Bayesian Recommender Systems

Abdulrahman Alabdulkareem[1] and Sarah Alnegheimish[1]

[1]CSAIL, MIT

May 2023

Abstract

With the proliferation of data nowadays, recommendation systems have become instrumental in filtering content for the user. These systems curate a set of personalized items to increase user satisfaction. In movie recommendation systems, the algorithm searches for content that would increase the user's watch time. In this paper, we design a Bayesian model to tackle the problem. Our goal is to predict the likelihood of a user liking an item. Experiments showed that our model is able to perform competitively with machine learning models. Moreover, in high confidence predictions, it surpasses them. However, the computational cost and lack of scalability of our model currently poses a limitation to its usage.

1 Introduction

Recommender systems are powerful tools in modern-day platforms, including online shopping, entertainment, and social media. Many companies such as Amazon, Netflix, and TikTok develop recommendation algorithms to increase their purchases and screen time. These systems are designed to predict users' preferences and recommend items or content that are most relevant to them, thereby enhancing user engagement and satisfaction. Many models have been developed to analyze large amounts of data in order to build recommendation systems, such as clustering, linear regression, and more recently deep learning.

In this project, we investigate a Bayesian methodology for predicting user preferences. We want to predict whether a user would like a new item based on the user's history. Using Bayesian modeling, we can improve on existing methods, such as quantifying uncertainty estimates in the predicted recommendation. Furthermore, Bayesian models can be easier to interpret; therefore they can be better at explaining predictions. For evaluation, we compare the model to simple baselines and existing machine learning models. We provide an extensive evaluation of the recommendation system models and identify the advantages and limitations of our Bayesian model. Moreover, we investigate areas in which Bayesian methods could outperform modern ML methods.

The remainder of this report is organized as follows: Section 2 goes through some of the most commonly known recommender systems algorithms. Our Bayesian model is illustrated in Section 3. Next, we evaluate the Bayesian model in Section 4 where we highlight the quality and computation performance of the Bayesian model. Lastly, we conclude in Section 5.

2 Related Work

Recommender systems algorithms can be categorized into: collaborative filtering approaches where the algorithm uses the behavior of similar users to predict a user's preferences; content-based filtering where the algorithm recommends items that are similar to the features of ones a user has liked in the past; matrix factorization where the algorithm reduces the dimensionality of the user-item matrix to make recommendations; clustering where the algorithm groups users or items into clusters and

recommends items to users within the cluster. There are also hybrid approaches where the algorithm combines multiple algorithms together.

One of the most cited papers in recommendation systems is by Koren et al [5]. The authors propose matrix factorization methods for collaborative filtering to create a low-dimensional representation. Their first model closely resembles singular value decomposition (SVD). However, SVD presented challenges when there is a high portion of missing values, i.e. sparse matrix, which is often the case in recommender systems. The authors also propose an alternative least squares (ALS) algorithm where it directly models the observed rating. This approach avoids overfitting compared to SVD. Moreover, it can be favorable with large data since it can be computed in parallel.

Recently, deep learning models have been introduced for recommendation [7]. For instance, multi-layer perceptron (MLP) can be developed to model the nonlinear interactions between users and items. Recurrent neural networks (RNN) take into account the temporal dynamics of content information. In addition, convolutional neural networks (CNN) extract global and local features from visual and textual information sources.

This project was originally based on the Condliff et al. work on mixed-effects Bayesian models [3]. The authors propose a Bayesian hierarchical modeling approach to find preferences of users. Their goal is to compute whether a user will like an item based on its features. More concretely, $P_i(L = 1 \mid \mathbf{f})$ where P_i is the probability that the i^{th} user will like ($L = 1$) the item represented by the feature vector $\mathbf{f}$. In Section 3.2, we will go through this model in detail.

3 Bayesian Model for Recommender Systems

In this section, we first describe the problem at hand. Second, we detail mixed-effects Bayesian model. Third, we present our own Bayesian model with latent variables. Lastly, we compare the aforementioned models.

3.1 Problem Description

We would like to investigate a Bayesian methodology for recommender systems. Given a set of users n and a set of items m (such as books, movies, songs, etc), we want to predict if a user $i \in [1, n]$ likes the item $j \in [1, m]$. Each user has a feature vector of length q, and each item feature vector is of length p.

