A comparative analysis of subreddit recommenders for Reddit

Jay Baxter
Massachusetts Institute of Technology
jbaxter@mit.edu

Abstract

Reddit has become a very popular social news website, but even though it now has over 10 million users, there is still no good way to discover "subreddits" - online communities based on specific discussion topics. This paper approaches the subreddit discovery problem by using collaborative filtering to recommend subreddits to Reddit users based on their past voting history. Three different methods are considered, and are evaluated on three metrics: accuracy, coverage, and novelty. We find that each method has its strengths and weaknesses, and that there is no clear-cut best method for this unusual dataset.

1 Introduction

1.1 What is Reddit?

Reddit is a popular social news website where any registered user can submit a link or text post. All users can then vote any submission up or down, signaling whether they like or dislike the submission. The total of all votes for a submission (an upvote is +1 and a downvote is -1) is used to determine how the submission is ranked (after accounting for how old the post is) on Reddit's front page and other pages. For the rest of the paper, I will use the words "link", "post", and "submission" interchangeably.

1.2 Subreddits

Reddit is divided into communities called "subreddits" based on areas of interest (e.g. programming, world news, gaming, atheism, or movies), and every submission must be submitted to one of these subreddit communities. Users can pick which subreddits they subscribe to, based on their own interests, but users are automatically subscribed to a default set of 20.

1.3 Why a recommender would be helpful

Since there are over 67,000 subreddits and over 10 million active users, finding a subreddit that matches your interests is not an easy problem. There are many sites that allow users to search for and browse subreddits, but there is no recommender yet, even though Reddit expressed a desire to have one two years ago. There are two types of recommenders you could make for Reddit: a submission recommender that would recommend individual posts that you are likely to like, and a subreddit recommender that recommends entire areas of interest to you. A number of people have made submission recommenders, but none that work well enough, and surprisingly, nobody has made a subreddit recommender (at least publicly). This paper focuses on the novel problem of recommending subreddits.

Subreddit discovery is a challenging problem for many users. Currently, the only ways to discover subreddits are to search, browse by popularity, browse randomly, or use a third party website like metareddit.com, subredditfinder.com, and yasiv.com that attempt to solve the subreddit discovery problem by using tags and user-defined lists of "subreddits similar to this one". However, the problem a subreddit recommender is trying to solve is fundamentally different: instead of just recommending more similar content, the system aims to recommend content that you will like, that could potentially be, and ideally will be, quite different from the content you already have seen.

2 Data

2.1 Data collection and format

Reddit allows users to check a box in their profile that gives Reddit permission to use their data. As of April 2, 2012, when this dataset was collected, 17,261 users had agreed to share their data publicly. In total, the dataset consists of all 5,260,381 votes those 17,261 users have made on 2,337,323 submissions that span 12,079 subreddits. For each vote, we have the user ID who made the vote, the submission ID of the submission he/she voted on, the subreddit name of the submission, and whether the vote was an upvote or downvote. All of the data is anonymized except for the subreddit names. Unfortunately, there is no time or content (what words the submissions contained) data available in this dataset, so all recommendations will be based on "collaborative filtering": giving recommendations (filtering) by collecting preferences or taste information from many users (collaborating).

2.2 Dealing with downvotes

When a user upvotes a post, it is an indication that that user liked the post, and would have liked that post to be recommended to him. If a user downvotes a post, it would be intuitive for that to mean that the user does not like the post. However, previous work on submission-level recommendations has shown that users tend to downvote submissions that they found interesting enough to read, even though they disagreed with some part of it enough to downvote it. I also found that the results got worse when I included downvotes in my dataset, so in this paper, all downvotes will be ignored when making recommendations. If we ignore downvotes, the dataset we are left with reduces to 3,944,301 upvotes: 75% of the total number of votes. For the rest of the paper, when I refer to "votes", I am referring to only upvotes.

2.3 Data statistics and sparsity

The dataset is very sparse: there is an average of 228.5 upvotes per user over 2,337,323 different submissions and 12,079 subreddits, and 326.5 upvotes per subreddit.

The 20 default subreddits contain 48% (1862415 out of 3894148) of the total votes. There is a strong overlap between those subreddits and the 20 most popular subreddits, which contain 65% (2514179 out of 3894148) of the total votes.

