Movie Recommender System with Matrix Factorization Models

Xingyu Dong XDONG1@SWARTHMORE.EDU
Xinxin Li XLI5@SWARTHMORE.EDU
Luyuan Fan LFAN1@SWARTHMORE.EDU

Abstract

Matrix factorization models are one type of collaborative filtering recommender system that decomposes a large, sparse user-item interaction matrix by representing users and items in latent dimensions and predicting based on the relationships among latent factors. This paper presents implementations and performance analysis of two matrix factorization algorithms based on movie ratings from the MovieLens dataset. After performing hyperparameter tuning, our experiment results show that Probabilistic Matrix Factorization (PMF, commonly referred to as SVD under the context of recommender systems) achieves its best accuracy with a higher number of latent factors, a medium learning rate, and a small regularization factor. For SVD++, an improved version of SVD by taking account of implicit feedback, yields the same parameter combination and a slightly better accuracy result. Analyzing their performances, we have inferred that our dataset has strong explicit feedback and less informative implicit feedback, and the explanatory powers of the two algorithms are similar in this case. Later in this paper, limitations to our models as well as the ethical and social implications of recommender systems are discussed in detail.

1. Introduction

Recommender systems are helpful tools that seek to assist individuals in navigating through a huge amount of online information in a more personalized manner. These systems are designed to improve user experience by filtering content tailored to individual preferences. With the increasing prevalence of online services such as online advertising, streaming services, and social networking, all characterized by their extensive databases, the importance of recommender systems has become more pronounced. They are not only instrumental for content providers in discerning user preferences—thereby enhancing engagement and profitability—but also for users, who benefit from more efficient and enjoyable services.

In general terms, recommender systems deal with the interaction between two key elements — users and items, where each user assigns a rating or indicates a preference level for a particular item, such as movies or books. These data build up the foundation of recommender systems to provide suggestions tailored to their tastes, i.e. obtaining the utility of an item to a certain user, in the future. There are two primary approaches in their designs:

- *Content-based Filtering*. This approach identifies shared characteristics of items that have received positive ratings or reviews from a user, and then makes recommendations on new items that share these features with this user, assuming users would like something in the future that shares attributes with something they liked before (**?**).
- *Collaborative Filtering*. This approach, beyond ratings made by a certain user, relies on identifying users who share preferences alike to offer item recommendations. Here, two types of feedback are taken into account: *explicit feedback* and *implicit feedback.* This approach can be further broken down into two classes: *neighborhood-based*, which directly uses user-item ratings in its predictions, and *latent factor models*, which makes use of a predictive model that is later used in making its predictions (Koren et al., 2009).

In content-based filtering, traditional machine learning algorithms like Decision Trees are well-suited, as the process essentially boils down to a classification problem. On the other hand, collaborative filtering requires more dedicated algorithms that can calculate the similarities among users and items. To achieve such a goal, they often consider both explicit and implicit feedback. *Explicit feedback* consists of direct user input, such as ratings or reviews, while *implicit feedback* is inferred from user actions, including the duration of time spent on a video or even mouse movement

CPSC 66 Machine Learning Proceedings, Swarthmore College, Fall 2023.

on a webpage. In so doing, collaborative filtering solves the problem that explicit feedback with good quantity and quality is not always available, capturing more complex user-item relationships that content-based systems could potentially overlook.

Yet this attempt to capture more complex patterns in user behavior data is two-fold, as it introduces the curse of dimensionality when dealing with higher-dimensional data space. Beyond this, some other issues are commonly faced by recommender systems such as *Cold Start, Data Sparsity,* and *Imbalanced Dataset*. Techniques like matrix factorization algorithms, one type of model-based algorithms in collaborative filtering, are proposed to deal with some of the problems while leveraging both explicit and implicit feedback.