Given a sparse rating matrix $R_{n \times m}$ of n users and m items.

$$R = \begin{bmatrix} r_{11} & ? & \cdots & ? \\ r_{21} & r_{22} & \cdots & r_{2m} \\ ? & r_{32} & \cdots & r_{3m} \\ \vdots & \vdots & \ddots & \vdots \\ ? & r_{n2} & \cdots & r_{nm} \end{bmatrix}$$

Where r_{ij} represents the rating that user i gave to movie j. Our objective is to hide some of the values in rating matrix (held-out set) and be able to predict the held-out values in the matrix (symbolized with ?) using the rest of the matrix.

3.2 Mixed-Effects Bayesian Model

The paper's model is a mixed-effects Bayesian model [3]. They predict the log-odds that a user i likes an item $L = 1$ based on the item feature vector $\mathbf{f}$. This is given by the equation:

$$\log \frac{P_i(L = 1|\mathbf{f})}{P_i(L = 0|\mathbf{f})} = \log \frac{P_i(L = 1)}{P_i(L = 0)} + \sum_{k=1}^{p} \log \frac{P_i(f_k = 1|L = 1)}{P_i(f_k = 1|L = 0)} \tag{1}$$

Which arises from assuming that the item features f_k are conditionally-independent and $P_i(L = 1|\mathbf{f})$ has a beta prior. Then, an important assumption is that $P_i(f_k = 1|L = 1)$ and $P_i(f_k = 1|L = 0)$ are

from a common distribution, which is an important part of Bayesian hierarchical modeling. Then the log-odds of a feature given the user's ratings is

$$\log \frac{P_i(f_k = 1|L = 0)}{P_i(f_k = 0|L = 0)} = \mu_{i_k} + \psi_{i_k} \tag{2}$$

and

$$\log \frac{P_i(f_k = 1|L = 1)}{P_i(f_k = 0|L = 1)} = \mu_{i_k} + \psi_{i_k} + \delta_{i_k} \tag{3}$$

Where μ_{i_k} and δ_{i_k} are Bayesian model parameters with defined priors, and $\psi_{i_k} = \beta_{0k} + \beta_{0k}x_{i1} + \cdots + \beta_{qk}x_{iq}$ where $\{x_{i1}, \cdots, x_{iq}\}$ are user features for user i. While the paper does not explicitly mention where β comes from, we assumed that it was computed from a linear regression model that was trained on predicting the fraction of liked items with feature f_k by user i given user features $x_{1\ldots q}$ as input.

The above set of equations define prior distributions for $P_i(f_k = 1|L = 1)$ and $P_i(f_k = 1|L = 0)$ for users $i = 1, \ldots, n$ and item features $k = 1, \ldots, p$. Together with the beta prior distribution for $P_i(L = 1)$ we get $P_i(L = 1|\mathbf{f})$ [3]. From there, we apply a sigmoid function on the log-odds to obtain a probability of a new item $P_i(L = 1|\mathbf{f})$ to be in the range [0, 1].

3.3 Latent Modeling Bayesian Model (Our Model)

The inhering limitations of the mixed-effects model (see Section 3.4) inspired the development of this Latent Modeling Bayesian Model. This Bayesian model learns a latent vector for each user and each item that can capture intrinsic correlations between them that are not present in the user or item features. We note that this model purposefully does not take input or make use of the item features nor the user features, instead all it takes as input is the user indices $i \in [1, n]$, item indices $j \in [1, m]$, and of course the sparse rating matrix.

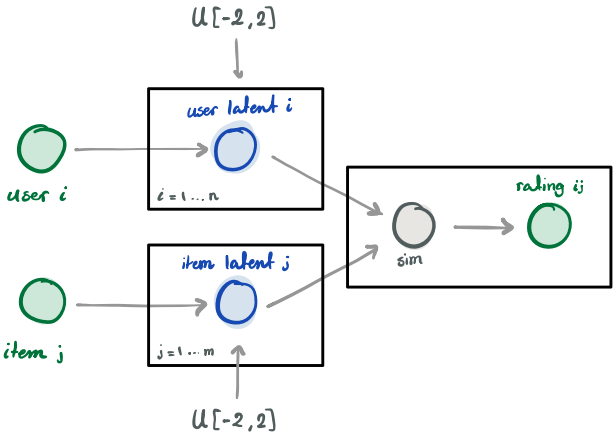


Figure 1: Model Graph

Given a dataset that contains n users $i \in [1, n]$ and m items $j \in [1, m]$, the model learns a d-dimensional vector $L_i \in \mathbf{R}^d$ for each user and a d-dimensional vector $L_j \in \mathbf{R}^d$ for each item where d is a hyperparameter. The prior on each element of the set of vectors L_i and set of vectors L_j is $\mathcal{U}[-2, 2]$[1]

