Uncertainty Guided Recommendation with Bandits

Baccas, Shane [*]
shane.baccas@mail.utoronto.ca

Du, Shu Jian[*]
dushu@cs.toronto.edu

Lin, Franco Ho Ting[*]
francohtlin@cs.toronto.edu

Soon, Chee Loong[*]
cheeloong.soon@mail.utoronto.ca

Teoh, Wei Zhen[*]
weizhen.teoh@mail.utoronto.ca

Abstract

In this paper we provide a model for Instantaneous Feedback Recommendation Systems. The observed user ratings from the MovieLens dataset are recorded in a Ratings Matrix and the missing entries are obtained by factorizing this matrix using 2 approaches: Probabilistic Matrix Factorization (PMF) and Neural Network Matrix Factorization (NNMF). Both methods predict the missing entries of the Ratings matrix as a Gaussian whose mean is a function of dense latent matrices (with Gaussian priors) representing the item data and user data. The latent's posterior distribution parameters are approximated using Stochastic Variational Inference in both factorizations. We then recast the items in the ratings matrix as arms in the Multi-armed Bandits context whose predictive reward distributions are given by that of the dot product of two independent Gaussian vectors. We use these distributions in conjunction with different Bandit policy functions to provide a sequence of recommendations to a given user. Finally we measure the quality of our recommendation sequence using the Regret metric. We found that the exploitation strategy dominates in the original dataset, while exploratory approaches dominates in a modified version of the dataset that reflect more ambiguous user preferences.

1 Introduction

Recommender systems are used in a variety of different applications and websites today. YouTube, Amazon are examples of classic recommender systems that recommend items from a large inventory containing videos or products in those cases. We focus on a particular type of recommender system where it is relatively cheap to receive instant feedback from users on given recommendations, such as those found on Tinder or Spotify. We also assume that we do not have access to contextual information about the users such as age, gender, purchase history, web browsing history etc.

These types of recommendation systems learn user preferences solely through interacting with the user through recommendations and feedback. They are sequential because we recommend to the user one item at a time. Once we have determined that they like or dislike a particular item (i.e. right swipe/left swipe), we can update their preference distribution instantly and immediately improve the personalization of the next item.

[*]Equal Contribution

1.1 Recommendation Problem as an Exploitation-Exploration Trade-off

A brand new user u_i queries our recommendation system for an item. We have no information on what the user likes. We could recommend the most popular item we have as rated by our current user database (i.e. exploitation), or we could recommend random items from our inventory that have only few ratings to the new user, in an attempt to gauge the users preferences and gain more information about our inventory (i.e. exploration). If we just recommend the most popular item, we might not learn the user's true preferences. On the other hand, if we only recommend random items, we face the risk of recommending terrible items to our new user, thus deterring the user from ever using our service again. The bandits framework provides the perfect model to balance this exploration vs exploitation dilemma.

1.2 Related Work

Jeremie Mary et al, recast the recommendation system in the bandits context and solved the problem as a sequential decision making problem where at each step we are interested in maximizing the user satisfaction measured by their rating for an item we recommended[9]. This reconceptualization of the problem induces the problem of balancing exploration (gathering data on items user has not rated yet, thus improving our predictions) vs exploitation (recommending popular items that we are confident the user will rate favorably.)

A personalized news article recommender system proposed by [8] provides a novel selection algorithm (LinUCB) for contextual bandits which outperforms traditional methods. Although their problem is slightly different from ours as they incorporated context information for each user, the motivation lies in the bandit formulation of the recommender system.

Zhao et al, formulated a similar problem to that of ours, where a system continuously recommends items to individual users, receiving interactive feedback. They proposed a PMF framework with several selection algorithms, namely Thompson Sampling, LinUCB and Generalized LinUCB [12].

The classic recommendation system architecture uses regular matrix factorization where the sparse ratings matrix R is factored as the product of two dense latent matrices U and V. In Dziugaite and Roy's paper [3], they provided a new regime where instead of factoring R as the matrix product (a *linear* operation) of two latent matrices, the missing entries of R are modeled by a *non-linear* Neural Network. By using a non-linear Neural Network function instead of the matrix product, the idea is that we capture greater "nuance" in the sparse relational data and hopefully provide better predictions on the missing entries of R. Stolee et al. [11] provided an implementation of "SVINNMF", which is NNMF with Stochastic Variational Inference approximation on user and item latent variables. SVINNMF provides a Bayesian interpretation of the model. We base our probabilistic NNMF model on top of this approach.

Another Matrix Factorization algorithm, known as Probabilistic Matrix Factorization (PMF), was proposed by Salakhutdinov et al. [10]. Under the PMF architecture we model each missing entry of the ratings matrix as a Gaussian whose mean is taken as the empirical sample mean from the linear product of 2 dense latent matrices[10].

1.3 Main contribution

We seek to combine the ideas of these papers in a unified attack that uses the sparse data we have to refine our estimates in the predictive distribution for each entry and maximize user satisfaction while carefully retaining the delicate balance of exploration and exploitation in the Bandits framework. In particular, we recast each item as an individual bandit arm, whose reward is given by the observed user rating for that item.

We model the ratings which are missing in the Ratings matrix R in two ways. 1) NNMF: as Gaussians whose mean is given by the output of a Neural network parameterized by θ and whose input is our latent matrices U, V, U', V' whose prior distributions are Gaussians parametrized by a vector ϕ [3]. and 2) PMF: as Gaussians whose mean is given by the dot product of rows in our latent matrices U_i and V_j whose prior distributions are also Gaussians parameterized by a vector ϕ [10].

We optimize both the NNMF and PMF approaches using SVI on the vectors ϕ and θ for the NN in NNMF. We then use these distributions in the bandits context to recommend items to the user that

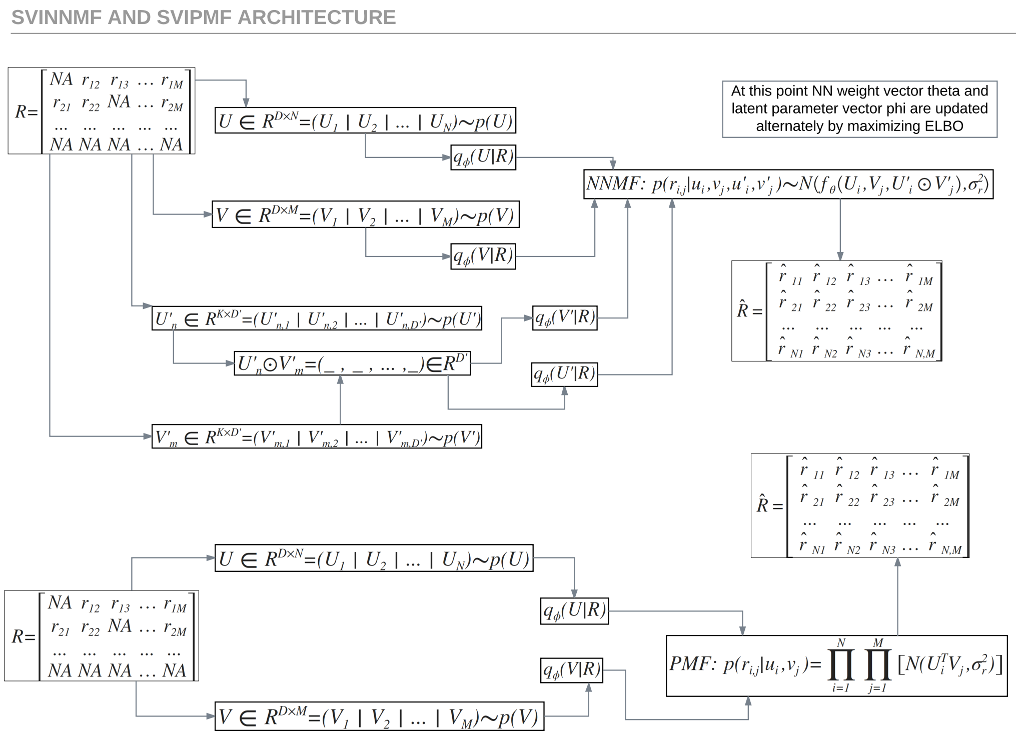
SVINNMF AND SVIPMF ARCHITECTURE
At this point NN weight vector theta and latent parameter vector phi are updated alternately by maximizing ELBO
$U \in R^{D\times N}=(U_1 \mid U_2 \mid \dots \mid U_N) \sim p(U)$
$V \in R^{D\times M}=(V_1 \mid V_2 \mid \dots \mid V_M) \sim p(V)$
$q_\phi(U|R)$
$q_\phi(V|R)$
$q_\phi(U'|R)$
$q_\phi(V'|R)$
$NNMF: p(r_{i,j}|u_i,v_j,u'_i,v'_j) \sim N(f_\theta(U_i,V_j,U'_i \odot V'_j),\sigma_r^2)$
$PMF: p(r_{i,j}|u_i,v_j)=\prod_{i=1}^{N}\prod_{j=1}^{M}[N(U_i^TV_j,\sigma_r^2)]$

Figure 1: Diagram of the SVINNMF model on top, and the SVIPMF model at the bottom.

they will rate favorably, balanced against gathering new data on items with few ratings in the optimal way. We apply the Thompson Sampling bandit selection strategies, as well as two novel strategies we will introduce, named "Empirical-UCB" and "Entropy-guided Acquisition". In addition, we compare these against selection strategies that do not require an uncertainty distribution: ϵ-Greedy, Boltzmann, and Exploitation.

