ISyE 7406 Project Report
Matrix Factorization Techniques for Movie Recommendation

Nevin Thomas[1] **, Praveen Muthukrishnan**[2] **and Vidyut Rao**[3]

[1]MS in Operations Research, GT ID: 903387577, nevinthomas@gatech.edu

[2]MS in Industrial Engineering, GT ID: 903512179, pmuthukr3@gatech.edu

[3]MS in Industrial Engineering, GT ID: 903511348, vrao67@gatech.edu

Abstract. *As many past studies have demonstrated, low dimensional factor models are superior to classic nearest neighbor techniques for generating product recommendations. The Probabilistic Matrix Factorization (PMF) approach towards collaborative filtering has been proven to be one of the simplest and most effective methods. Such models which are fitted on data by finding the maximum a posteriori estimates of the model parameters, scale linearly with the number of observations in addition to reasonable performance on large, sparse and imbalanced datasets. In this study we investigate the performance of multiple state of the art PMF variants on the 100k MovieLens dataset. We also investigate a biased version which can account for the individual user and movie specific biases, and a constrained version which addresses the comparable preferences of users who have rated similar sets of movies. We then move on to implement a deep version of the PMF, wherein instead of using a fixed dot-product as recommendation we will utilize some dense layers so that the network can find better combinations. Further, we explored a very effective Bayesian treatment of the PMF to bypass the difficulty of manually tuning the hyper-parameters on large datasets. All the models were implemented using the PyTorch framework in Python which helped us exploit the power of parallel computing using GPUs. We used a cosine user-user similarity model as our baseline and observed that all the PMF versions have significantly outperformed the baseline. The Deep version of PMF displayed the best performance (RMSE: 0.8297) followed closely by the Biased and Bayesian versions.*

1 Introduction

In the domain of machine learning methods for building recommender systems, there are primarily two schools: Content-based Filtering and Collaborative Filtering. Content-based filtering uses item features to recommend other items similar to what the user likes, based on their previous actions or explicit feedback. In order to do so, some features relevant to the items are required to be hand-engineered. This requires a lot of domain knowledge and the models are only as good as the hand-engineered features. Also, recommendations are based only on the existing interests of the user, limiting their ability to expand on the users' existing interests.

Collaborative Filtering instead uses similarities between users and items simultaneously to provide recommendations. This allows for serendipitous recommendations; that is, collaborative filtering models can recommend an item to user A based on the interests of a similar user B. Furthermore, the embeddings can be learned automatically, without relying on hand-engineering of features. For this reason, most state-of-the-art machine learning algorithms for recommender systems are based on Collaborative Filtering.

One of the most popular approaches to collaborative filtering is based on low-dimensional factor models. The idea behind such models is that the preferences of a user are determined by a small number of latent factors. In a linear factor model, a user's preferences are modeled by linearly combining item factor vectors using user-specific coefficients. For example, for N users and M movies, the $N \times M$ preference matrix R is given by the product of an $N \times D$ user coefficient matrix U^T and a $D \times M$ factor matrix V. Training such a model amounts to finding the best rank-D approximation to the observed $N \times M$ target matrix R under the given loss function.

Probabilistic Matrix Factorization techniques, introduced by Salakhutdinov *et. al.* [1], have been shown to be a particularly flexible and effective framework to address large, sparse and very imbalanced datasets. Constrained Probabilistic Matrix Factorization (cPMF), a variation on the simple PMF model is also introduced in the same paper, designed to deal with predictions for infrequent users. Multiple variants such as biased PMF, kernelized PMF and ckPMF, all of which possess the underlying PMF framework have been introduced in past studies aimed at recommending movies, music, books etc. following the first research article [2, 3, 4, 5].

Because the above methods are prone to over-fitting unless the regularization parameters are tuned carefully, Salakhutdinov *et. al.* [6] also propose a Bayesian treatment of Probabilistic Matrix Factorization. Here, model capacity is controlled automatically by integrating over all model parameters and hyperparameters. We reproduce their results by efficiently training the Bayesian PMF model using Markov Chain Monte Carlo methods.

In this project, we begin with a preliminary analysis of our movie recommendation data of choice: the MovieLens dataset. We then build two basic models, the Weighted Mean Model and the Cosine User-User Similarity Model to set a baseline for our results. We compare and contrast the different Probabilistic Matrix Factorization based models mentioned in literature by applying them to movie rec-

ommendation.