Then, we define the log-odds of user i liking item j as similarity metric (simply a dot-product divided by d) of those two vectors defined by:

$$\text{logit}\left(P(L = 1|user = i, item = j)\right) = \text{sim}_{ij} = \frac{L_i^T L_j}{d} \tag{4}$$

Finally, we take the sigmoid of the log-odds to get our likelihood and prediction

$$P(L = 1|user = i, item = j) = \text{sigmoid}(\text{sim}_{ij}) = \text{sigmoid}\left(\frac{L_i^T L_j}{d}\right) \tag{5}$$

[1]The choice of [-2, 2] for the prior was chosen computationally as the models almost never chose values outside of this range, this also makes intuitive sense as the range of log-odds values we can get is -4 to 4 (if both latent vectors took the maximum value of all 2's) which give a likelihood range of $\approx -99\%$ to $\approx 99\%$ (when taking the sigmoid of the log-odds)

In total, the model has $(n * d + m * d)$ parameters which represent the n d-dimensional user latent vectors and the m d-dimensional item latent vectors, where d is a hyperparameter.

3.4 Comparing Mixed-Effects Model to Latent Model

Initial results of the mixed-effects model indicate a significant gap between the performance on our synthetic data compared to real data. On the synthetic data where the rating for user i on item j was determined (with little to no noise) based on the fully visible user and item features, the mixed-effects model was able to achieve impressive metrics including an F1 score close to 99% on the held-out test set. However, when we moved to any subset of the real datasets, the model was barely able to achieve metrics above random guessing even on the training set. This significant gap between real-world and synthetic data performance called for a deeper investigation.

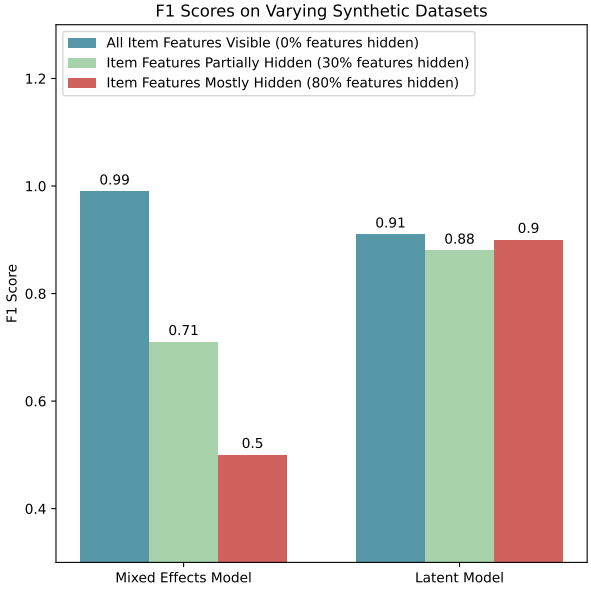


Figure 2: F1 scores of both models on our synthetic dataset where the rating is deterministic on the user and item features. Blue: No item features were hidden. Green: 30% of item features are hidden. Red: 80% of item features are hidden.

A major indicator of the issue was that the mixed-effects model was not able to capture any knowledge about items or users that are not present in the feature vector. In other words, imagine if we pick from our data three different items, e.g. items $j = \{5, 7, 12\}$ that have the exact same feature vector $\mathbf{f}$ and that items 5 and 7 consistently gets low ratings while item 12 gets high ratings, then the model would not be able to learn that correlation because the items have the same feature vector and thus cannot learn relationships outside of that. This coupled with the Naive-Bayes Assumption made on the feature vectors in Eq. (1) means the model is also not able to detect correlations within item features.

We investigated this hypothesis by creating a slightly different synthetic dataset where each item had its visible feature vector provided to the model and a hidden feature vector not provided to the model, then the rating depended on a combination of the two. The model achieves a much lower F1 score on this modified synthetic dataset[2].