We optimize both the NNMF and PMF models via Stochastic Variation Inference, therefore we call them the SVINNMF and SVIPMF models respectively. These models used in our approach differs from existing work because unlike the matrix factorization offered by [3] and [10] we obtain approximations of the underlying *distributions* of the latent parameters. The earlier approaches simply obtain MAP estimates for the latent parameters and use them to approximate the likelihood function of $r_{i,j}$. The explicit distribution approximation of the latent parameters, allow us to sample from them several times, and obtain a much more accurate approximation.

2 Model Description

We implement 2 different models to predict distribution over missing ratings on the rating matrix. These distributions are utilized in the Distribution Guided Bandit Selection strategies for selecting the next item to be recommended to a user. Refer to Figure 1 for a walkthrough of both of our models.

2.1 NNMF and PMF Matrix Factorization using SVI i.e. (SVINNMF and SVIPMF)

2.1.1 Definitions

- Let $R \in \mathbb{R}^{N\times M}$ be the Ratings Matrix for N users and M items where $r_{i,j}$ is the $(i, j)_{th}$ entry of R and is equal to the rating that user i assigns to item j.

- Let $U \in \mathbb{R}^{D\times N}$ be the dense latent matrix representing the user data.
- Let $V \in \mathbb{R}^{D\times M}$ be the dense latent matrix representing the item data.
- Let $V'_m \in \mathbb{R}^{K\times D'}$ be the dense latent matrix associated with column m of R. And let $U'_n \in \mathbb{R}^{K\times D'}$ be the dense latent matrix associated with row n of R. -For the purposes of our experiments we let $K = 1$ This was not the case in [3]. Effectively giving us two latent matrices for the user data and two for the item data.
- Let $f_\theta\left(U_i, V_j, U'_i \odot V'_j\right)$ be a Neural Network with $2D + D'$ inputs and θ weight vector. Whose output is the prediction of $r_{i,j}$ Where $\odot$ is the Haddamard product.
- In NNMF context Let $p(r_{i,j}|u_i, v_j, u'_i, v'_j) \sim N\left(f_\theta\left(U_i, V_j, U'_i \odot V'_j\right), \sigma_r^2\right)$ Its important to note that σ_r^2 is the Gaussian noise associated with the prediction of $r_{i,j}$ but this is separate and apart from the inherent noise associated with f_θ itself which arises from the variance in latent inputs distributions. In the Bandits context both of these variances will be important to make accurate approximations of the predictive distributions for $r_{i,j}$. In the PMF context we let $p(r_{i,j}|u_i, v_j) = \prod_{i=1}^{N}\prod_{j=1}^{M}\left[N(U_i^T V_j, \sigma_r^2)\right]$[10]
- Let $q_\phi(U|R), q_\phi(V|R), q_\phi(U'|R), q_\phi(V'|R)$ be our variational Gaussian approximations to the true posteriors given by $p(U|R), p(V|R), p(U'|R), p(V'|R)$ These posterior estimates are fully parameterized by the single vector ϕ:
- Finally we let $p(U), p(V), p(U'), p(V')$ be our prior distributions on the latent parameters with variances $\left\{\sigma_U^2, \sigma_V^2, \sigma_{U'}^2, \sigma_{V'}^2\right\}$ respectively.

2.1.2 SVI Approximation

For notational convenience we let $Z = (U, V, U', V')$. We are interested in approximating the latent posterior distribution $P(Z|R)$. In order to do this we use the Variational Lower Bound for Mean-field Approximation. We postulate a family of Gaussian distributions $Q_\phi(Z|R)$ fully parameterized by ϕ and we try to minimize the distance between $Q_\phi(Z|R)$ and $P(Z|R)$ over ϕ. We measure this "distance" using the reverse KL Divergence, and seek to minimize:

$$\mathbf{KL}[Q_\phi(Z|R)||P(Z|R)] \tag{1}$$

We know this is equivalent to maximizing the Evidence Lower bound (ELBO) given by

$$ELBO = \mathbf{E}_q[\log P(R|Z)] - KL[Q_\phi(Z|R)||P(Z)] \tag{2}$$

The Expectation term is referred to in the literature as the Reconstruction and the second term is referred to as the KL divergence from the prior. [5]

2.1.3 Estimating the Reconstruction term for SVINNMF

We will approximate $\mathbf{E}_q[\log P(R|Z)]$ using a Monte Carlo estimate with T samples. Giving us:

$$\mathbf{E}_q[\log P(R|Z)] \approx \frac{1}{T}\sum_t^T \sum_{i,j} \log p(r_{i,j}|u_{i,t}, v_{j,t}, u'_{i,t}, v'_{j,t}) \tag{3}$$

and since $p((r_{i,j}|u_i, v_j, u'_i, v'_j)$ is Gaussian we have

$$\mathbf{E}_q[\log P(R|Z)] \approx -\frac{1}{\sigma_r^2}\frac{1}{T}\sum_t^T \sum_{i,j}\left(r_{i,j} - f_\theta(u_{i,t}, v_{j,t}, u'_{i,t}, v'_{j,t})\right)^2 \tag{4}$$

2.1.4 Estimating the KL Divergence of Q_ϕ from the prior for SVINNMF

$$\begin{aligned} -KL[Q_\phi(Z|R)||P(Z)] = -KL[Q_\phi(U|R)||P(U)] - KL[Q_\phi(V|R)||P(V)] \qquad (5)\\ -KL[Q_\phi(U'|R)||P(U')] - KL[Q_\phi(V'|R)||P(V')] \qquad (6)\end{aligned}$$

Consider the first term:

$$\begin{aligned} -KL[Q_\phi(U|R)||P(U)] &= -\int q_\phi(u|r)\log(\frac{q_\phi(u|r)}{p(u)}) & (7)\\ &= -\int q_\phi(u|r)\big[\log(q_\phi(u|r) - \log(p(u))\big] & (8)\\ &= -\sum_{i,d}\left[\log(\frac{\sigma_U}{\sigma_{i,d}}) + \frac{\sigma_{i,d}^2+\mu_{i,d}^2}{2\sigma_U^2}\right] & (9)\end{aligned}$$

We perform similar calculations and obtain similar results for the other terms and altogether we maximize :

$$\begin{aligned} ELBO &= \mathbf{E}_q[\log P(R|Z)] - KL[Q(Z|R)||P(Z)]\\ &= -\frac{1}{\sigma_r^2}\frac{1}{T}\sum_t^T\sum_{i,j}\big(r_{i,j} - f_\theta(u_{i,t}, v_{j,t}, u'_{i,t}, v'_{j,t})\big)^2\\ &\quad -\sum_{i,d}\left[\log(\frac{\sigma_U}{\sigma_{i,d}}) + \frac{\sigma_{i,d}^2+\mu_{i,d}^2}{2\sigma_U^2}\right] - \sum_{j,d}\left[\log(\frac{\sigma_V}{\sigma_{j,d}}) + \frac{\sigma_{j,d}^2+\mu_{j,d}^2}{2\sigma_V^2}\right]\\ &\quad -\sum_{i,d'}\left[\log(\frac{\sigma_{U'}}{\sigma_{i,d'}}) + \frac{\sigma_{i,d'}^2+\mu_{i,d'}^2}{2\sigma_{U'}^2}\right] - \sum_{j,d'}\left[\log(\frac{\sigma_{V'}}{\sigma_{j,d'}}) + \frac{\sigma_{j,d'}^2+\mu_{j,d'}^2}{2\sigma_{V'}^2}\right]\end{aligned}$$

over θ and ϕ using Stochastic Gradient Descent.