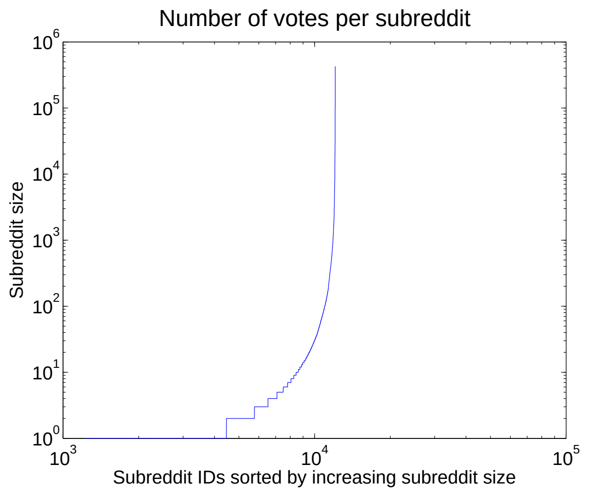

Figure 1: A log-log plot demonstrating the long tail of subreddit popularity. The horizontal axis shows the 12,079 subreddits sorted by increasing size, and the vertical axis represents the size of the given subreddit by number of total votes. (The plotted line becomes indistinguishable from the horizontal axis.)

There is a long tail of subreddit popularity. There are a few very popular subreddits and many many other unpopular subreddits, as is demonstrated in Figure 1.

It is important that the methods used are able to deal with this sparsity. However, in some cases, there is so little data that it doesn't even make sense. For example, if a subreddit has less than 10 different users who have ever voted on it, it is very hard to get a picture of what kind of user likes that subreddit. In this paper, the cold-start problem is ignored, and we remove subreddits with less than 10 users, leaving 1,876 out of the 12,079. We also ignore users that have less than 10 votes on non-default subreddits since this paper is focused on how to recommend non-default subreddits, which leaves 7,363 users out of the original 17,261. While that is only 16% of the subreddits and 43% of the users, we still have 95% of the original votes: 3690933 out of 3894148.

3 Evaluation

There are many, many ways to evaluate recommender systems in the literature, and no consensus about which is best. However, the evaluation method used obviously has a major impact on what type of algorithms do best. The most important thing to evaluate is "accuracy": does the user actually like the recommendations? Other considerations are novelty (would the user have been able to find this item on her own), coverage (what proportion of total items does the system ever recommend?), and learning rate (how many items does the user need to rate to start getting good recommendations?). In this paper, the focus is on accuracy, novelty, and coverage.

Since nobody has ever used this dataset for subreddit recommendations before, a huge portion of my time was spent defining the problem, choosing proper training and testing splits, and choosing proper evaluation methods.

To evaluate recommendations, we must keep in mind that our goal is to recommend a subreddits that the user will upvote posts on.

3.1 Training and testing splits

At the subreddit level, the recommendation problem is fundamentally a content discovery problem. I decided that the training and testing splits should resemble the real user experience as closely as possible. At the time of data collection, when a new user joined Reddit, they were automatically subscribed to a default set of 20 subreddits (since then, a few more subreddits have been added to the default set). In this sense, all users have all seen the same default set of subreddits. Then, as a user spends more and more time browsing Reddit, she slowly discovers more and more subreddits outside the default set. Therefore, since all users have all seen the default set, we will never recommend those subreddits, and will always use those as training data. For a given user, there will be two testing scenarios: one where the training data is only the data from the default subreddits, and all votes on non-default subreddits are testing data to simulate new users, and another scenario where we randomly select a portion of non-default subreddits to additionally be included in the training set to simulate more experienced users. In all results shown in this paper, we perform 10-fold cross-validation over the users. On subreddits, we always train on the default, and then either test on all the rest, or perform 2 or 10-fold cross validation over the non-default subreddits. Confirming our intuition that we can give better recommendations with more user data, we find that every method gets a higher accuracy score when we train on some of the non-default subreddits in addition to the defaults.