This flaw of the mixed-effects model is exacerbated when learning from the real datasets we evaluate on which contain structure beyond the superficial and low-resolution features. This inherent lack of capability to model beyond the feature is what inspired our latent model in Section 3.3 which was capable of capturing hidden features beyond the visible features in the dataset. We report the F1 score of the mixed-effects model alongside the latent model in Figure 2. We note that the latent model, as mentioned in Section 3.3, does not accept as input the user or item features which is why the latent model doesn't change in performance when varying the amount of hidden features.

We note that the poor performance of the mixed effects model on real datasets is consistent with the results mentioned in the original paper [3], where the model achieved 56% accuracy on EachMovie data. Moving forward, we adopt our latent Bayesian model for our experiments.

4 Evaluation

In this section, we provide results of our evaluation. We first describe the datasets we are utilizing, then we report the performance of our models alongside other benchmarks.

[2]The F1 score drops lower as we increase the dependence on the hidden features. Once the dependence was higher than some threshold, our benchmarks show that even predicting the mean per-item would achieve a better result.

	Each Movie	Anime	MovieLens
# Items	1,623	8,520	1,681
# Users	61,265	68,048	943
# Ratings	2,811,983	4,059,564	100,000
# User features	3	0	0
# Item features	14 (genres)	22 (genres) 3 (continuous)	20 (genres)
Rating Scale	1-5	1-10	1-5
Sparsity %	97.2%	99.3%	93.7%

Table 1: Datasets Properties

Datasets. We utilize three real datasets for evaluation. We first use the dataset described in [3], which is from the Each Movie database [3]. The paper utilizes only a subset of the mentioned dataset. For a more extensive evaluation, we adopt MovieLens [4] datasets. MovieLens is a stable benchmark dataset for movie ratings, we use the 100k dataset. Lastly, we use the Anime dataset [5] which contains anime ratings for users. Table 1 summarizes the properties of these three datasets. We also construct a synthetic dataset with varying # of users and items.

Before directly using the data, we needed to process the data to generate features that would be beneficial to model prediction. Different data sources posses different information and therefore the features used in each dataset is different. Some of the calculated features include: the average rating of an item, the year the item was released, the average rating of a user, the top liked genre for a user, etc. Most datasets contain ratings as a continuous value, to make this problem a classification one, we threshold the ratings according to the median value. Moreover, for fair comparison, we subset the data based on the number of users and items since Bayesian models take longer with large datasets. We split the data into training, validation, and testing with a 80/10/10 division. In addition, all the data is curated such that both classes $L = 0$ and $L = 1$ are balanced in training, validation, and testing.

Models. For our evaluation, so far, we have adopted 6 models to compare against. The top 3 are simple baselines and the other 3 are machine learning models. **Item Mean (IM)** which is predicting the average rating per item. This model assumes that all users will have the same rating for each item. **User Mean (UM)** which is predicting the average rating per user. This model is will predict a constant value for each user across any item. **Item User Mean (IUM)** which is taking the average between the two baselines above. **Linear Regression (LR)** which is a simple linear regression model built to predict ratings from user and item features. **XGBoost Regressor (XGB)** which is a gradient boosting algorithm. Similar to linear regression, XGBoost predicts ratings from user and item features. **Multilayer Perceptron (MLP)** which is a feed-forward neural network with 5 dense layers. The first two layers are followed by a dropout layer to reduce overfitting. Moreover, the network features an embedding layer for both user and item inputs.

For implementation of the Bayesian model, we use the `PyMC` python framework. For learning our parameters, we use HMC[2] with the No U-Turn Sampler[4]. We run 5 parallel chains which achieves better results over one or two chians and gives us the ability to more easily verify whether we have converged or need to sample more from our posterior.