Previously, several works have proposed possible matrix factorization algorithms. Probabilistic Matrix Factorization (PMF) is one of the approaches that could mitigate the data sparsity problem while being scalable and computationally affordable (Salakhutdinov & Mnih, 2008). Building upon PMF, models such as SVD++ and Time-aware Factor Models add more complexity to it in order to capture implicit feedback and temporal effects (Ricci et al., 2010). In general, matrix Factorization algorithms are able to capture both explicit and (many types of) implicit feedback, and is well-suited for dealing with sparse dataset.

This paper presents our implementation of two matrix factorization algorithms, PMF and SVD++, and focuses on evaluating their performance in predicting movie ratings based on the MovieLens dataset. Section 2 explains the mechanisms and implementation strategies for two selected algorithms. Subsequently, Section 3 introduces a series of experiments that assess the efficacy of these algorithms, followed by a discussion of the results obtained. Section 4 explores the broader ethical considerations associated with recommender systems. Lastly, section 5 concludes our experiment results and highlights the potential enhancements for prediction accuracy.

2. Methods

The matrix factorization algorithms are *latent factor models* that attempt to represent users and items using a series of latent factors inferred from the data patterns. In other words, given U number of users, I number of items, and a user-item matrix R where each data point represents each user-item interaction, we want to map users and items to a latent dimension of k by creating user factor matrix of size $U \times k$ and item factor matrix of size $I \times k$. Therefore, each item i is represented as a vector $q_i \in \mathbb{R}^k$, where each element of q_i represents to what extent item i contains the corresponding latent factor; similarly, each user u is represented as a vector $p_u \in \mathbb{R}^k$, where each element of p_u represents the amount of interest user u has towards the corresponding latent factor (Koren et al., 2009). The dot product between q_i and p_u will therefore capture the opinion (or commonly referred to as ratings r) user u has towards item i (i.e. the prediction of ratings of user u on item i):

$$\hat{r}_{ui} = q_i^T p_u$$

Thus, the training goal of matrix factorization algorithms is to update q_i and p_u such that the error of prediction can be minimized.

Speaking from a practical context of predicting movie ratings, a user u will be whoever registered the rating website and has left any ratings. An item i will be a movie on the website that can be rated. Before training the model, an arbitrary k needs to be determined by inspecting the dataset.

Besides the method above, there are other ways to compute latent factors. *Singular Value Decomposition* (SVD) has been a useful tool in many scenarios to reduce the dimensionality and find the latent representations of the dataset. However, conventional SVD does not perform well in the recommender system context: while SVD requires a full matrix with no missing data, the user-item matrix that a recommender system relies on very often has empty entries (Salakhutdinov & Mnih, 2008). This is not difficult to imagine - it is very rare for a user to rate every single movie on the website. Early works have employed imputation to amend the missing data, but the increasing number of computations still prevents the model from being practically useful (Koren et al., 2009). Besides, imputation also faces the possibility of severely corrupting the dataset, if new data is not carefully determined. To solve the drawbacks of SVD while leveraging the scalability of matrix factorization algorithms, several SVD-based algorithms are proposed. Two specific algorithms will be discussed in this section: *Probabilistic Matrix Factorization* (PMF, or commonly simplified just as SVD under the context of recommender systems) and *Probabilistic Matrix Factorization Plus* (or SVD++ under the context of recommender system).

2.1. SVD

Probabilistic Matrix Factorization, or SVD, is the most straightforward version of matrix factorization models. Besides computing $q_i^T p_u$, the prediction also incorporates a few other bias terms:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u \tag{1}$$

where μ is a constant bias, which is usually the mean of all existing ratings, b_i is a bias term specific to items, and b_u is a bias term specific to users (Ricci et al., 2010). Our goal is to adjust b_i, b_u, q_i, and p_u to minimize the error between

r_{ui} and $\hat{r}_{ui}$, which we refer to as e_{ui}. Specifically, we want to minimize the regularized squared error:

$$\sum_{(u,i)\in\kappa} (r_{ui}-\mu-b_i-b_u-q_i^T p_u)^2+\lambda(\|p_u\|^2+\|q_i\|^2+b_u^2+b_i^2)$$

where κ is a training set that contains all (u, i) pairs, and λ is a regularization hyperparameter applied to reduce the effect of overfitting. To perform this minimization, we use *stochastic gradient descent* (SGD). The steps of training are as follows:

For a given data point r_{ui}, we make a prediction point $\hat{r}_{ui}$. Then we calculate e_{ui} to be $e_{ui} = r_{ui} - \hat{r}_{ui}$. After getting e_{ui}, we use it to update b, q_i, and p_u by performing SGD:

$$\begin{aligned} b_u &\leftarrow b_u + \gamma \cdot (e_{ui} - \lambda \cdot b_u) \\ b_i &\leftarrow b_i + \gamma \cdot (e_{ui} - \lambda \cdot b_i) \\ q_i &\leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i) \\ p_u &\leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u) \end{aligned}$$

where γ is the learning rate hyperparameter and λ is the regularization rate hyperparameter. The algorithm repeats the process until it processes all data points for a certain amount of epochs. The pseudocode is as below:

```
def fit(train_matrix):
   initialize latent factors randomly using
       Gaussian distribution
   initialize bias terms based on
       train_matrix

   for epoch in epochs:
      for user in users:
         for item in items:
            // skip missing data
            if train_matrix[user, item] >
                0:
               prediction = predict(user,
                   item)
               error = train_matrix[user,
                   item] - prediction

               // perform SGD
               update each factor for
                   user_factor[user]
               update each factor for
                   item_factor[item]
               update bias for user in
                   user_bias[user]
               update bias for item in
                   item_bias[item]
```

After the fitting process, we will have b_i, b_u, q_i, and p_u that can be plugged into equation (1) and make predictions.

2.2. SVD++

SVD++ is an improvement on top of SVD by adding implicit feedback when making predictions. In the context of movie rating prediction, the implicit feedback can be thought of as the implicit preference of user u of rating movie A instead of movie B. In other words, users can be assumed to never rate randomly but deliberately, therefore making their choices of rating indicative of their preferences (Ricci et al., 2010). To incorporate implicit feedback, a new term to the prediction is added:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u + |R(u)|^{-\frac{1}{2}} \sum_{j\in R(u)} y_j \quad (2)$$

where $R(u)$ is all the ratings provided by user u, and the sum of y_j represents the implicit feedback to all the corresponding items rated (Ricci et al., 2010). Our goal is to train b_i, b_u, q_i, p_u, and y_j to minimize the error e_{ui}. Specifically, we want to minimize the regularized squared error:

$$\sum_{(u,i)\in\kappa} (r_{ui} - \mu - b_i - b_u - q_i^T p_u - |R(u)|^{-\frac{1}{2}} \sum_{j\in R(u)} y_j)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2 + \sum_{j\in R(u)} \|y_j\|^2)$$

Our SGD now becomes the following:

$$\begin{aligned} b_u &\leftarrow b_u + \gamma \cdot (e_{ui} - \lambda \cdot b_u) \\ b_i &\leftarrow b_i + \gamma \cdot (e_{ui} - \lambda \cdot b_i) \\ q_i &\leftarrow q_i + \gamma \cdot (e_{ui} \cdot (p_u + |R(u)|^{-\frac{1}{2}} \sum_{j\in R(u)} y_j) - \lambda \cdot q_i) \\ p_u &\leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u) \\ \forall j \in R(u) &: y_j \leftarrow y_j + \gamma \cdot (e_{ui} \cdot |R(u)|^{-\frac{1}{2}} \cdot q_i - \lambda \cdot y_j) \end{aligned}$$

The algorithm repeats the process until it processes all data points for a certain amount of epochs. The pseudocode is as below:

```
def fit(train_matrix):
   initialize latent factors randomly using
       Gaussian distribution
   initialize bias terms based on
       train_matrix

   for epoch in epochs:
      for user in users:
         // calculate implicit feedback for
             user i
         find sqrt_Ru using train_matrix[
             main]
         find sum_yj using implicit_factor

         for item in items:
            // skip missing data
            if train_matrix[user, item] >
                0:
               prediction = predict(user,
                   item)
               error = train_matrix[user,
                   item] - prediction
```

```
// perform SGD
update each factor for
    item_factor[item]
update each factor for
    user_factor[user]
update bias for user in
    user_bias[user]
update bias for item in
    item_bias[item]

for item in train[user]:
   update each factor for
       implicit_factor[item]
```

After the fitting process, we will have b_i, b_u, q_i, p_u, and y_j that can be plugged into equation (2) and make predictions.