3.2 Accuracy Metric

For evaluating subreddit recommendations, I considering many different types, and decided that a utility metric would be most accurate. First, we must define a user's rating of a subreddit, since there is no obvious way. We will define the rating as the number of times user i has upvoted a post in subreddit j, $v_{i,j}$, divided by his total number of upvotes.

$$r_{i,j} = \frac{v_{i,j}}{\sum_j v_{i,j}} \tag{1}$$

We define the utility of a recommendation to the user to be the user's rating of the recommended subreddit times the likelihood that the user will see the recommendation. Common likelihood functions are exponential decay with half-life α and the step function that only consider the top N recommendations. [1] Let R_i be the expected utility for user i over subreddits j.

$$A_i^{step} = \sum_{j=1}^{N} r_{i,j} \tag{2}$$

To more appropriately model a real use case for this recommender system, I chose to use step function likelihood, because a user will most likely view all the recommendations on the page, and be unlikely to look at the next page.

The overall score for a dataset over all users, A, is shown below, where A_i^{max} is the utility achieved from giving perfect recommendations for user i.

$$A = \frac{\sum_i A_i}{\sum_i A_i^{max}} \tag{3}$$

This score can be interpreted as the percentage of all of the user's held-out votes that are contained within the subreddits we recommended.

3.3 Coverage

Coverage is one way to determine if a recommender recommends the same popular items to everyone instead of coming up with a reasonable degree of personalization. Coverage is defined as the percentage of all recommendable items that the system ever recommends to any user (as one of the top N recommendations).

3.4 Novelty and Serendipity

Novelty and serendipity are two crucially important aspects of a recommender system. Novelty measures how likely it is that the user has never seen the recommended item before, and serendipity measures how likely it is that the item is both novel and hard for the user to find. If an item is serendipitous, it is therefore also novel. Novelty and serendipity are important because the entire point of a recommender is to show the user content he hasn't already seen before. Unfortunately, novelty and serendipity are very hard to quantitatively measure without performing a study with live users and observing their actions to recommendations.

The baseline method described in the next method always returns the most popular subreddit. We can get an approximate idea of novelty by finding the difference between these most popular results and the results of a recommender. We will measure novelty of a set of recommendations as the sum of the inverse popularities of all subreddits (where popularity means the number of votes in the subreddit), where j ranges over the N recommended subreddits, and where i denotes the user id.

$$NOV = \sum_{j=1}^{N} \frac{1}{\sum_i r_{ij}} \tag{4}$$

With N=20 recommendations, returning the most popular items gives 0.448 and returning the least popular items gives 11440. Novelty scales with N: if N increases, so does novelty. Of course, returning the least popular items is not useful, so this metric must be considered as something to trade off with accuracy.

We are unable to measure serendipity with this dataset, but as future work, more data could be collected to determine which subreddits are easily discoverable for which users.

4 Baseline Method

The simple recommendation algorithm used as a baseline makes the same predictions for all users. Given that constraint, the baseline method will maximize its score by always recommending the most popular subreddits from the test set based on the training users' preferences.

The above tables give us an intuition for the nature of the dataset and the typical values accuracy, coverage,

N	1	5	10	20	50
Accuracy	0.070	0.246	0.336	0.448	0.611
Coverage	0.0005	0.003	0.005	0.010	0.025
Novelty	0.005	0.039	0.139	0.448	2.53

Figure 2: Train on default subreddits; test on rest

N	1	5	10	20	50
Accuracy	0.116	0.325	0.439	0.567	0.728
Coverage	0.0001	0.005	0.010	0.020	0.049
Novelty	0.007	0.07	0.24	0.83	5.11

Figure 3: 2-fold cross-validation over subreddits

and novelty take on for each different cross-validation setup. Here, we see that coverage and accuracy increase as N increases, but accuracy does not.

Since these variables have such predictable relationships with N, for the sake of brevity, I only display results with $N = 20$ for the rest of the paper, since that is the most likely use case. However, the results do not qualitatively change as N changes to 10 or 50, for example.

5 Nearest Neighbors

The first real recommendation method we will try is nearest neighbor, or k nearest neighbors (kNN). We must first define some notion of distance between users. Then, when we are asked to give a recommendation for a user, we compute the distance between that user and all other users. We take the average of the k most similar users' ratings to predict the ratings for the query user, and then return the N items with the highest predicted rating.

This approach is called a memory-based approach, as opposed to model-based, because kNN never builds a model - it looks all all the data for every query. My implementation computes a matrix of user similarities in order to efficiently compute similarities using vectorized Matlab code, but on a larger dataset, this algorithm does not scale. The time it saves not building a model is quickly lost when computing queries in $O(|U| \cdot |V|)$ time, although there are many faster approximation methods. In a live implementation, this user matrix would need to be recomputed every time a new item was added, making it impractical unless approximations are used.