We primarily implement these models in PyTorch to harness its ability to speed up computation by running on GPUs as well as its ability to automatically compute gradients and backpropagate. PyTorch's deep learning capabilities also help us build on the basic PMF model by utilizing dense layers over fixed dot products. By exploiting the power of deep neural networks, we build a Deep PMF model that gives superior predictions.

2 Dataset

As this project is a study in the various collaborative filtering techniques used in recommender systems, the dataset required should be a matrix of user-item preferences. In order to highlight the utility of the probabilistic and bayesian treatment of matrix factorization and the other models discussed, we need a large, sparse and imbalanced dataset. For this purpose, the MovieLens dataset seems most appropriate.

Item	Original Dataset	Filtered Dataset
No. of Users	610	610
No. of Movies	9724	2121
No. of Ratings	100836	79636
Rating Density	1.69%	6.15%

Table 1: Statistics of the dataset used

GroupLens Research has collected and made available rating data sets from the web site, Movielens. The data sets were collected over various periods of time, depending on the size of the set. Various benchmarks already exist for the datasets present, making it very convenient for use in testing and validation. We build our models on their 100k dataset which contains 100,000 ratings from 600 users and 9000 movies.

To facilitate smooth splitting of the data into training and validation sets, sparse users and movies (with fewer than 10 occurrences) had to be filtered out. We are finally left with a dataset of around 80k ratings to work with. A simple pivot of the original dataset gives us the required matrix of user-movie preferences.

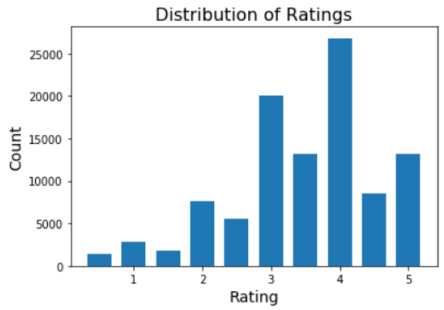

Figure 1: Distribution of Ratings

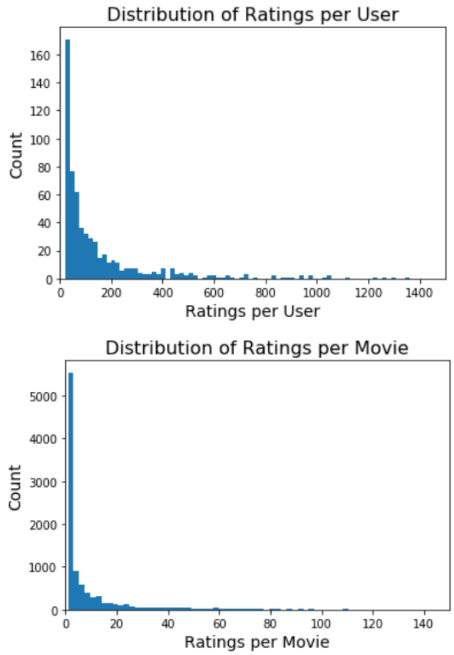

Figure 2: Distribution of ratings by entity (pre-filter).

3 Methodology

3.1 Notations

The main notations used in this project are defined as follows:
N : Number of users
M : Number of movies
$R \in \mathcal{R}^{N \times M}$: Preference Matrix
D : Number of Latent Vectors
$U \in \mathcal{R}^{N \times D}$: Latent user feature matrix
$V \in \mathcal{R}^{M \times D}$: Latent movie feature matrix
$W \in \mathcal{R}^{M \times D}$: Latent similarity constraint matrix
$I \in \mathcal{R}^{N \times M}$: Indicator matrix taking value 1 if R_{ij} is an observed entry, and 0 otherwise.

3.2 Baseline: Cosine User-User

Cosine similarity is used to measure the similarity between two items irrespective of their size. By measuring the cosine of the angle between two vectors projected in multi-dimensional space, it can identify the similarity between users as the similarity between the rating vectors [7]. Since our data contains a lot of empty values, we imputed it by filling the mean of each user into the empty values. Further, this can be extended to find similar items by calculating item-item similarity score. We are going to use this as a baseline accuracy throughout the study.