3. Experiments and Results

3.1. Data

We used the MovieLens dataset for testing, hyperparameter tuning, and performance analysis of our model. MovieLens dataset is a collection of movie ratings, and is maintained by the GroupLens Research Project. The GroupLens Research Project provides multiple versions of the dataset with different dataset sizes and various features such as user ID, movie ID, ratings, and movie tags (GroupLens Research, 1998).

SciKit Surprise's dataset module also provides preprocessed MovieLens 100k and MovieLens 1M datasets, and are organized in the form of `userID-movieID-rating` combinations, which provides simplicity in usage. `userID` and `movieID` are represented as integers, and `rating` is a float that can take the values of 1.0, 2.0, 3.0, 4.0, and 5.0. Due to the limitation in computing power, we used the MovieLens 100k dataset for this project, which contains 100k userID-movieID-rating combinations and is sufficiently large for generating reliable results.

3.2. Performance Metric

We use root mean squared error (RMSE) as the metric for assessing model performance. RMSE is computed from

$$\sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}{N}}$$

where N is the sample size, x_i is the true value for the i^{th} element, and $\hat{x}_i$ is the predicted value for the i^{th} element (Hyndman & Koehler, 2006).

RMSE has the same unit as original data, and can be interpreted as the average Euclidean distance between the true and predicted values. RMSE is very commonly used for the performance of recommender systems due to its consistency in units with the data and simplicity in interpretation (Koren, 2008). For the MovieLens 100k dataset, RMSE can be interested as to how far are the predicted values away from the true values on average. Therefore a model with a lower RMSE is more ideal.

3.3. Experimental Methodology

We structured our experimental methodology for both SVD and SVD++ in the following order:

1. Implement model and train on a 75-25 train-test split.
2. Test for predicting a single point. We chose the pair of user 196 and item 242, with a true rating of 3.0. This point could potentially be in either the training set or the testset, so the result is purely an indicator of the completeness of the algorithm, not an assessment of the performance.
3. Test on the entire testset and populate the RMSE. This step ensures that our model and testing functions are capable of a full round of fitting, predicting, and testing.
4. Conduct hyperparameter tuning.

Both our implementation of SVD and SVD++ involves 3 hyperparameters: number of latent factors, learning rate, and regularization factor. We want to train SVD and SVD++ such that we can analyze and compare their performances and decide their best hyperparameters. Our initial hypotheses are that:

- In general, SVD++ performs better than SVD, since it takes account of implicit feedback when making predictions.
- Higher number of latent factors will give a better performance, since it has more capacity to capture the relations between users and items.

3.4. Hyperparameter Tuning

For tuning, we have decided to use the following values for the 3 hyperparameters:

- Number of latent factors: 5, 10, 15
- Learning rate: 0.01, 0.03, 0.05
- Regularization factor: 0.1, 0.3, 0.5

We performed k-fold cross-validation on the above parameters, with $k = 5$. The parameters are arranged through

grid search, where each parameter value would be matched with all other values once for testing. We aim to find the best combination of these three parameters to achieve the highest accuracy.

3.4.1. SVD Hyperparameter Tuning

For SVD, the smallest RMSE occurs with

$$\begin{aligned} \texttt{num\_factors} &= 15 \\ \texttt{learning\_rate} &= 0.03 \\ \texttt{reg} &= 0.1 \end{aligned}$$

with a value of 0.9279 on 5 folds. `num_factors` represents the number of latent factors, `learning_rate` represents the learning rate, and `reg` represents the regularization rate.