4.1 Benchmark

Figure 3 illustrates the results of our experiments. The plot depicts the accuracy and precision scores on test data for each model across the four datasets with $users = 100$ and $items = 100$. Immediately we notice that our Bayesian model is comparable to ML models. The best model overall is MLP and XGB and Bayesian are able to reach the same performance on both accuracy and precision. Moreover, we see that the performance on real datasets differs from the synthetic one. This suggests that the

[3] `http://www.research.digital.com/src/eachmovie/`
[4] `https://grouplens.org/datasets/movielens/`
[5] `https://www.kaggle.com/datasets/CooperUnion/anime-recommendations-database`

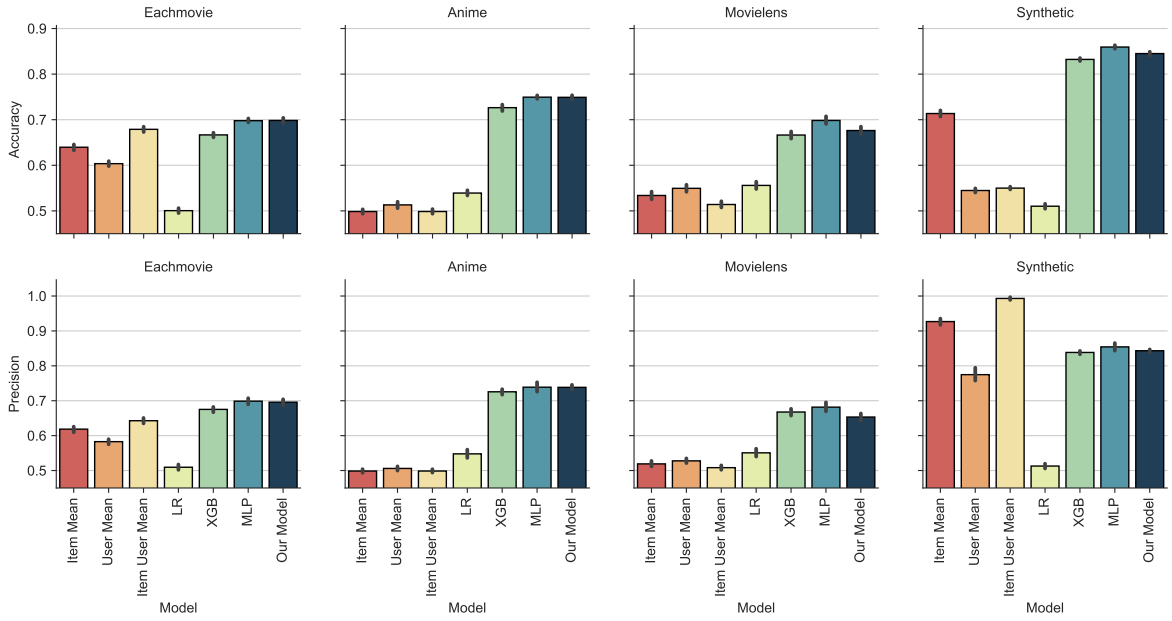
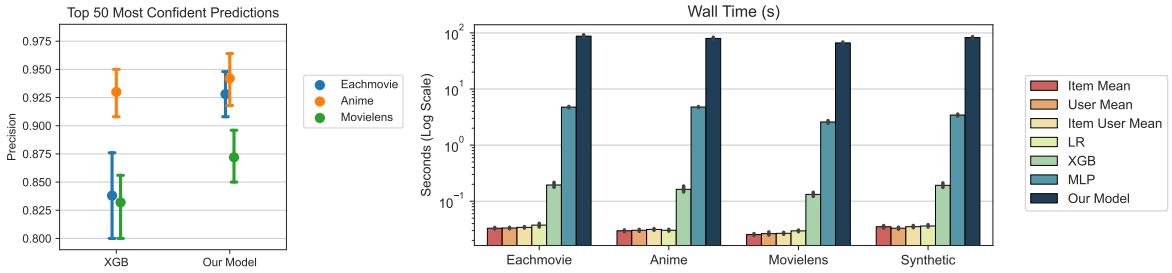


Figure 3: Benchmark results of models with accuracy and precision scores

synthetic data is not representative of real-world settings and explains why in some cases models struggle with real data. F1 and recall scores are presented in Table 2 in the Appendix.

4.1.1 XGB vs Our Model

Gradient boosting trees are considered one of the most commonly adopted models in industry given their state-of-the-art performance [1]. In our experiments, XGB (avg. accuracy = 0.72, avg. precision = 0.72) and our Bayesian model (avg. accuracy = 0.74, avg. precision = 0.73) produce competitive results. The differences between the accuracy and precision is miniscule, *so what does our model give?* One important feature of Bayesian modeling is confidence. Therefore, we inspect the performance of both models on their most confident predictions. Figure 4a showcases the precision scores for the top 50 predictions. We can see that when our Bayesian model is confident, it is highly reliable compared to XGB. This is especially true for the EachMovie and MovieLens datasets.



