible. This suggestion naturally leads to the idea of using a model $R_{ij}|U_i, V_j \sim \mathcal{N}(f(U_i V_j^T), \sigma^2)$ where f is some function. Instead of fixing a function f, and then learning U, V, it would be ideal to impose a distribution over f and learn it at the same time as learning U, V. This suggests the idea of putting a Gaussian Process prior $f \sim \mathcal{GP}(\boldsymbol{m}, \boldsymbol{K})$, with the covariance kernel K constrained to have a certain form and parametrised by hyperparameters. Then we learn these hyperparameters as well as learning U, V by the approach given in [11]. The paper [12] uses instead the model $R_{ij}|U_i \sim \mathcal{N}(f_j(U_i), \sigma^2)$, omitting V. The paper successfully applies this model to MovieLens data of 10 million ratings, and it would be interesting to see if this can be extended to the Netflix data of 100 million ratings. It would also be worthwhile to compare the two possible models and explore which would be more suitable for the Netflix data.

6.3 Models with Variance Learning

It is intuitive that the variance of R_{ij} will be important for creating recommender systems, as people have limited time and cannot watch too many movies. So ideally we wish to recommend movies with not only a high expected rating but also a small variance, so that the user will be satisfied with high probability.

We could measure this variance by $\mathrm{Var}_{p(U,V|R)}[U_i V_j^T]$ or by $\mathrm{Var}(R|U, V)$. Note that for the above approaches, $p(U, V|R)$ is intractable, and so is $\mathrm{Var}_{p(U,V|R)}[U_i V_j^T]$. However for VB, we estimate the posterior by $Q(U)Q(V)$, for which we know the variances (Φ and Ψ), hence we can estimate $\mathrm{Var}_{p(U,V|R)}[U_i V_j^T]$ under the independence assumptions. So one could say that VB inference provides us with more information concerning posterior variances than with PMF or BayesPMF.

However it seems more intuitive to use $\mathrm{Var}(R|U, V)$ as the measure of variance we are interested in. Hence we may want to model the conditional distribution of R given U,V as:

$$R_{ij}|U, V \; N(U_i V_j^T, f(\sigma_i, \rho_j)) \tag{42}$$

where σ_i and ρ_j are some parameters inherent to each user/movie, which can be interpreted as a measure of "rationality" of a given user, or how much a movie stirs up extreme emotions in the viewer. With this interpretation, the current model for the distribution of $R|U, V$ assumes that all users give ratings "rationally" or at least with the same degree of rationality, since $\mathrm{Var}(R_{ij}|U, V)$ is the same for all pairs $\{i, j\}$. Hence were we to learn parameters based on data where some users have given ratings completely at random, not only will this model not be able to tell, but it will also provide skewed predictions for rational users which have been harmed by these random ratings. So we can say that by introducing a Bayesian model with different variance parameters for each user/movie, we may be able to build a more intelligent system which can automatically learn these parameters and distinguish users for which successful recommendation will be easier than others.

Acknowledgments

I would like to thank Dr. Rajen Shah for his helpful comments and answers to my questions. I would also like to thank Dr. Yew Jin Lim for providing the Python code of VB, which helped fill in the missing details of the algorithm as well as helping me to initialise parameters with suitable values in the MATLAB code.

References

[1] Salakhutdinov, R. & Mnih, A. (2007) Probabilistic Matrix Factorization. In *Advances in Neural Information Processing Systems 20*.

[2] Salakhutdinov, R. & Mnih, A. (2008) Bayesian Probabilistic Matrix Factorization using MCMC. In *Proceedings of the 25th International Conference on Machine Learning (ICML-2008)*.

[3] Lim, Y. J., & Teh, Y. W. (2007). Variational Bayesian approach to movie rating prediction. In *Proceedings of KDD Cup and Workshop*.

[4] Bennett, J., & Lanning, S. (2007]). The netflix prize. In *Proceedings of KDD cup and workshop*.

[5] Ungar, L. H., & Foster, D. P. (1998). Clustering methods for collaborative filtering. In *AAAI workshop on recommendation systems (Vol. 1)*.

[6] Hofmann, T. (2003). Collaborative filtering via Gaussian probabilistic latent semantic analysis. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval*.

[7] Eckart, C., & Young, G. (1936). The approximation of one matrix by another of lower rank. In *Psychometrika, 1.3*.

[8] Srebro, N., & Jaakkola, T. (2003). Weighted low-rank approximations. In *Proceedings of the 20th International Conference on Machine Learning (ICML-2003)*.

[9] Beal, M. J. (2003). Variational algorithms for approximate Bayesian inference*(Doctoral dissertation, University of London)*.

[10] Salakhutdinov, R. (2008) Bayesian Probabilistic Matrix Factorization [computer program]. Available at http://www.cs.toronto.edu/ rsalakhu/BPMF.html

[11] Lawrence, N. (2005). Probabilistic non-linear principal component analysis with Gaussian process latent variable models. In *The Journal of Machine Learning Research, 6*.

[12] Lawrence, N. D., & Urtasun, R. (2009). Non-linear matrix factorization with Gaussian processes. In *Proceedings of the 26th International Conference on Machine Learning (ICML-2009)*.