We plotted heatmaps with num_factors - learning_rate (Figure 1), num_factors - reg (Figure 2), and reg - learning_rate (Figure 3) combinations.

From Figure 1, the RMSE gets smaller as the learning rate decreases from 0.05 to 0.01, and from Figure 2, the RMSE gets smaller as the regulation term decreases from 0.5 to 0.1. In both cases, RMSE gets smaller with a larger number of factors, but the number of factors does not affect RMSE as much as the learning rate and regulation rate. However, it is interesting to notice in Figure 3 that the smallest RMSE actually occurs with a learning rate of 0.03, which is not the smallest learning rate tested. This could be caused by some interactions between regulation terms and learning rates, and worth investing more in the future.

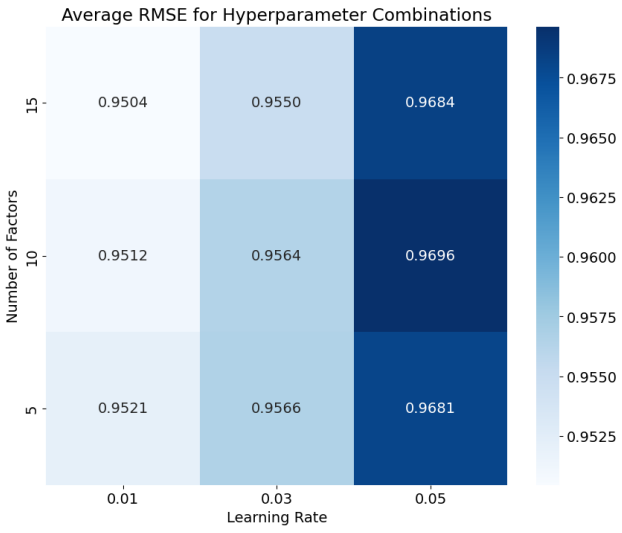

Figure 1. num_factors - learning_rate RMSE Heatmap for SVD

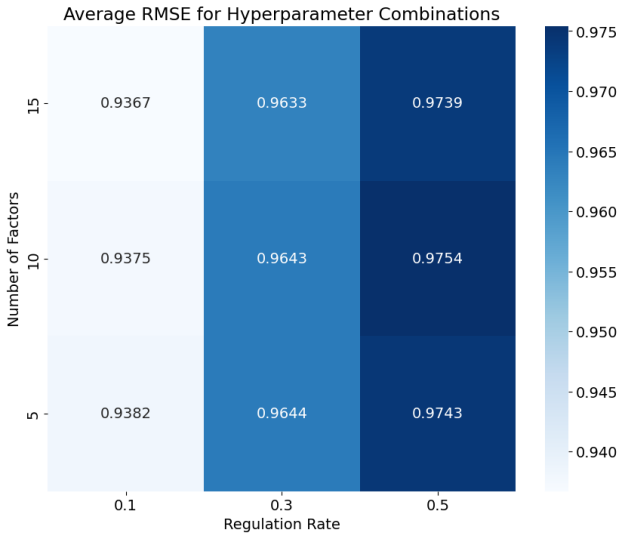

Figure 2. num_factors - reg RMSE Heatmap for SVD

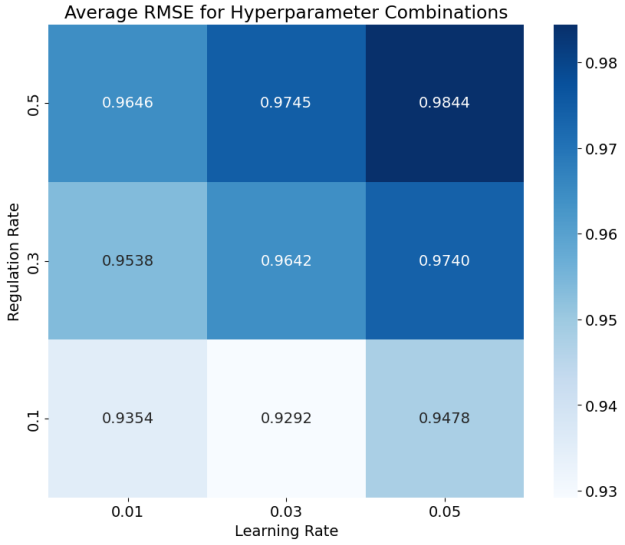

Figure 3. reg - learning_rate RMSE Heatmap for SVD

3.4.2. SVD++ Hyperparameter Tuning

For SVD, the smallest RMSE occurs with

$$\begin{aligned} \texttt{num\_factors} &= 15 \\ \texttt{learning\_rate} &= 0.03 \\ \texttt{reg} &= 0.1 \end{aligned}$$

with a value of 0.9290 on 5 folds.

We also plotted heatmaps with num_factors - learning_rate, num_factors - reg, and reg - learning_rate combinations, which can be found in Appendix 1. The hyperparameters choice for SVD++ is the same as for SVD, and the heatmaps also look similar.

3.4.3. Comparison Of SVD and SVD++