(a) (b)

Figure 4: (a) Precision of 50 most confident predictions in our model and XGB. (b) Wall Time (s) of all models. Our Bayesian model takes slightly less than 1.5 minutes to train.

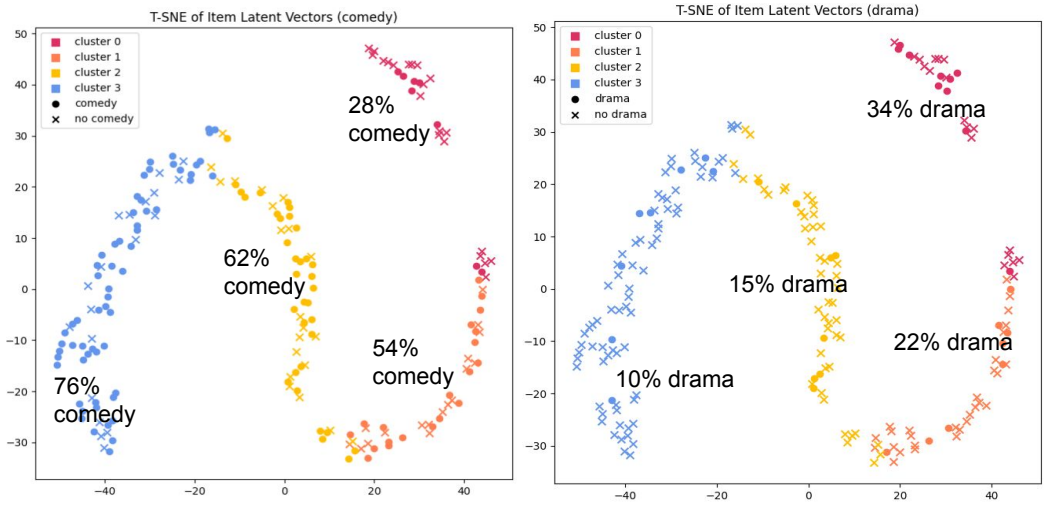


Figure 5: T-SNE projection of the item latent vector learns by our Bayesian model, the colors are from K-means cluster with 4 clusters. Note that the model does not accept as input the item genres and the latent space is learnt solely from the ratings. Left: The percentage of Comedy shows in each cluster. Right: The percentage of Drama shows in each cluster. The red cluster has the highest concentration of Drama shows and lowest concentration of Comedy shows. The relationship is reversed in the blue cluster, while the orange and yellow clusters are in-between.

4.1.2 Runtime

An important dimension to models is their execution time (wall time). Bayesian models are notorious for taking a large amount of time and our model is no different. Figure 4b illustrates the runtime of all models in the benchmark. Evidently, the Bayesian model takes the longest amount of time with almost 1.5 minutes to compute the posterior and sample via NUTS. This is a clear drawback to our model. Moreover, when testing the runtime of our model under varying the number of users and the number of items, we see how badly the computational cost becomes. As we double the amount of users, the wall time almost doubles as well (from 1.5 minutes to slightly under 3). More scalability experiments are shown in Figure 7 in the Appendix. This presents a serious challenge to the usability of our model.