2.1.5 Maximizing ELBO for PMF model

For PMF the optimization is simplified because there is no NN weight vector θ because we are using the dot product of rows in our latent matrices. Therefore we only need to train our Gaussian parameter vector ϕ. Which we can do by maximizing log posterior of:

$$p(r_{i,j}|u_i, v_j) = \prod_{i=1}^N\prod_{j=1}^M[N(U_i^TV_j, \sigma_r^2)]$$

and this is equivalent to minimizing:

$$E = \frac{1}{2}\sum_{i=1}^N\sum_{j=1}^M(R_{i,j} - U^TV_j)^2 + \frac{\lambda_U}{2}\sum_{i=1}^N||U_i||_{Frob}^2 + \frac{\lambda_V}{2}\sum_{j=1}^M||V_j||_{Frob}^2$$

Where $\lambda_U = \sigma^2/\sigma_U^2, \lambda_V = \sigma^2/\sigma_V^2$ and $||.||_{Frob}$ is the Frobenius norm.[10]

3 Bandit Selection Strategies

Now that we have probability distributions for missing entries in our Ratings matrix, we can recast each item as a bandit arm with reward equal to the ratings given by the users. The following section will go over simple definitions and the two classes of selection strategies.

3.1 Definitions

- We denote the true Ratings Matrix as R and the predicted ratings matrix with probability distributions as $\hat{R}$
- We consider a multi-armed bandit problem with M independent arms, where each arm corresponds to an item. In this context we consider the user i fixed and refer to the predictive distribution $\hat{r}_{ij}$, simply as $\hat{r}_j$
- Let us denote J as the item selected and the reward $reward(J) = r_J$ to be the true rating given by user I for item J.
- The optimal item I is given by the highest rated item by the user $J^* := \underset{j=1,\cdots,M}{\operatorname{argmax}}\,(r_j)$.
- Unlike other multi-armed bandit problems, the number of times each arm has been pulled (item has been selected) is omitted. Once that item has been recommended to a user, it is not useful to recommend it again.
- The set of items M is reduced as we recommend more items to the user, and the user provides ratings for the recommended items.

3.2 Distribution Guided Approach

The following family of selection algorithms utilize the predictive distributions for each item. In previous work the probability distribution for each arm was estimated on an ad-hoc basis using some confidence interval around point estimates. Since we used SVI to obtain an actual predictive distribution we need not perform this exercise. We obtain these distributions by sampling $N = 100$ times from our latent factors (U, V) for each user-item combination and compute the ratings for each sample. We will denote each sample rating for item j as $\hat{r}_j^{(n)}$

3.2.1 Empirical Upper Confidence Bound (E-UCB)

E-UCB is a proposed variant to the well-known UCB1 algorithm [1]. The difference lies in using the predictive distributions to acquire our confidence bounds. We assume that our predictive ratings are Gaussian and acquire the empirical mean and standard deviation as follows.

$$\hat{\mu}_j = \frac{1}{N}\sum_{n=1}^{N} \hat{r}_j^{(n)} \quad \text{and} \quad \hat{\sigma}_j = \sqrt{\frac{1}{N}\sum_{n=1}^{N} (\hat{r}_j^{(n)} - \hat{\mu}_j)} \tag{10}$$

For user I, we select the item with the highest empirical confidence bound, given some tuning parameter α which controls the magnitude of exploration.

$$J = \underset{j}{\operatorname{argmax}}(\hat{\mu}_j + \alpha\hat{\sigma}_j) \tag{11}$$

3.2.2 Thompson Sampling

A variant of Thompson Sampling is used since we are able to acquire the predictive distributions. For user I, we simply take a single sample of the latent factors (U, V) and recommend the item with the highest sample rating.

$$J = \underset{j}{\operatorname{argmax}}\, \hat{r}_j^{(1)} \tag{12}$$

3.2.3 Entropy-guided Acquisition

We have designed an acquisition policy to improve expected information gain on the user and item latent factors at every acquisition step. We assume that the uncertainty about the latent variables is captured by the entropy function (denoted by H). Information gain is thus defined as the change in entropy of the of the latent variables conditional on the new observations. The expected information gain (G) can thus be defined as the mutual information (I) between acquired rating variable r_j and the latents Z.

Let $Z = (U, V)$ represents the latent variables, R_{obs} represents observed ratings. Assume that $Z \sim P(Z \mid R = R_{obs})$, $r_j \sim P(r_j \mid R = R_{obs})$:

$$\begin{aligned} G &= H(Z) - H(Z \mid r_j) \\ &= I(Z, r_j) \\ &= H(r_j) - H(r_j \mid Z) \\ &= H(r_j) - \frac{1}{2} log(2\pi e {\sigma_r}^2) \end{aligned} \tag{13}$$

The above implies that acquiring the rating that we are most uncertain about (highest entropy) is expected to give us the most information about the underlying latent variables across the users and items. Note also that since $r_j \sim N(\hat{r}_j, {\sigma_r}^2)$, and $\hat{r}_j$ is a function of random variable Z, the entropy of the rating is bounded below by the entropy of rating mean $\hat{r}_j$ and a constant term:

$$H(r_j) \geq H(\hat{r}_j) + \frac{1}{2} log(2\pi e {\sigma_r}^2) \tag{14}$$

$H(\hat{r}_j)$ thus defines a lower bound of the expected information gain. We can then derive an exploration-exploitation balanced acquisition policy for a new item J to be recommended to a user I as follow:

$$J = \underset{j}{\text{argmax}}[E(\hat{r}_j) + \lambda * H(\hat{r}_j)] \tag{15}$$

In our implementation we estimate the entropy and the mean of $\hat{r}_j$ empirically from the samples drawn.

3.3 Simple Heuristics Approach

The following algorithms do not make use of our predictive distributions. Instead, they use point estimates for item selection. We will take the empirical mean $\hat{\mu}_j$ from our $N = 100$ samples as our point estimates.

3.3.1 ϵ-Greedy

We choose $0 < \epsilon < 1$ and let $\xi \sim unif(0, 1)$ our policy is:

$$J = \begin{cases} \underset{j}{\text{argmax}} \hat{\mu}_j & \text{if } \xi < \epsilon \\ \text{unif}(1, M) & \text{otherwise} \end{cases} \tag{16}$$

effectively choosing the predicted optimal arm $1 - \epsilon$ of the times and a random item the other ϵ times. In contrast to the the Distribution Guided Methods, the ϵ-Greedy approach relies on random exploration but there are good empirical results to support its use. [7]

3.3.2 Boltzmann Exploration

Boltzmann exploration is another heuristic approach that maps our estimates for each item $\hat{\mu}_j$ to probabilities through the Boltzmann(Softmax) function. Additionally, we have a temperature scale τ which adjusts our rate of exploration. The probability of selection for each item j is given by:

$$p_j = \frac{e^{\hat{\mu}_j / \tau}}{\sum_{i=1}^{M} e^{\hat{\mu}_i / \tau}} \tag{17}$$

By increasing (decreasing) τ, we are increasing (decreasing) the probability of selecting the item with the highest predicted rating. For higher exploitation (exploration), a smaller (larger) value of τ is used. The benefit of using Boltzmann Exploration, is that the estimated optimal item is most likely but not guaranteed to be selected allowing for exploration. Again, [7] showed good empirical results to support its use.

3.3.3 Exploitation Approach

The final acquisition function is to strictly exploit the structure of our model by selecting the item with the highest empirical mean with no exploration factor.

$$J = \underset{j}{\text{argmax}} \hat{\mu}_j \tag{18}$$

4 Ranking Evaluation

We randomly selected $n_u = 20$ dense test users and selected $n_i = 50$ items for each test user. We finally calculated the mean of the results over 50 items for all 20 users as shown in figures 5, 6, 7, 8. We chose these numbers due to time constraints of running the models. The full run takes about 5 hours on an Nvidia 1080 Ti Pascal Architecture GPU. We treated each test user as a cold-start user by initializing their items seen to none. This way, we ensure the dense users have at least 50 items that we can sequentially iterate over. The order that the items are picked for each user can be transformed into ranking problem, whereby the items are ranked based on the order in which they were selected. We evaluated our algorithms using a traditional ranking approaches and regret based approaches.

4.1 Positive Negative Ranking

Traditional ranking approaches requires a binary positive and negative labels on the results.