Consider two users A and B. The similarity s_{AB} between these users can be obtained as follows:

$$\mathbf{s}_{AB} = \frac{\mathbf{r}_A . \mathbf{r}_B}{||\mathbf{r}_A|| \times ||\mathbf{r}_B||} \tag{1}$$

where $\mathbf{r}_A$ and $\mathbf{r}_B$ represents the rating vectors of users A and B

3.3 Weighted Mean

In the mean rating model, movies are ranked based on the mean ratings. Consequently, the recommendation will be similar for all users irrespective of the user information. However, this approach is biased and favours movies with fewer ratings, as movies with large number of ratings seems to be less extreme in its mean ratings[7]. By giving weighted score to movies, it overcomes the biased nature of mean rating model as many good ratings outweigh few ratings. Changing the regularization parameter determines how much weight is put on movies with many ratings and thus RMSE might decrease compared to mean rating model. Essentially, there is a trade-off between interpretability and predictive power

3.4 Probabilistic Matrix Factorization

Input data in the context of movie recommender systems contain information about the ratings of (say) N users for M movies. We construct the preferences matrix R such that R_{ij} is the rating that user i gives the movie j. The objective of PMF is to factorize this matrix into matrices U and V where U is an $N \times D$ latent user features matrix while V is an $M \times D$ latent movie features matrix. We adopt a probabilistic linear model with Gaussian observation noise as seen in the panel below.

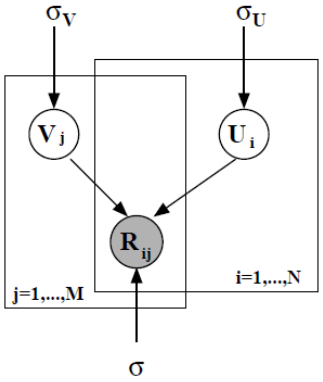

Figure 3: Graphical model for PMF

We introduce regularization hyperparameters $\lambda_U = \sigma^2/\sigma_U^2$ and $\lambda_V = \sigma^2/\sigma_V^2$, to penalize the user and movie coefficients and reduce the complexity of the model. With I_{ij} as the indicator if user i has rated movie j, we minimize the following sum of squared errors to obtain a local minimum by performing stochastic gradient descent on U and V.

$$\min_{U,V} \sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}(R_{ij} - U_i^T V_j)^2 + \lambda_U \sum_{i=1}^{N} ||U_i||_{Fro}^2 + \lambda_V \sum_{j=1}^{M} ||V_j||_{Fro}^2 \quad (2)$$

A simple example of calculating the ratings matrix R from the user and item feature matrices (U and V) is demonstrated in the Appendix Figure 10.

3.5 Biased PMF

Typical collaborative filtering data exhibits large systematic tendencies for some users to give higher ratings than others, and for some movies to receive higher ratings than others. This results in variations in rating values, independent of any interactions, termed as *biases* or *intercepts*. Bias terms are added to user and movie vectors which serves the role of normalization. Consequently, the observed rating is broken down into three components: movie-bias, user-bias and user-movie interaction [2].

As with Probabilistic Matrix Factorization (PMF), maximizing the log-posterior over movie and user features with hyper-parameters kept fixed is equivalent to minimizing the sum-of squared errors objective function with quadratic regularization terms:

$$\min_{U,V,b^u,b^v} \sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}(R_{ij} - b_i^u - b_j^v - U_i^T V_j)^2 + \lambda_U(||U_i||^2 + (b_i^u)^2) + \lambda_V(||V_j||^2 + (b_j^v)^2) \quad (3)$$

where $\lambda_U = \sigma^2/\sigma_U^2$, $\lambda_V = \sigma^2/\sigma_V^2$ and $||.||$ represents Frobenius norm.

3.6 Constrained PMF

Once a PMF model has been fitted, users with very few ratings will have feature vectors that are close to the prior mean, so the predicted ratings for those users will be close to the movie average ratings. This is a very crude way to make predictions for infrequent users. A better way is to find other users who watched the same(or similar) movies as the infrequent user i, and constrain the latent vectors of the similar users to be similar. Constrained PMF, another variation of PMF[1], utilizes this intuition. It constraints U with a latent similarity constraint matrix $W \in \mathcal{R}^{M*D}$ as follows:

$$U_i = Y_i + \frac{\sum_{k=1}^{M} I_{ik} W_k}{\sum_{k=1}^{M} I_{ik}} \quad (4)$$

where *I* is the observed indicator matrix with I_{ij} taking on the value 1 if user i rated movie j and 0 otherwise. Intuitively, the i^{th} column of the W matrix captures the effect of a user having rated a particular movie has on the prior mean of the user's feature vector.We define the conditional distribution over the observed ratings as

$$p(R \mid Y, V, W, \sigma^2) = \prod_{i=1}^{N}\prod_{j=1}^{M}[\mathcal{N}(R_{ij} \mid g([Y_i + \frac{\sum_{k=1}^{M} I_{ik} W_k}{\sum_{k=1}^{M} I_{ik}}]^T V_j), \sigma^2]^{I_{ij}} \quad (5)$$