N	1	5	10	20	50
Accuracy	0.255	0.560	0.691	0.803	0.927
Coverage	0.005	0.025	0.049	0.098	0.246
Novelty	0.02	0.30	1.150	4.86	39.17

Figure 4: 10-fold cross-validation over subreddits

5.1 Subreddit-based User Similarity

Looking at similarity at the subreddit level, as opposed to looking at similarities between votes on individual posts, is a way of dealing with data sparsity. Since there are so many different posts, and the probability of two users both upvoting the same specific post is so low, we can aggregate posts together by subreddit and compute how similar users are based on how much they seem to like each subreddit as a whole instead of each individual post.

Cosine similarity is one similarity metric that is commonly used to compare users and items in recommender systems, and it is used here because it is natural given the domain, and is easy to compute.

$$sim(u_1, u_2) = \cos(\vec{u}_1, \vec{u}_2)) = \frac{\vec{u}_1 \cdot \vec{u}_2}{\|\vec{u}_1\| \|\vec{u}_2\|}$$

Computability is a real concern, since we are required to compute the distance between all pairs of users, and even cosine similarity can be too slow for massive datasets like Amazon. This rules out many more complex similarity functions.

k	1	2	5	10	50
Accuracy	0.324	0.325	0.330	0.325	0.328
Coverage	0.249	0.255	0.249	0.252	0.250
Novelty	5.29	5.56	5.35	5.44	5.63

Figure 5: kNN results with cosine similarity distance. Trained on default subreddits; tested on rest

k	1	2	5	10	50
Accuracy	0.438	0.419	0.433	0.428	0.434
Coverage	0.452	0.447	0.453	0.451	0.250
Novelty	9.69	10.05	9.90	10.0	5.63

Figure 6: kNN results with cosine similarity distance. 2-fold cross-validation over subreddits

5.2 Weighting similarities based on subreddit popularity

As a way to give more weight to subreddits that are smaller (or larger), I computed subreddit popularities. Let A be the vote matrix where A_{ij} is the number of times user i has upvoted subreddit j. First, the total number of votes V per subreddit can be found by summing out the users: $V_j = \sum_i A_{ij}$. I then normalize V and then compute a popularity weight $W_j = -\log(V_j)$. The logarithm is there to ensure that the numbers stay reasonable: without it, the results become very erratic.

Then, I compute user similarity by looking at the subreddits that both users have voted on, normalizing their votes across those subreddits, then computing the elementwise product of those vectors, and then let the similarity be the dot product of that vector with the subreddit popularity weights, to take into account how it's more informative that two users both like the same unpopular subreddit than if they both like the same popular subreddit.

k	1	2	5	10	50
Accuracy	0.329	0.329	0.329	0.329	0.328
Coverage	0.011	0.011	0.011	0.011	0.011
Novelty	13.68	13.75	14.30	13.68	13.71

Figure 7: kNN with unpopular-weighted cosine similarity, trained on default subreddits; tested on rest.

Surprisingly, this method gets similar accuracy scores as nearest neighbors using unweighted cosine similarity, in addition to getting better novelty scores. Counterintuitively, coverage scores decreased.

I also took the inverse of the weightings, so that similarity is more heavily affected by larger subreddits.

k	1	2	5	10	50
Accuracy	0.497	0.465	0.463	0.469	0.467
Coverage	0.022	0.022	0.022	0.022	0.022
Novelty	7.57	7.74	8.31	7.82	7.55

Figure 8: kNN results with popular-weighted cosine similarity, trained on default subreddits; tested on rest

This method works surprisingly well: it has the highest accuracy of the paper, while still achieving good novelty. The surprisingly good results of this method may be a result of how the training and test sets were constructed, but either way, the success of this method is one of the most unintuitive results of this paper.

6 SVD

Singular Value Decomposition (SVD) is a way to find low-rank approximations that minimize the sum squared distance the the ratings matrix R, where each rating R_{ij} is the number of times user i has upvoted a post in subreddit j. SVD factors R into USV^T, where U can be thought of as the user matrix, V can be thought of as the subreddit matrix, and S is the singular value matrix. To obtain a low-rank approximation of the data, we limit the dimensions by limiting the dimensionality of S. With the dimension limited, SVD computes the approximation matrix $\hat{R} = USV^T$ that minimizes the sum-squared distance of the observed entries in R.