We treated ratings 1 and 2 as negatives and ratings 3, 4, 5 as positives.

We only evaluated based on the first K predictions where we chose K = 10.

We did not use $K = n_i = 50$ as we needed a way to define negative predictions in order to calculate recall and wanted to be consistent with $K = 10$ used in rec@K.

The total number of available $condition\ positives$ are defined as the number of ratings for each user that is rated ≥ 3. However, during evaluation of rec@K, we calculated the $condition\ positive$ as $condition\ positive@k = min(K, count(n_i \geq 3))$. This ensures the rec@$K$ is a value within $[0, 1]$.

Therefore, if a user has rated 80 items, but only 8 of those ratings are ≥ 3, then that user's $condition\ positive$ is 8. If 12 of those ratings are ≥ 3, then that user's $condition\ positive$ is $K = 10$.

We define the number of $prediction\ positives$ as k itself as we assume the first k predictions are in fact the positive predictions. The remaining $n_i - k$ are assumed to be the negative predictions as they are lower in the rank. $prediction\ positives@k = k$

We define the number of $true\ positives@k$ as the number of ratings for a user which is ≥ 3 for the first k predictions. Using python's numpy notation, $true\ positives@k = n_i[0 : k] \geq 3$

We implemented both $mAP@k$ and rec@K.

A useful future work would be to also implement the Normalized Discounted Cumulative Gain (NDCG) as it is also popularly used to evaluate ranking for recommender systems applications.

4.1.1 Mean Average Precision at K

We implemented mean average precision at K (mAP@K) as shown in equation 19.

For each user, we first calculate the precision at k, ($prec@k$). We then calculate the average $prec@k$ ($AP@K$) by iterating over k from 1 to K. Finally, we calculate the mean of the $AP@k$, ($mAP@K$) over all n_u users.

$$\begin{aligned} prec@k &= \frac{true\ positive@k}{prediction\ positive@k} \\ AP@K &= \frac{1}{K}\sum_{k=1}^{K} prec@k \\ mAP@K &= \frac{1}{n_u}\sum_{u=1}^{n_u} (AP@K)_u \end{aligned} \tag{19}$$

4.1.2 Recall at K

We also implemented recall at K, (rec@K) as shown in equation 20.

$$rec@k = \frac{true\,positive@k}{condition\,positive@k} \tag{20}$$

4.2 Regret

We can also calculate the order of the picked choices as a regret metric.

4.2.1 Optimal Regret

For each test user, we calculate the optimal regret based on a discount factor, $\gamma = 0.99$. Here, we assume we can only regret about each item not being picked once.

We calculate the $user\ regret$ for a test user based on the ratings of the order the items were pick where $reward(J_t)$ is the value of the rating matrix for the item picked J for the user at time step t. The user's regret calculation is shown in equation 21.

$$user\ regret = \sum_{t=1}^{n_i} \gamma^{t-1} * reward(J_t) \tag{21}$$

The best item to pick for a test user at time step t given M_t the items we can select from is shown in equation 22.

$$J_t^* = \underset{j \in M_t}{\text{argmax}}[reward(j)] \tag{22}$$

The best optimal regret for a test user would be to first sort the ratings of that user and pick the items from highest rating to lowest. In other words, the user always picks the highest available rating first. This is calculated in equation 23.

$$max(regret) = \sum_{t=1}^{n_i} \gamma^{t-1} * reward(J_t^*) \tag{23}$$

As a regret should be 0 if we always pick the best sequence of items for a test user (hence, having no regrets), we take the difference between the max(regret) the user's regret as shown in equation 24.

$$optimal\ regret = max(regret) - (user\ regret) \tag{24}$$

4.2.2 Instantaneous Regret

Here, we assume that we can continue regretting as long as we are not picking the best available rating for an item over multiple time steps. This approach is similar immediate regret as done in [9].

The instantaneous regret shown below takes an order of magnitude longer to compute compared to the optimal regret as we will need to compute the maximum available item at each time step. However, in order to be able to plot the the cumulative regret over discrete time sequences, we will need to compute the instantaneous regret. This graph is meaningful as at every timestep, we are able to see the marginal regret that occur for not picking the best available item. If the the graph is a horizontal line, it means we are consistently picking the best available item for a user.

$$instantaneous\ regret = reward(J_t^*) - reward(J_t) \tag{25}$$

where J_t^* is the optimal item available at time t and $reward(J_t)$ is the rating of the selected item J_t. We punish the system whenever it makes a recommendation that is suboptimal to the current available best rating.

4.2.3 Cumulative Instantaneous Regret (CIR)

In order to evaluate our Regret over time, we use a cumulative instantaneous regret which is equal to 26

$$cumulative\ instantaneous\ regret = \sum_{t=1}^{n_i}(instantaneous\ regret) \tag{26}$$

5 Experimental Setup

We're interested in the problem of recommending a sequence of items to a user with the least cumulative regret. To this end, we proceed by first selecting a user, then pretend each item that we can recommend to the user is a "bandit arm". Note that the selected user may have viewed some items already, or may have not seen any items at all. Next, we use SVIPMF or SVINNMF to obtain a distribution of ratings for each item that we can recommend to the user, and use a bandit algorithm to pick one of those items. After we reveal the selected item to the user and obtain a rating, we remove that item from future recommendations. Then, we retrain our model to update the rating distributions for the remaining items, and repeat this procedure some finite number of steps or until no more items are left for a current user.

In practice, retraining the model from scratch after each item is too slow, and prohibits us from running any meaningful experiments. Instead, we opted to split this into two phases: Pretraining and Finetuning. In the Pretraining phase, we train the model for a fixed number of iterations (eg. 2000). Then during testing, upon seeing a new set of ratings from a user, we load the pretrained model and train the model for a small number of iterations (eg. 500); we call this the Finetuning phase. Finetuning allows the model to incorporate newly observed ratings while keeping the running time low. Our complete experimental procedure is described in Algorithm 1.

Algorithm 1 Experiment Procedure

```
Require: Rating matrix $R$, random seed.
  Set random seed.
  Select a set of users $T$ for testing, corresponding to a set of rows $R_{T,:}$
  for Model $M \in \{\text{SVIPMF}, \text{SVINNMF}\}$ do
    Let $R_{\text{pretrain}} = R \setminus R_{T,:}$.
    Pretraining: Train $M$ on $R_{\text{pretrain}}$ for 2000 iterations, save the model.
    for Bandit algorithm $B \in \{\epsilon - \text{Greedy}, \text{UCB}, \cdots\}$ do
      for each user $u \in T$ do
        Load M from just after Pretraining.
        Let $R_{\text{train}} = R_{\text{pretrain}}$.
        Let $R_{\text{test}} = R_{\text{u},:}$.
        while $R_{\text{test}}$ is not empty do
          Finetuning: Train $M$ on $R_{\text{train}}$ for 500 iterations.
          Obtain recommended item from bandit algorithm $x = B(M, R_{\text{test}})$.
          Compute and store instantaneous regret: $r = \max(R_{\text{test}}) - x$.
          $R_{\text{train}} = R_{\text{train}} \cup \{x\}$.
          $R_{\text{test}} = R_{\text{test}} \setminus \{x\}$.
        end while
      end for
    end for
  end for
```

5.1 Hyperparameters

We only conducted a limited amount of hyperparameters search due to limited computing power. Our final SVINNMF model used 60 dimensions for U and V, 35 dimensions for U' and V', and 3 layers of fully connected neural network with 50 neurons in each. Our final SVIPMF model used 10 dimensions for U and V; we found that higher dimensions only provided marginal improvement in terms of MSE and qualitative uncertainty estimates.

For the bandit selection strategies, there are a few hyperparameters that can be tuned. In the results, we set the exploration parameter in E-UCB to be $\alpha = 0.5$ for equation 11. For Entropy Guided Acquisition, we selected $\lambda = 0.5$ in equation 15. In the simple heuristics approaches, we selected $\epsilon = 0.1$ for ϵ-Greedy in 16 and $\tau = 0.1$ for the temperature scale in Boltzmann Exploration in equation 17.

5.2 MovieLens dataset

MovieLens provides a rating matrix with integer ratings from 0, 1, 2, 3, 4, 5. 0 signifies that the user has not rated a given movie. As we required actual user feedbacks and are unable to simulate user behavior, we chose to ignore all 0 ratings.

Instead, we mask out a portion of given ratings and only allow picking from those ratings so that we get a non-zero feedback from each bandit choice.