We regularize the latent similarity constraint matrix W by placing a zero-mean spherical Guassian prior on it:

$$p(W|\sigma_W) = \prod_{k=1}^{M} \mathcal{N}(W_k|0, \sigma_W^2 I) \quad (6)$$

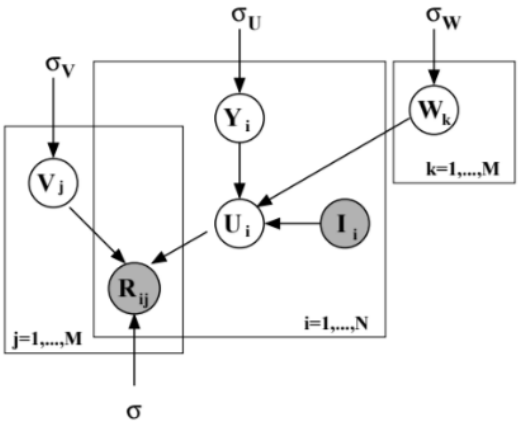

Figure 4: Graphical model for Constrained PMF

As with the PMF model, maximizing the log-posterior is equivalent to minimizing the sum-of-squared error functions with quadratic regularization terms:

$$\min_{U,V,W} \sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}(R_{ij} - g([Y_i + \frac{\sum_{k=1}^{M} I_{ik}W_k}{\sum_{k=1}^{M} I_{ik}}]^T V_j))^2 + \lambda_Y \sum_{i=1}^{N} ||Y_i||^2 + \lambda_V \sum_{j=1}^{M} ||V_j||^2 + \lambda_W \sum_{k=1}^{M} ||W_k||^2 \quad (7)$$

where $\lambda_U = \sigma^2/\sigma_U^2$, $\lambda_V = \sigma^2/\sigma_V^2$, and $\lambda_W = \sigma^2/\sigma_W^2$

3.7 Deep PMF

Deep PMF exploits the predictive power of deep neural networks to give accurate rating predictions. Instead of using a fixed dot-product as recommendation, we utilize dense layers so the network can find better combinations. We applied a $ReLU$ activation function on the output of the initial matrix multiplication output and then multiplied this with an $M \times M$ dense layer. We passed the results of this hidden layer through a sigmoid activation function and later scaled it back to the 0-5 scale to achieve predictions. The predictions are optimized by minimizing the following cost function

$$\min_{\mathbf{U},\mathbf{V},\mathbf{L}} ||\mathbf{I} * (\mathbf{R}^{'} - \hat{\mathbf{R}}_s)||^2 + \lambda_u ||\mathbf{U}||^2 + \lambda_v ||\mathbf{V}||^2 + \lambda_l ||\mathbf{L}||^2 \quad (8)$$

where,

$$\hat{R} = \sigma(ReLU(\mathbf{U}^T\mathbf{V})\mathbf{L}) \quad (9)$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$, $ReLU$ is the rectified linear unit activation function and $\mathbf{L}$ is an $M \times M$ matrix, $\hat{\mathbf{R}}_s$ is the predicted ratings matrix which is re-scaled back to the 0-5 scale. Note that the $*$ in Equation 8 is an element-wise matrix multiplication operator.

3.8 Bayesian PMF

With the Bayesian PMF model, we assume the prior distributions of the user and movie feature vectors to be Gaussian and further place Gaussian-Wishart priors on the user and movie hyperparameters $\theta_U = \{\mu_U, \Lambda_U\}$, $\theta_V = \{\mu_V, \Lambda_V\}$

$$\begin{aligned} p(\theta_U|\theta_o) &= p(\mu_U|\Lambda_U)p(\Lambda_U) \\ &= \mathcal{N}(\mu_U|\mu_0, (\beta_0\Lambda_U)^{-1})\mathcal{W}(\Lambda_U|W_0, \nu_0), \end{aligned} \quad (10)$$

$$\begin{aligned} p(\theta_V|\theta_o) &= p(\mu_V|\Lambda_V)p(\Lambda_V) \\ &= \mathcal{N}(\mu_V|\mu_0, (\beta_0\Lambda_V)^{-1})\mathcal{W}(\Lambda_V|W_0, \nu_0) \end{aligned} \quad (11)$$