In contrast to nearest neighbors, SVD is a model-based method. Consequently, it requires more up-front model-building time, but can answer recommendation queries much faster than kNN.

SVD has two parameters that I set: the dimensionality and the way we initialize the held-out ratings. To optimally pick these parameters, I tested many values with cross-validation.

In some sense, you need to fill the blank ratings in. I tried three methods: filling them all with zeros, fill them all uniformly, and filling them all based on subreddit popularity. However, the differences between these three filling methods were extremely negligible - there was no difference in the resulting recommendations given. The dimensionality, however, was very important.

Novelty varies highly from run to run with SVD, but definitely decreases as the dimensionality increases. The below table shows scores averaged over 5 separate runs of 10-fold cross-validation. Accuracy and coverage remain nearly constant across trials.

Dim	1	2	3	4	5
Accuracy	0.445	0.462	0.464	0.436	0.434
Coverage	0.020	0.022	0.022	0.030	0.033
Novelty	153.8	32.18	1.47	0.691	4.390

Figure 9: Unscaled SVD trained on default subreddits; tested on rest

Dim	1	2	3	4	5
Accuracy	0.528	0.573	0.576	0.563	0.560
Coverage	0.267	0.041	0.048	0.056	0.059
Novelty	68.98	7.42	1.06	1.16	2.28

Figure 10: Unscaled SVD with 2-fold cross-validation over subreddits

Dim	1	2	3	4	5
Accuracy	0.803	0.816	0.811	0.804	0.813
Coverage	0.126	0.173	0.224	0.237	0.263
Novelty	10.76	7.74	6.98	7.58	8.51

Figure 11: Unscaled SVD with 10-fold cross-validation over subreddits

After performing cross-validation, we found that the 2-dimension model and 3-dimension model get very similar accuracy. Additionally, we find that novelty is by far the highest with 1 dimension, and drastically decreases as dimensions are added. Coverage is fairly constant throughout. Depending on whether novelty or accuracy is more important for the situation, the rank 1 and 2 models are by far the best.

6.1 Scaling the data

Rank	1	2	3	4	5
Accuracy	0.438	0.459	0.422	0.430	0.434
Coverage	0.020	0.022	0.023	0.027	0 .031
Novelty	163.4	45.7	0.67	8.53	1.24

Figure 12: Scaled. Trained on default subreddits; tested on rest

Normalizing the vote data causes accuracy to go down, but causes novelty to go up. Again, coverage and accuracy are quite constant, but novelty has high variance. For example, with dimension 4 in the table above, one fluke run had a novelty of over 30, skewing the average.

7 Probabilistic Matrix Factorization

Probabilistic Matrix Factorizaiton (PMF) is a bayesian approach to matrix factorization that attempts to deal with very large, sparse datasets [3]. The authors provide a partial implementation of their code intended for the Netflix challenge, but I needed to modify the code to implement the missing pieces and adapt its parameters to fit the Reddit problem. To adapt their implementation, I closely followed [3], and adjusted the code so that it took Reddit data instead of Netflix data. This includes changing the average rating and removing the sigmoid function from the ratings outputs.

PMF can be viewed as a probabilistic extension to SVD, because if all ratings are observed and prior variances are infinite, then the objective function reduces to the SVD objective. As in SVD, our goal is to fit

$D \times N$ user matrix U and $D \times M$ subreddit matrix V that multiply to give the best matrix R under the loss function. We use a probabilistic linear model with Gaussian observation noise, where the conditional distribution on observed ratings is:

$$p(R|U, V, \sigma^2) = \prod_{i=1}^{N} \prod_{j=1}^{M} (\mathcal{N}(R_{ij}|U_i^T V_j, \sigma^2))^{I_{ij}} \tag{5}$$

where I_{ij} is equal to 1 if user i rated subreddit j. We also place zero-mean spherical Gaussian priors on user and movie feature vectors. The resulting graphical model is shown in Figure 13.