For ranking evaluations such as Mean Average Precision, we treated ratings 1 and 2 as negative ratings, and ratings 3, 4, 5 as positive ratings.

We chose the 100k MovieLens dataset as we are limited by computational resources [4].

In addition to using the entire ML100k dataset, we also tried our experiments on a smaller dataset derived from ML100k, which we call the “Dense" dataset. We were motivated to create this dataset because our initial experiments on the original dataset showed that bandits did not perform better than the Exploitation strategy, so we wanted to explore in which situations would bandits perform better. First, we observed that there are many movies that are considered universally popular or unpopular (by more than 80% of users); we removed these so good uncertainty estimates would play a more important role, and Exploitation cannot “cheat” by simply recommending popular movies. Second, we removed rows that contained less than 10% items or columns that contained less than 25% users. This step makes the matrix more dense, in order to make learning easier for the uncertainty models. Figure 2 shows the data distributions before and after these two modifications.

	Original (943 users $\times$ 1682 items)		**Dense (231 users $\times$ 135 items)**	
Rating	Count	Percentage of Ratings	Count	Percentage of Ratings
No Rating	1486126	-	18927	-
1	6110	0.06	883	0.07
2	11370	0.11	2025	0.17
3	27145	0.27	4173	0.34
4	34174	0.34	3791	0.31
5	21201	0.21	1386	0.11

Figure 2: Distribution of ratings in the original MovieLens 100k dataset (Left) and the Dense version after we removed polarized ratings and removed mostly empty rows and columns (Right).

6 Experimental Results

6.1 Point Estimates

Before we can use SVIPMF and SVINNMF to provide uncertainty estimates for bandits, we first check that they achieve competitive results in the classic matrix factorization task. We randomly split 70% of the data into a training set and 30% into a test set. The point estimate of PMF and SVINNMF are taken as the means of the predicted rating distributions. The results on the test set are shown in Figure 3.

6.2 Bandit Recommendations

Our first experiment used SVINNMF, however we found that results showed that distribution based strategies (E-UCB, Thompson Sampling, and Entropy-guided) did not perform better than ϵ-greedy. This led us to suspect that SVINNMF was not doing a good job of modeling uncertainty. Indeed,

Method	Test MSE
SVIPMF	0.958
SVINNMF	0.918
NNMF	0.919

Figure 3: Performance of various matrix factorization methods on the 100k MovieLens dataset. The Test MSE was evaluated on a 20% random holdout set. Each model was trained for 2000 iterations. The SVIPMF and SVINNMF parameters are those found in Section 5.1, and the NNMF parameters are taken from [3]. We weren't able to reproduce the MSE from the original NNMF paper, although our MSE isn't far off. Note that SVINNMF and SVIPMF both achieve competitive results.

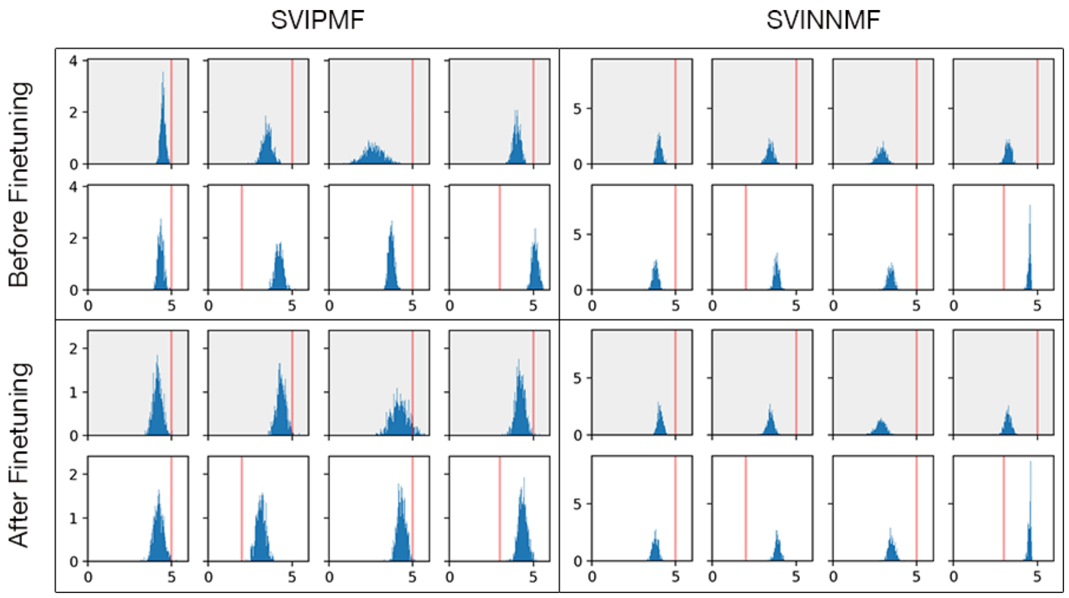

Figure 4: Each block of 8 plots represent the rating distribution of 8 randomly sampled items from a randomly sampled user (user 37, in this case). Rows 1 and 2 show the items' rating distributions generated by the SVIPMF and SVINNMF models after training for 2000 iterations (the "pretraining" phase). Rows 3 and 4 show the same items' rating distributions after the models were shown 10 more items, and trained for an additional 500 iterations (the "finetuning" phase). Rows 1 and 3 show four of the 10 ratings used in finetuning, rows 2 and 4 show four other ratings that were never shown to the models. The red lines indicate where the true ratings are. Observe that the rating distribution of SVINNMF does not change much after finetuning, while that of SVIPMF change significantly to adapt to the newly observed ratings, and it even improves the distributions on unseen ratings.

we found that SVINNMF does not adapt its uncertainty estimates well after finetuning (Figure 4), which suggests that perhaps it is stuck in a local minima after the pretraining phase. Furthermore, we observed that the entropy of the SVINNMF's latent distributions (U, V, U', V') do not change much after finetuning, which suggests that perhaps the latents are not being learned fast enough. On the other hand, we saw that SVIPMF adapts its uncertainty estimates much more readily to new information, and is much faster to train (around 8x faster). Therefore, we decided to proceed with SVIPMF for the rest of the experiments, and leave it as future work to adapt SVINNMF for this task.

Applying SVIPMF On the full data set, the pure Exploitation strategy, Boltzmann, and ϵ-Greedy all performed comparably in terms of average cumulative regret measure, as seen in Figure 7. However, it appears that all of the distribution based strategies still do not performed as well as Exploitation, see Figure 8. This prompted us to develop the Dense dataset described in Section 5.2.

Applying SVIPMF on the Dense data set, we observe Boltzmann exploration achieved significant improvement over pure exploitation, see Figure 5. The optimism motivated strategy, E-UCB, and information gain motivated strategy, Entropy-guided acquisition, both exhibit marginal improvement over exploitation, see Figure 6.

In addition to cumulative regret, we also recorded the change in MSE for remaining unseen items, as well as the change in entropy of user latent variables. See Figures 9, 10, 11, 12. We found no observable difference in the MSE decay curve and entropy decay curves for user latent variables over time in either of the two data sets.

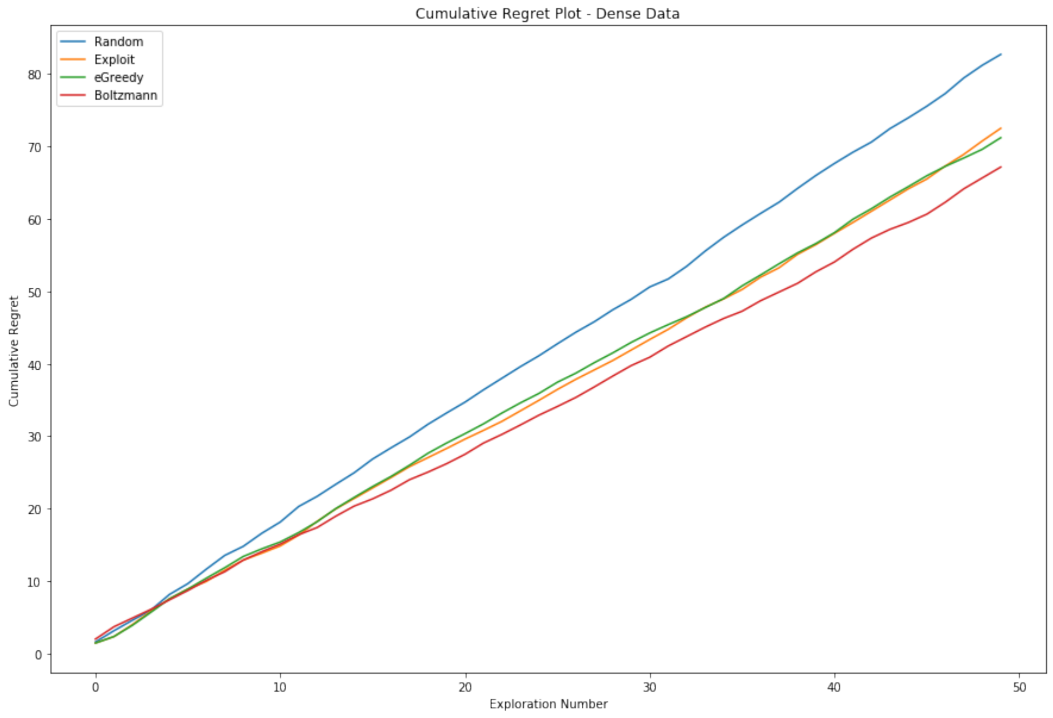

Figure 5: Average cumulative regret by heuristic strategies on the dense data set

6.3 Ranking Evaluation Results

Both results uses the PMF model. We show results for different bandit choices done on both Dense dataset and Full dataset.