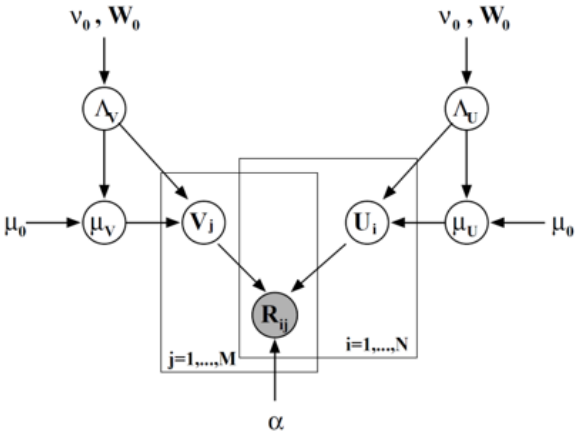

Figure 5: Graphical model for Bayesian PMF

We use Markov Chain Monte Carlo methods to approximate the predictive distribution of the rating value R_{ij}^* as shown below:

$$p(R_{ij}^* \mid R, \theta_0) \approx \frac{1}{K}\sum_{k=1}^{K} p(R_{ij}^* \mid U_i^{(k)}, V_j^{(k)}) \quad (12)$$

We use one of the simplest MCMC algorithms called the Gibbs sampling algorithm, which cycles through the latent variables, sampling each one from its distribution conditional on the current values of all other variables. For each iteration (or Monte Carlo step) 't', we:

1. Sample the hyperparameters

$$\begin{aligned} \theta_U^t &\sim p(\theta_U \mid U^t, \theta_0) \\ \theta_V^t &\sim p(\theta_V \mid V^t, \theta_0) \end{aligned} \quad (13)$$

2. For each i = 1,...,N, sample user features simultaneously:

$$U_i^{t+1} \sim p(U_i \mid R, V^t, \theta_U^t) \quad (14)$$

3. For each i = 1,...,M, sample movie features simultaneously:

$$V_i^{t+1} \sim p(V_i \mid R, U^{t+1}, \theta_V^t) \quad (15)$$

4 Analysis and Results

For all our experiments, we use the Root Mean Square Error (RMSE) as our evaluation metric for a model's performance:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(r_i - \hat{r_i})^2}{n}} \quad (16)$$

where r_i and $\hat{r_i}$ are the actual and predicted ratings respectively.

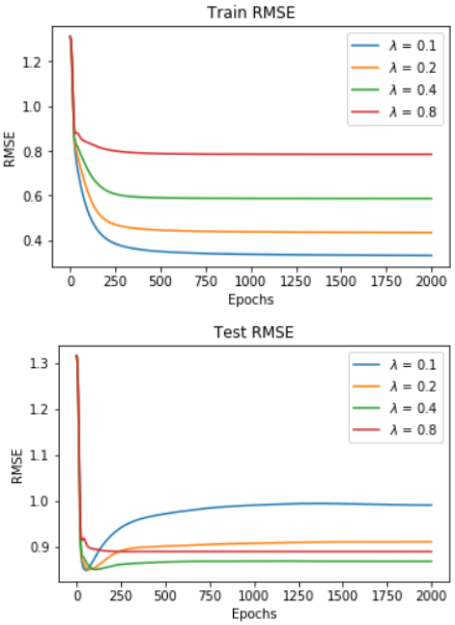

Figure 6: Train/Test RMSE vs epochs for different values of λ for PMF

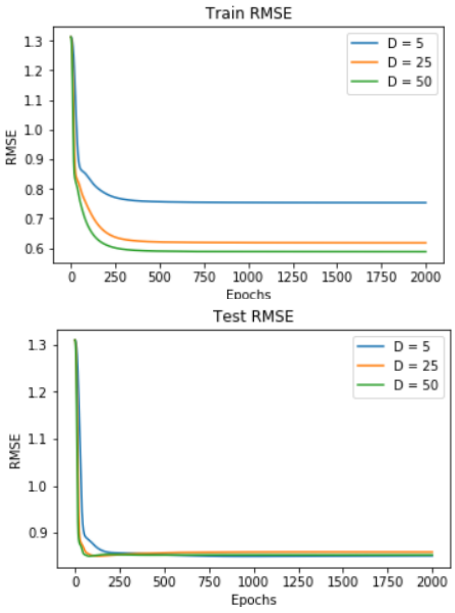

Figure 7: Train/Test RMSE vs epochs for different values of number of latent vectors for PMF

To obtain the best set of hyperparameters for each model, we study the evolution of the test RMSE with the number of training epochs. In the case of PMF we study the effect of both the number of latent features D as well as the regularization paramters λ_U and λ_V on the training and testing RMSE.