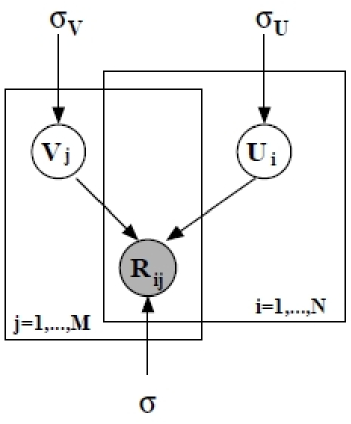

Figure 13: The bayesian network for PMF

[3] shows that given this setup, maximizing the log-posterior distribution over movie and user features with constant hyper parameters is equivalent to minimizing the sum of squared error objective function with quadratic regularization:

$$\begin{aligned} E =& \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{M} I_{ij} (R_{ij} - U_i^T V_j)^2 \\ &+ \frac{\lambda_U}{2} \sum_{i=1}^{N} \|U_i\|_{Fro}^2 + \frac{\lambda_V}{2} \sum_{j=1}^{M} \|V_j\|_{Fro}^2 \end{aligned}$$

where $\lambda_U = \sigma^2/\sigma_U^2$, $\lambda_V = \sigma^2/\sigma_V^2$, and $\| \cdot \|_{Fro}^2$ denotes the Frobenius norm. We can optimize this objective function by performing gradient descent on U and V.

I was very surprised by the results of PMF. I thought it would be a high-accuracy method like SVD, but I tried a very large set of possible parameters and never got accuracy close to the baseline method when training on the default subreddits and testing on the rest. Since I was forced to train this model using gradient descent, it's possible that there is a parameter setting that I missed, but my results are so consistent that I doubt PMF can do better on this dataset.

One peculiarity is that with λ ranging from 0 to 0.1, the best accuracy is achieved with $\lambda = 0$. Better novelties are achieved with higher regularization, which also makes sense: the more regularization, the less we overfit. Additionally, the results show that the initialization method does not have a noticeable impact on the final recommendations.

Instead of giving high accuracies, PMF gives acceptable accuracies that are roughly around 10%, which means that 2 out of any 20 results are relevant. However, PMF has by far the highest novelty out of any recommendation method. Without user testing, it is unclear what the preferred tradeoff between novelty and accuracy is, but PMF has exceedingly high novelty. PMF also gives quite good coverage compared to other methods, which increases its potential usefulness.

λ	0.1	0.01	0.001	0.0001	0
Acc.	0.002	0.078	0.105	0.127	0.126
Cov.	0.083	0.151	0.156	0.140	0.141
Nov.	924270	14975	51562	83488	41596

Figure 14: With $\epsilon = 50$ and default ratings initialized to zero. Trained on default subreddits; test on rest.

CV Folds	0	2	10
Accuracy	0.130	0.220	0.345
Coverage	0.158	0.274	0.783
Novelty	41596	170550	721630

Figure 15: With $\epsilon = 50$, $\lambda = 0$, and default ratings initialized to the average rating for each user. 0 folds of CV means that I trained on the default subreddits and tested on the rest.

There is also a fully bayesian version of PMF, Bayesian PMF (BPMF), that puts priors on all the parameters [2]. BPMF has been shown to get better results than PMF, especially for users with few votes. However, it must be trained with approximate inference, e.g. Gibbs Sampling, that takes days to converge on a dataset this size, even when initialized to the MAP solution found by PMF. Since PMF already takes hours to train, I must leave testing BPMF as work for future experiments.

8 Conclusion

Different methods are better depending on which evaluation metric is most important to us. We see that kNN with subreddit popularity weighting gives the highest accuracy, with SVD close behind. SVD also gets good novelty when 1 or 2 dimensions are used. PMF gives the best novelty and acceptable accuracy, and nearest neighbor gives the best coverage for k less than about 10 when using normal weightings. Multiple variations of these methods were tried as well: we found that scaling the data hurts SVD performance and weighting unpopular subreddits more heavily hurts kNN performance, both results that I did not expect. PMF's performance was surprising as well: the Reddit dataset has characteristics different enough from the Netflix challenge that algorithms that worked well on that task do not necessarily work well on this task, as has been shown empirically. In summary, many more methods should be tried as well, with a focus on kNN and SVD-like methods.

References

[1] J.L. Herlocker, J.A. Konstan, L.G. Terveen, and J.T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

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

[3] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. *Advances in neural information processing systems*, 20:1257–1264, 2008.