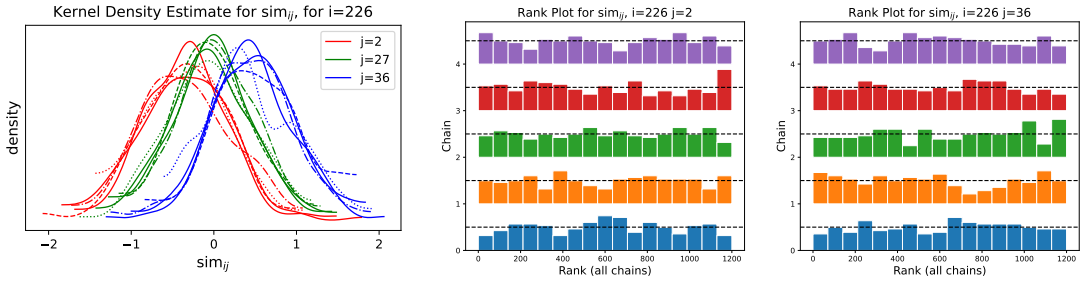
4.2 Interpretability

We attempt to investigate the latent space that the model learns for the items, we take the latent vectors for each item $L_j \in \mathbf{R}^d$ where $j \in [1, m]$ and we calculate the cosine similarity between each pair of vectors which results in a $\mathbf{R}^{m \times m}$ similarity matrix such that row j is the similarity between the latent vector of item j and all others item latent vectors[6]. We perform a T-SNE projection of the similarity matrix and apply K-means clustering to produce the result in Figure 5. The two plots in Figure 5 represent the percentage of items in each cluster for two item features that we randomly chose (comedy and drama). We find that the different clusters have very different distributions of item genres such that the red cluster has a much higher concentration of drama items and lower comedy items compared to the other clusters. This trend reverses as we move from the red cluster to the orange, yellow, and then finally blue where the trend flips.

We emphasize that the model does not accept as input the item genres and that the latent space is learnt solely from the ratings.

[6]We obtain this similarity matrix for each chain and each timestep of our sampling. We take the average of all of those.

4.3 Convergence



(a) (b)

Figure 6: (a) Kernel density estimates for *sim* values. (b) Rank plot.

To check that our model has converged and sufficiently explored the parameter space, we plot the kernel density estimate (KDE) of certain values in the chain across all time steps, such as in Figure 6a where we plot the KDE of the variable sim_{ij} from Eq. (4) for certain random values (here we chose the user $i = 226$ and items $j = \{2, 27, 36\}$). We note that the KDE of sim_{ij} for each of the 5 chains is plotted with a slightly different linestyle and that we can confirm that the model hasn't converged if the different KDE's for the same variable sim_{ij} do not look alike. In Figure 6a we can more confidently say that our model has converged and predicts that user $i = 226$ seems to slightly dislike item $j = 2$ while liking item $j = 36$, with $j = 27$ in-between. As another diagnostic, we use rank plots as mentioned in [6] and displayed in Figure 6b to check whether the chains have sufficiently mixed which is indicated by the different chains having close to uniform rank plots. Finally we note that our model does not appear to have any diverging samples as reported by NUTS.

5 Conclusion

In this project, we develop a Bayesian model for predicting which users would like an item given data of previously liked items. We leverage the power of Bayesian learning to build a latent model that can capture the latent features of users and items and predict with a given confidence whether a given user would like an item. We have evaluated our model with several other benchmark models including Multilayer Perceptrons (MLP) and gradient boosting algorithms (XGB). Our model consistently achieves good results when compared to these benchmarks and outperforms them when the model is confident.

One drawback we have constantly identified with our proposed Bayesian model is the increasing time complexity as the data grows which causes the failure to scale to extremely large datasets. One future improvement could be to parallelize the training of the model such that it can handle orders of magnitude more data. We hypothesize that this parallelization could be successfully achieved through alternate training of either the set of item or user feature vectors while keeping the other fixed. Holding one of the two fixed means that all the the model parameters that are currently training are mutually independent which can thus be split into different machines to run in parralel. We leave this as future work.

References

[1] Anghel, A., Papandreou, N., Parnell, T. P., Palma, A. D., and Pozidis, H. Benchmarking and optimization of gradient boosted decision tree algorithms. *CoRR abs/1809.04559* (2018).

[2] Betancourt, M. A conceptual introduction to hamiltonian monte carlo. *arXiv preprint arXiv:1701.02434* (2017).

[3] Condliff, M. K., Lewis, D. D., Madigan, D., and Posse, C. Bayesian mixed-effects models for recommender systems. In *ACM SIGIR* (1999), vol. 99, Citeseer, pp. 23–30.

[4] Hoffman, M. D., Gelman, A., et al. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res. 15*, 1 (2014), 1593–1623.

[5] Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer 42*, 8 (2009), 30–37.

[6] Vehtari, A., Gelman, A., Simpson, D., Carpenter, B., and Bürkner, P.-C. Rank-normalization, folding, and localization: An improved r for assessing convergence of mcmc (with discussion). *Bayesian analysis 16*, 2 (2021), 667–718.

[7] Zhang, S., Yao, L., Sun, A., and Tay, Y. Deep learning based recommender system: A survey and new perspectives. *ACM computing surveys (CSUR) 52*, 1 (2019), 1–38.