As a result of the increased regularization we see that higher values of λ result in a drop in train RMSE but an improvement in test RMSE; upto a point. After a λ value of around 0.4, the effect of increased bias outweighs that of reduced variance and increases the test RMSE as can be seen in Figure 6. This is a consequence of the bias-variance tradeoff.

The number of feature vectors D in the matrix factorization corresponds to the amount of information or noise retained, which directly influences the variance in the model. Increasing the number of features captures more information and hence always decreases the training RMSE. However, the increased variance can hinder performance on the validation set, reducing its RMSE, although it doesn't seem very apparent in the Figure 7. A final test RMSE of 0.8567 was obtained at $D = 10$ and λ_U, $\lambda_V = 0.45$. This is a 22.85% improvement over the baseline.

Biased PMF displays a general decrease in the training and test RMSE with an increase in the number of latent factors for the range of D values we checked. The RMSE vs epoch plots for different D values are depicted in Appendix Figure 11. However, we cannot generalize this trend because choosing a large D means increased variance in the model and might not result in improved test RMSE as mentioned earlier. We observe that adding the bias terms significantly improves the performance of the PMF model as is evident from Table 2.

We then implement the constrained version of PMF. It demonstrates an improvement in the test and train RMSEs as we increase the number of latent vectors involved in estimating the rating, similar to the biased variant. We observe that the assumption of similar preferences by users who have rated similar sets of movies made by constrained PMF does lead to an improved performance over PMF. However, it could not beat the performance of the biased PMF version as can be seen from the RMSE values reported in Table 2. Detailed convergence behavior for different D values can be found in Appendix Figure 12.

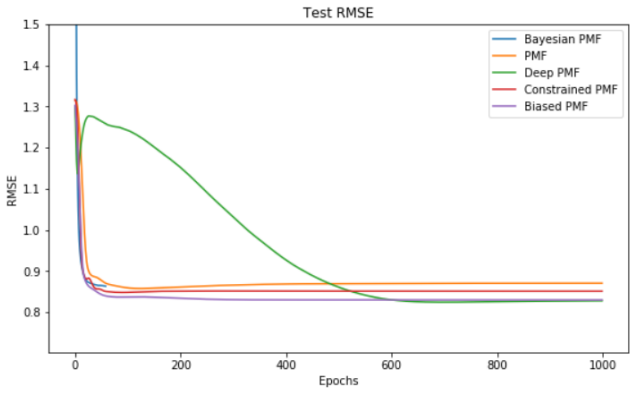

Figure 8: Contrasting the test RMSE of all models

Next, we implement deep PMF which gave us the best RMSE of 0.8297 on the test set, harnessing the power of deep layers. Compared to other methodologies, deep PMF takes considerably more amount of time as well as number of epochs for convergence because of the large number of parameters involved. As in the previous models, we did the analysis for three different values of D, details of which

can be found in Appendix Figure 13. Unlike the previous models the increase in run time with increase in D was way more perceivable in the deep version. We included only one hidden layer in the model we used. Adding multiple hidden layers might improve the performance of the model but at the cost of increased computational complexity.

Finally we implement the Bayesian PMF model which eliminates the need for manual hyper-parameter tuning. We run Bayesian PMF for different values of D, each for 40 MCMC steps. As seen with previous models, larger values of D lead to more variance .The best results were obtained at D = 10, with test RMSE of 0.833, a 24.99% improvement over the baseline. The details of the Bayesian PMF implementation can found in Appendix Figure 14.

Note that we could conduct an extensive cross validation only for the normal PMF because of the limited computational power at our disposal. Therefore, there is room for further improving the performance of the advanced models listed in Table 2.

Method	RMSE	% Improvement
Weighted Mean	0.9346	15.84
PMF	0.8567	22.85
Biased PMF	0.8305	25.21
Constrained PMF	0.8510	23.37
Deep PMF	0.8297	25.29
Bayesian PMF	0.8333	24.99

Table 2: Summary

5 Conclusion

Probabilistic Matrix Factorization far outperforms simple methodologies like cosine user-user similarity and weighted means for rating predictions. Among the different versions of PMF that we have implemented, Deep PMF exhibited the best performance in terms of test RMSE (0.8297). Biased PMF performed almost as well as deep PMF in spite of its simplicity, pointing towards the importance of accounting for user and movie specific biases in modelling the ratings.

A very important point to keep in mind while training the matrix factorization models is the choice of D. The number of latent vectors should be chosen such that it can hold sufficient information about the users and movies while ensuring the model variance is not too high. Further, we observed that the values of regularization parameters have a considerable impact on the test RMSEs and therefore tuning them is important. However, choosing the right hyperparameters (latent factors, regularization factors etc.) using cross validation might not be computationally feasible for large datasets, i.e., there is a scaling issue. In such scenarios, a Bayesian treatment of PMF would be more appropriate. Without the need for manually tuning the hyperparameters, the Bayesian PMF offers a scalable alternative with greater performance.