I included worst and best just to give a bound between the worst and best possible values.

6.3.1 Dense

Metric	mAP@10	recall@10	$optimal\ regret$	CIR
Best	1.0	1.0	0.0	0
exploit	0.8220	0.795	86.92	1450
eGreedy	0.8170	0.775	93.64	1424
Boltzmann	0.7774	0.780	86.60	1343
UCB	0.8305	0.780	83.71	1391
Thompson Sampling	0.7653	0.735	103.34	1433
Entropy	0.8183	0.805	86.11	1406
Random	0.7974	0.745	102.48	1654
Worst	0.0661	0.080	159.92	2198

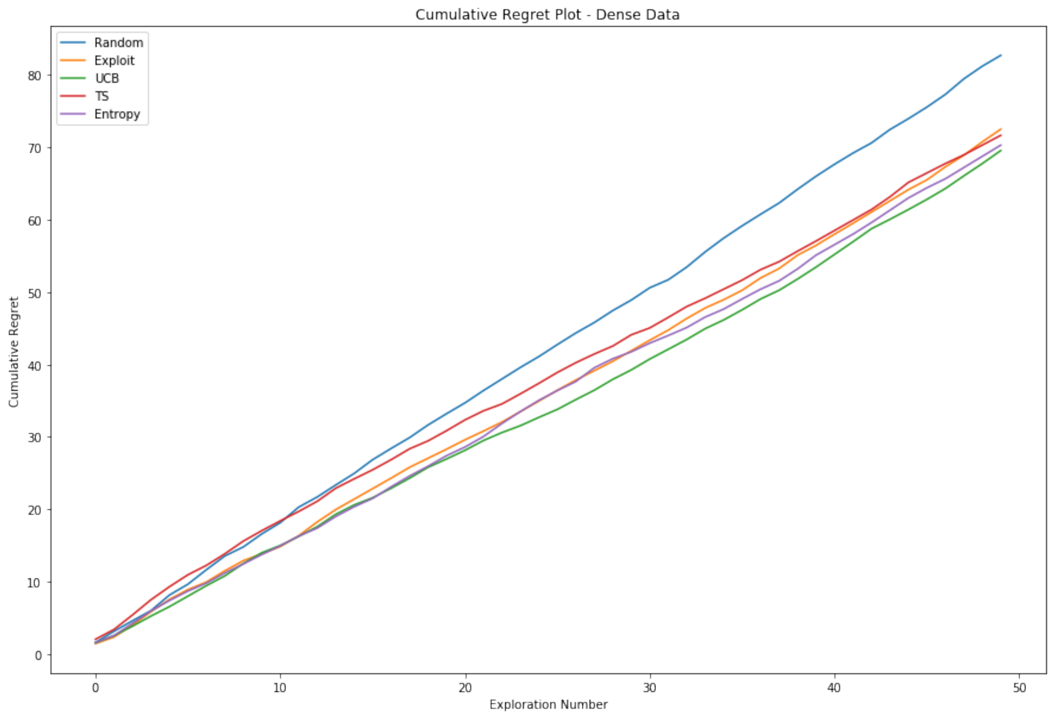

Figure 6: Average cumulative regret by distribution based strategies on the dense data set

6.3.2 Full

Metric	mAP@10	recall@10	*optimal regret*	*CIR*
Best	1.0	1.0	0.0	0
exploit	0.7919	0.810	101.72	1122
eGreedy	0.7902	0.805	100.35	1126
Boltzmann	0.8363	0.810	103.44	1128
UCB	0.8205	0.815	99.55	1182
Thompson Sampling	0.8400	0.815	110.45	1303
Entropy	0.8339	0.830	97.32	1205
Random	0.8694	0.820	102.53	1472
Worst	0.0108	0.020	105.07	2857

7 Discussion

We're not completely sure why distribution based bandit algorithms (E-UCB, Thompson Sampling (TS) , and Entropy-Guided) don't perform well on the full dataset. There could be many reasons. Our main suspect is that there are many universally popular movies in ML100k. Due to the way we split our data, the SVIPMF model can easily pick up on which movies are universally popular, and the Exploitation strategy can keep recommending those movies to achieve low regret. This was the main motivation behind why we developed the Dense dataset described in Section 5.2.

Interestingly, we found that the UCB and Entropy-guided strategies marginally outperform Exploitation on the Dense dataset. We believe this is the case because after removing universally popular and unpopular movies, we're left with ambiguously rated movies. This makes it more important to model uncertainty about user preference well, and to reduce uncertainty as quickly as possible. All 3 algorithms we tested can do this in theory. In particular, our proposed Entropy-guided strategy is designed with this implicit objective folded in. We see that it is competitive with the UCB algorithm. However, judging by the MSE and entropy decay curves, we do not see evidence that any strategy

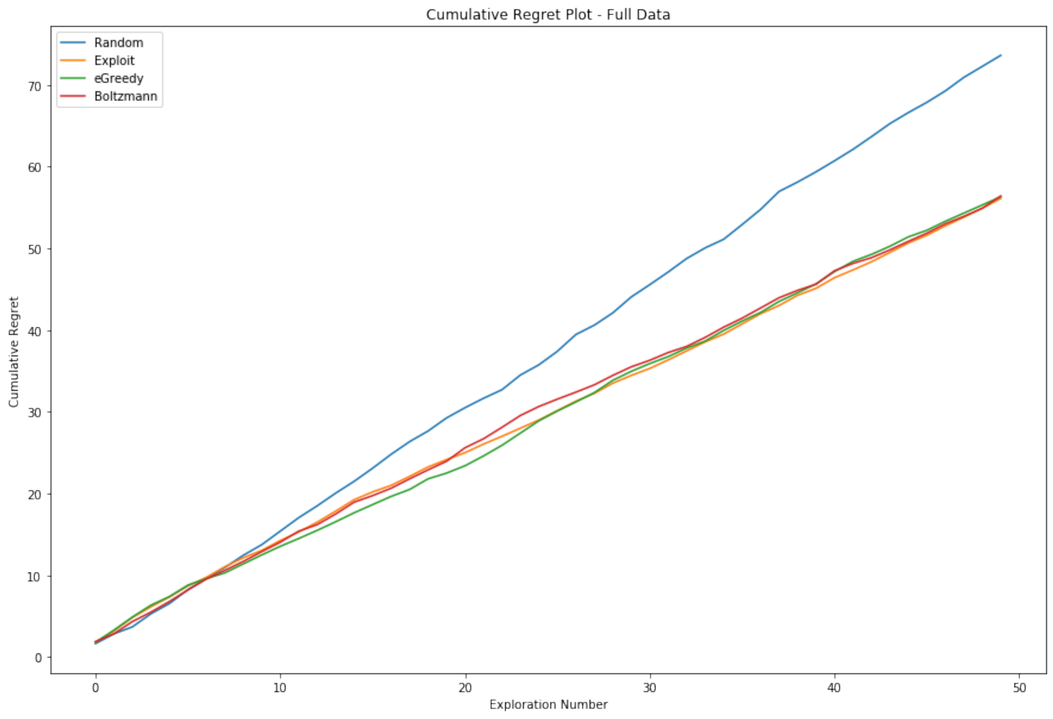

Figure 7: Average cumulative regret by heuristic strategies on the full data set

learns any faster than any other strategies. Therefore, we cannot say for sure which bandit strategy is better in this setting.