We observe a similar pattern in SVD++ where having a high number of latent factors, a low regularization factor, and a medium level of learning rate gives the best performance. The increasing accuracy of having more latent factors matches our hypothesis. It is interesting to see that having a medium learning rate gives the best performance, which may indicate that 0.01 and 0.05 descent too slow/-fast that we cannot find a good local minimum with them under the limited epochs.

We conducted a Wilcoxon two-sided hypothesis test on the RMSE values from the grid search in hyperparameter tuning and got p-value = 0.6963. Using p = 0.05 as the benchmark, we fail to reject the null hypothesis that SVD and SVD++ results do not have a statistically significant difference.

The similar results between SVD and SVD++ show that implicit feedback for our dataset does not significantly increase the prediction accuracy. This may be caused by not having very informative implicit feedback (e.g. the user-item interaction matrix is too sparse to provide meaningful implications), or the explicit feedback is strong enough to yield a meaningful result.

3.5. Error Analysis

We conducted error analysis using the best hyperparameters for SVD and SVD++ derived from grid search and k-fold cross-validation.

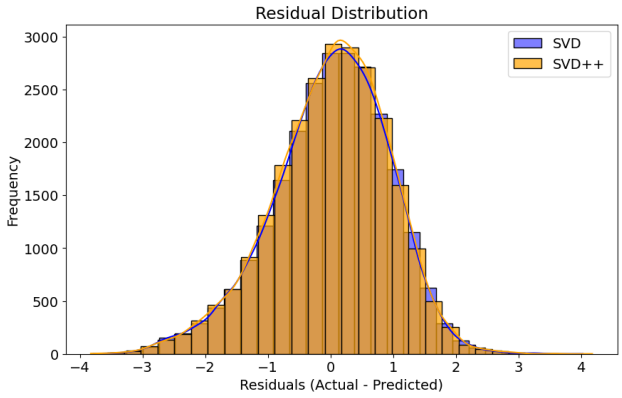

Figure 4. Distribution of Residual Errors for SVD and SVD++

Figure 4 presents two bell-shaped residual distributions. Both peaks are centered around 0, and the distributions spread on the x-axis from approximately -4 to 3, which are slightly left-skewed. The left skewness may indicate a slight bias that causes both models to more frequently underestimate than overestimate. SVD++ is slightly less left-skewed than SVD, indicating that it is less likely to underestimate the results and tends to provide a more accurate prediction. Nonetheless, the distribution difference between these two algorithms is very small, so the prediction powers between them are not significantly different.