6 Appendix

	EachMovie		Anime		MovieLens		Synthetic	
Model	μ	σ	μ	σ	μ	σ	μ	σ
IM	0.679	0.017	0.665	0.011	0.679	0.016	0.619	0.035
UM	0.662	0.014	0.671	0.012	0.686	0.015	0.223	0.027
IUM	**0.722**	0.014	0.665	0.011	0.673	0.014	0.188	0.018
LR	0.558	0.058	0.488	0.023	0.589	0.025	0.518	0.022
XGB	0.667	0.015	0.726	0.018	0.668	0.019	0.831	0.008
MLP	0.704	0.022	**0.755**	0.015	**0.715**	0.017	**0.861**	0.011
Our Model	0.707	0.013	0.753	0.013	0.701	0.019	0.846	0.010

Table 2: Mean (μ) F1 Scores and standard deviation (σ) on test data with $users = 100$ and $items = 100$. Results were obtained from 20 independent runs.

	EachMovie		Anime		MovieLens		Synthetic	
Model	μ	σ	μ	σ	μ	σ	μ	σ
IM	0.744	0.022	**1.000**	0.000	0.982	0.006	0.463	0.045
UM	0.766	0.025	0.999	0.002	0.979	0.011	0.132	0.017
IUM	**0.818**	0.016	**1.000**	0.000	**0.998**	0.001	0.104	0.009
LR	0.580	0.114	0.423	0.022	0.602	0.035	0.527	0.062
XGB	0.653	0.023	0.718	0.023	0.656	0.026	0.825	0.008
MLP	0.697	0.039	0.765	0.057	0.742	0.046	**0.865**	0.037
Our Model	0.713	0.012	0.760	0.017	0.755	0.021	0.847	0.011

Table 3: Mean (μ) recall scores and standard deviation (σ) on test data with $users = 100$ and $items = 100$. Results were obtained from 20 independent runs.

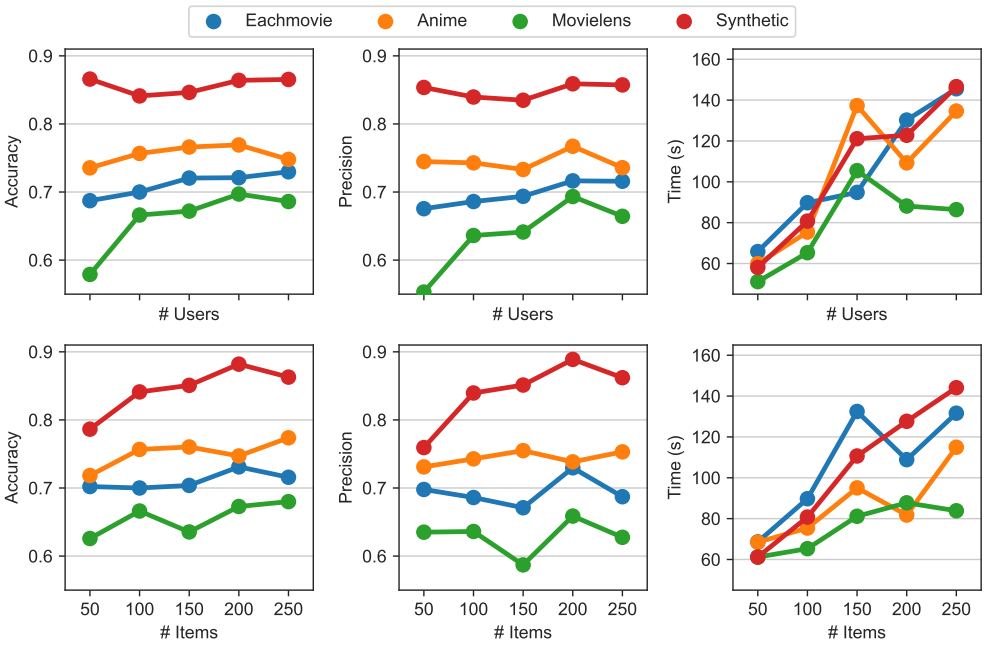


Figure 7: Scaling # users and # items and reporting accuracy, precision, and runtime. In the first row, we keep the # items fixed at 100. In the second row, we keep the # user fixed at 100. The performance (accuracy, recall) slightly increases as the training data increases, whether when looking at number of users or items. However, the runtime significantly increases as the dimension of the data increases.