It might be the case that the problem we want to address is not a good fit for the bandit framework. In our setup, a bandit arm is a rating, hence it can only be "pulled" once, at which point it disappears. Even if we were able to find the optimal arm, we are unable to exploit this selection again.

It also appears that our model has limited capacity in terms of projecting the true predictive distribution of the ratings. In particular, since each rating mean is defined as the dot product of two independent multivariate normal random vectors, the rating mean distribution always falls under the family of generalized chi-squared distributions, which has very limited representation power (e.g. generally unimodal). Furthermore, this model is built upon the assumption that the true rating is a continuous random variable. However the MovieLens 100k rating data is actually discrete.

7.1 Future Work

We found it fascinating that while SVINNMF is very good at matrix factorization, it seems to be unable to adjust its uncertainty estimates to new data. Perhaps this is a problem with hyperparameter tuning. For instance, maybe the learning rate on the latent variables should be higher than the learning rate on the neural network. We think an interesting future direction is to investigate why this is the case.

In the previous section, we alluded to our concern that perhaps our experiment setup is not ideal for the bandit framework. One alternative is to pretend that each user is an arm. Every "pull" yields a random item rating from the user. Hence, we're interested in learning about a user's ratings distribution, instead of each item's distribution. This better models the classic setup of Multi-arm Bandits, but it addresses a different recommendation system problem than the one we intend to solve. Another alternative is to cluster users, and use each cluster as an arm. Our initial attempts to cluster the users using their latent variables were unsuccessful, but we think this may be an interesting future direction for research

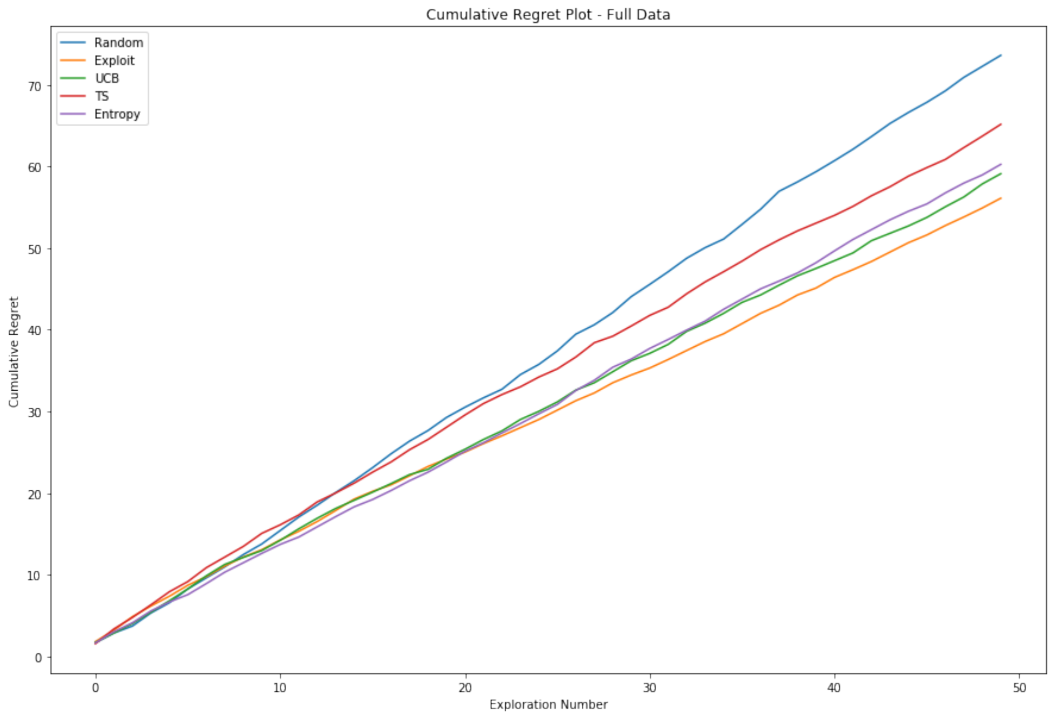

Figure 8: Average cumulative regret by distribution based strategies on the full data set

For many bandit selection strategies, the hyperparameter for exploration is decayed over time. [7] This should provide better item selection since after t number of recommendations, we should be more confident in our predictions and use more exploitative strategies.

Better bandit selection strategies could be used. Since we have probability distribution, some other selection strategies could use Bayes-UCB proposed by [6] or KL-UCB proposed by [2] An interesting method used by [12] is to hold the latent factors for items V fixed assuming the rewards to be a linear form and use the LinUCB algorithm for recommendation.

Finally, an extension of our experiments is to explore the behavior of the recommender system when we're allowed to recommend items for multiple users at the same time. For instance, it would be interesting to see if the system strategically recommends items that may not be optimal for a small set of users in order to minimize regret for a larger set of users, thereby minimizing the regret overall.