5.1 Key Takeaways

5.1.1 Project Takeaways

- A deep understanding of many of the state of the art collaborative filtering techniques.
- Hands on experience in developing machine learning and deep learning models in PyTorch.
- Insight on how researchers arrive at new machine learning algorithms.

5.1.2 Course Takeaways

- We appreciate the deep dive that Dr. Mei took on many machine learning algorithms, with a special emphasis on their mathematical underpinnings.
- The deliverables for the assignments and project have honed our report writing skills.

5.2 Code

The code for this project can be found at here.

References

[1] Andriy Mnih and Russ R Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

[2] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[3] Tinghui Zhou, Hanhuai Shan, Arindam Banerjee, and Guillermo Sapiro. Kernelized probabilistic matrix factorization: Exploiting graphs and side information. In *Proceedings of the 2012 SIAM international Conference on Data mining*, pages 403–414. SIAM, 2012.

[4] Lu Liu. Probabilistic matrix factorization for music recommendation.

[5] Cécile Logé and Alexander Yoffe. Building the optimal book recommender and measuring the role of book covers in predicting user ratings.

[6] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887, 2008.

[7] Data Liftoff. How to recommend anything / deep recommender. `https://www.kaggle.com/morrisb/how-to-recommend-anything-deep-recommender`.

APPENDIX

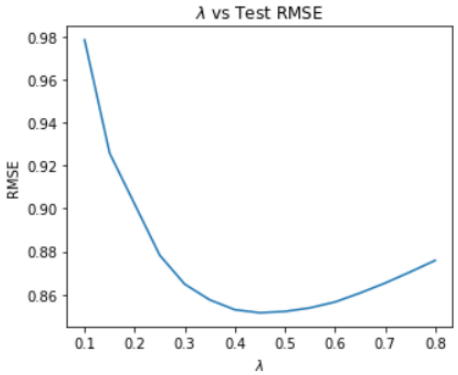

Figure 9: Tuning the regularization parameter for PMF

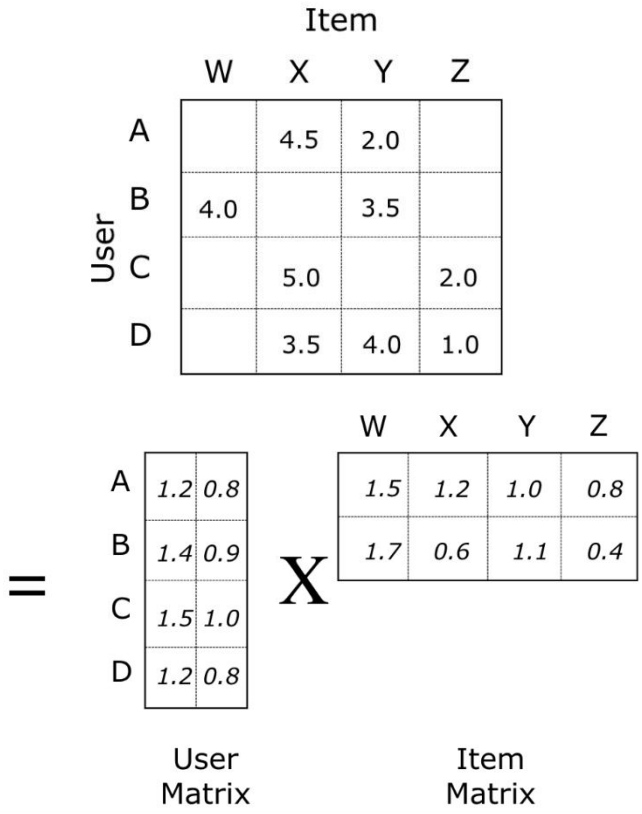

Figure 10: The Ratings matrix can be decomposed into a product of User and Item latent factor matrices.