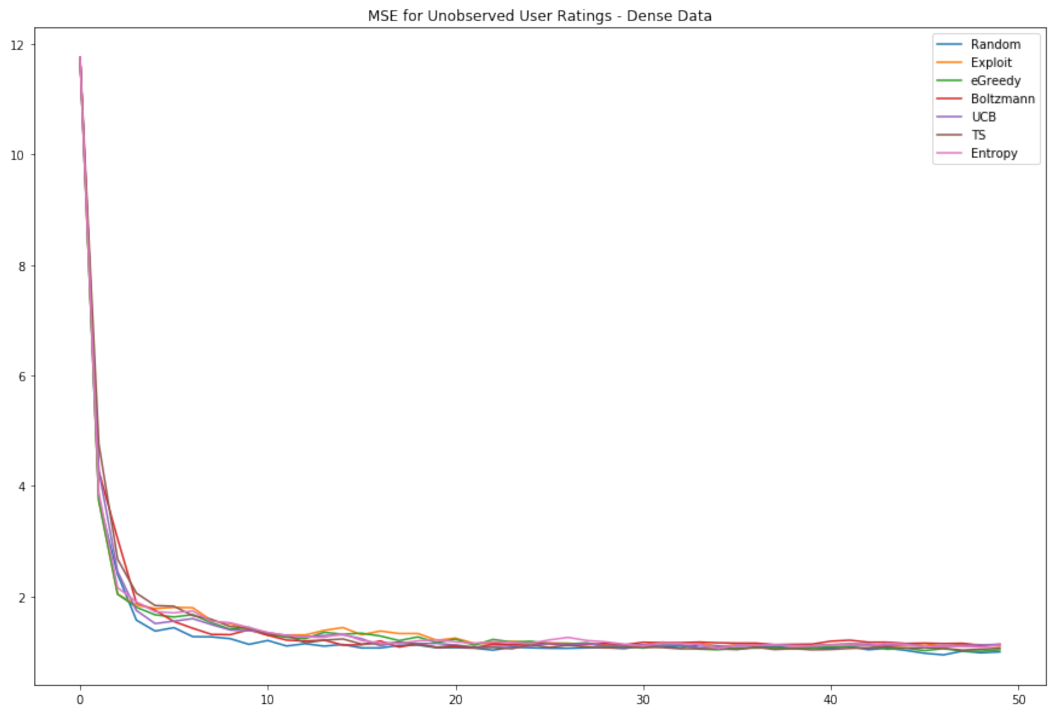

Figure 9: Average MSE on the unobserved item ratings over 50 time steps

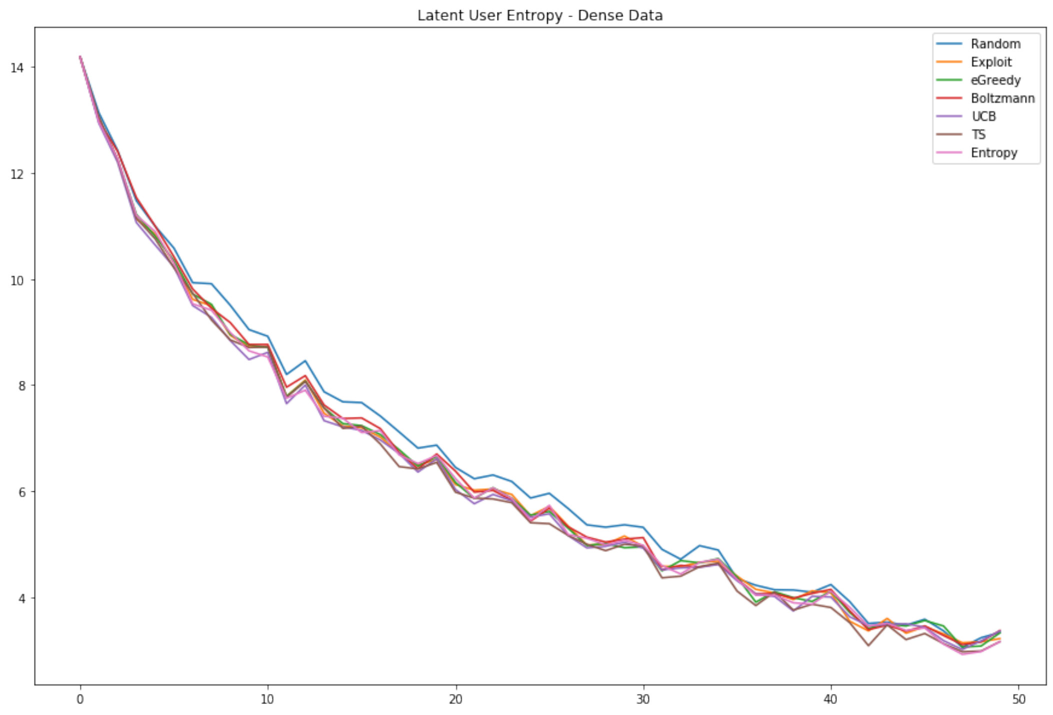

Figure 10: Average entropy of the user latent variables over 50 time steps

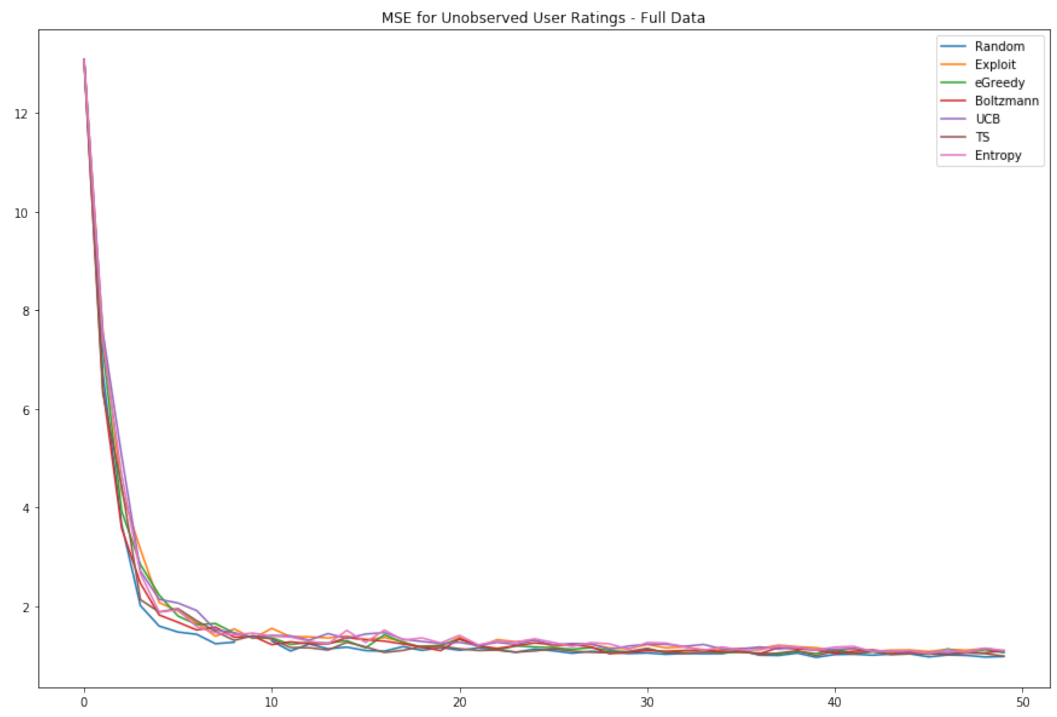

Figure 11: Average MSE on the unobserved item ratings over 50 time steps

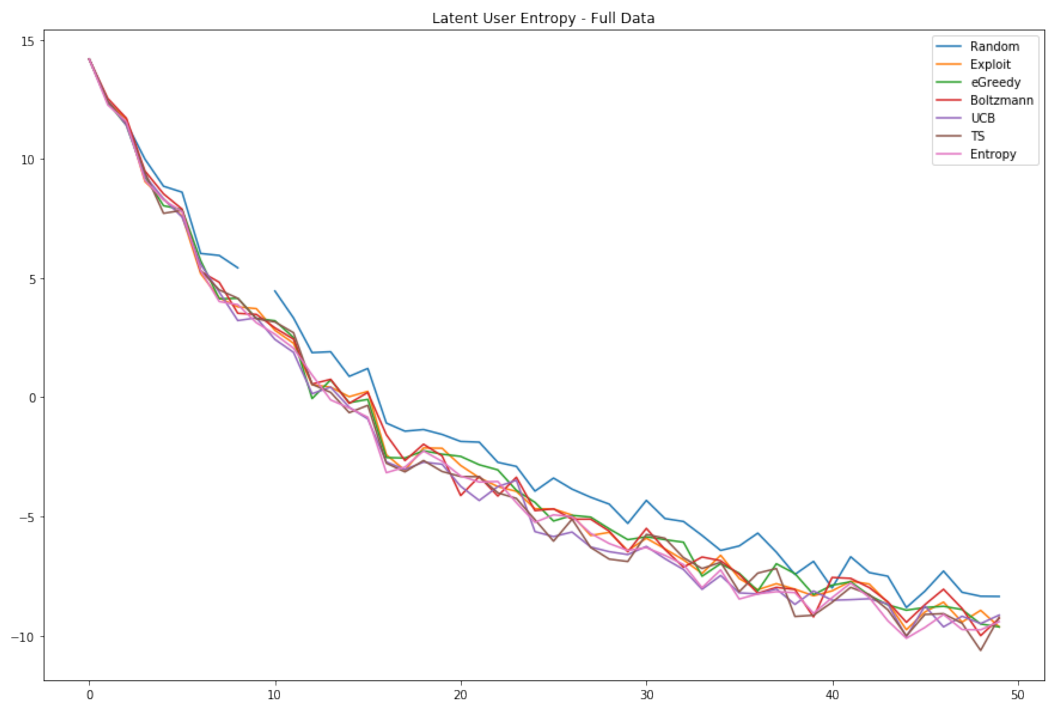

Figure 12: Average entropy of the user latent variables over 50 time steps (The training ran into numerical error at action step 10 for random selection)

References

[1] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, May 2002.

[2] Olivier Cappé, Aurélien Garivier, Odalric-Ambrym Maillard, Rémi Munos, and Gilles Stoltz. Kullback-Leibler Upper Confidence Bounds for Optimal Sequential Allocation. *Annals of Statistics*, 41(3):1516–1541, 2013. Accepted, to appear in Annals of Statistics.

[3] Gintare Karolina Dziugaite and Daniel M. Roy. Neural network matrix factorization. *CoRR*, abs/1511.06443, 2015.

[4] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, December 2015.

[5] Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic variational inference. *J. Mach. Learn. Res.*, 14(1):1303–1347, May 2013.

[6] Emilie Kaufmann, Olivier Cappe, and Aurelien Garivier. On bayesian upper confidence bounds for bandit problems. In Neil D. Lawrence and Mark Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 592–600, La Palma, Canary Islands, 21–23 Apr 2012. PMLR.

[7] Volodymyr Kuleshov and Doina Precup. Algorithms for multi-armed bandit problems. *CoRR*, abs/1402.6028, 2014.

[8] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 661–670, New York, NY, USA, 2010. ACM.

[9] Jérémie Mary, Romaric Gaudel, and Philippe Preux. Bandits warm-up cold recommender systems. *CoRR*, abs/1407.2806, 2014.

[10] Andriy Mnih and Ruslan R Salakhutdinov. Probabilistic matrix factorization. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1257–1264. Curran Associates, Inc., 2008.

[11] Jake Stolee and Neill Patterson. Matrix factorization with neural networks and stochastic variational inference, 2016.

[12] Xiaoxue Zhao, Weinan Zhang, and Jun Wang. Interactive collaborative filtering. In *Proceedings of the 22nd ACM international conference on Conference on information &#38; knowledge management*, CIKM '13, pages 1411–1420, New York, NY, USA, 2013. ACM.