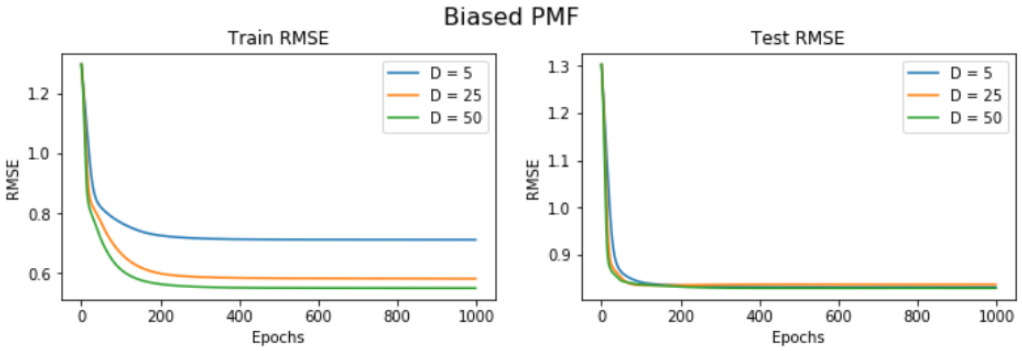

Figure 11: RMSE vs epochs for Biased PMF

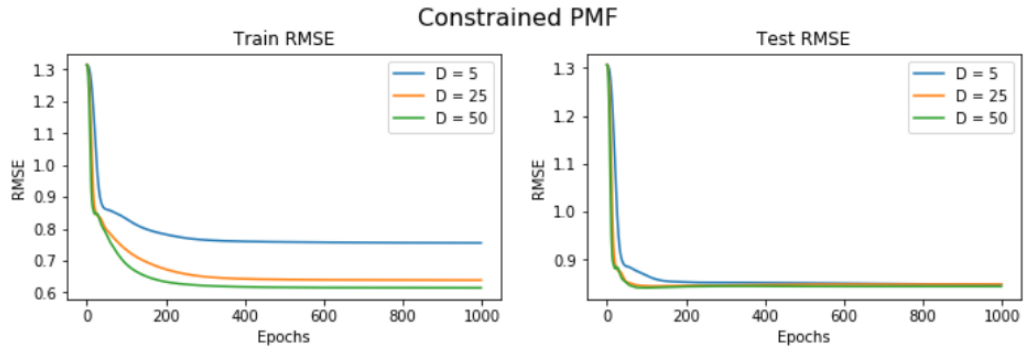

Figure 12: RMSE vs epochs for Constrained PMF

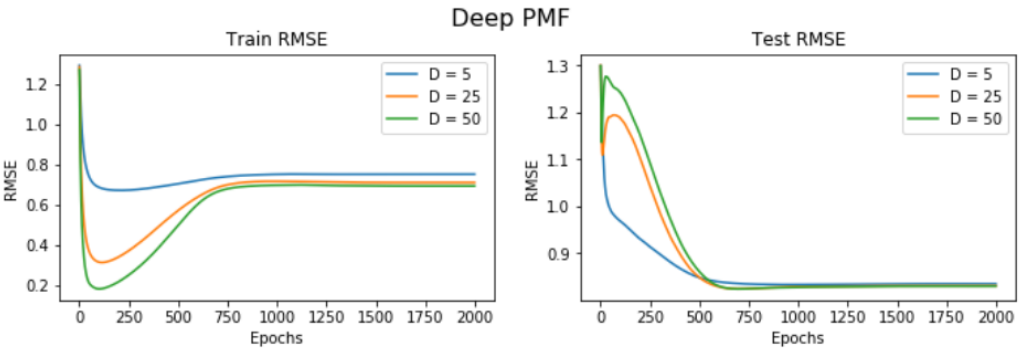

Figure 13: RMSE vs epochs for Deep PMF

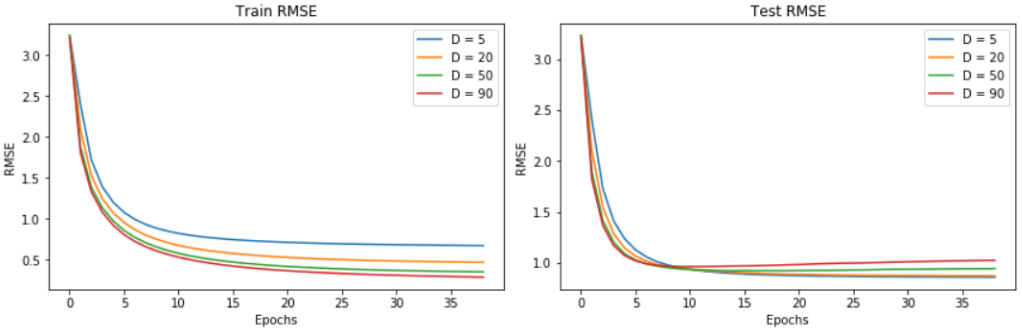

Figure 14: RMSE vs epochs for Bayesian PMF