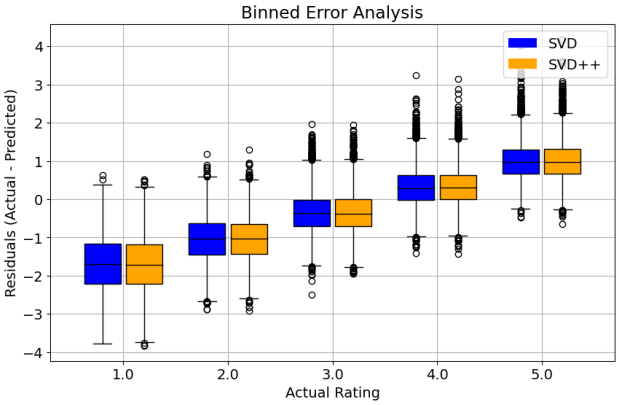

Figure 5. Binned Error Analysis for SVD and SVD++

Figure 5 shows that both models tend to underestimate, with more positive residuals. Both models achieve the best accuracy when the actual rating is 3.0 or 4.0. One potential explanation is that the dataset has more examples with a rating of 3.0 or 4.0 as shown in Table 1. Having more data points for these two ratings may thus lead to a better estimation.

Rating	Count
4.0	34174
3.0	27145
5.0	21201
2.0	11370
1.0	6110

Table 1. Number of Examples for Each Rating

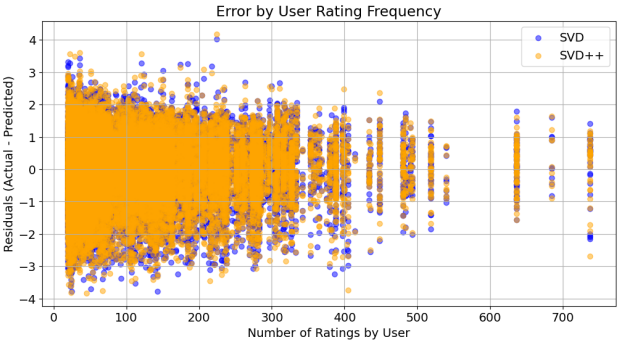

Figure 6. User Rating Frequency for SVD and SVD++

Figure 6 illustrates the relationship between the prediction residuals and the frequency of user ratings. We again observe that SVD and SVD++ produce similar results, with

SVD results slightly scattering more than SVD++, especially when the number of ratings by users is relatively small. As the bulk of the data points appear to cluster around the lower end of the number or ratings, the graph indicates that most users in the dataset only rated a small number of movies. These users tend to get a less accurate prediction, as the absolute residuals are approximately 3, which is relatively large. This shows an obvious trend where users who rate items more frequently tend to receive more accurate predictions from the system, as evidenced by the lower residual errors when the number of ratings increases.

Notably, there is a slight increase in prediction residuals when a user is a very frequent rater, with around 600 to 700 ratings in our graph. Since our models are essentially comparing similar users to provide predictions, the less accurate ratings for these frequent raters might result from having a small number of them, thus not having enough data to provide a precise estimation using the latent factors.

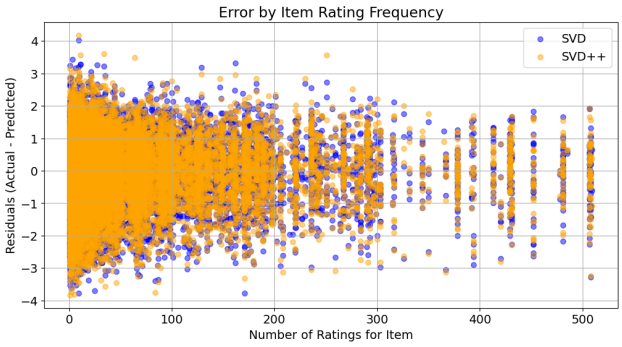

Figure 7. Item Rating Frequency for SVD and SVD++

Similarly in figure 7, the data points are clustered to the left of the graph, indicating that having fewer ratings will yield a larger residual, which is about 2.5 for less than 100 ratings. In terms of the difference between SVD and SVD++, we can see a more obvious scattering from SVD, indicating that SVD++ has slightly better predictions, since it concentrates more around 0. The predictions of SVD++ achieve the best accuracy when the item has around 100 - 200 ratings. Having even more ratings, such as around 400 to 500, gives a less accurate prediction, which reflects a similar issue of having a limited number of frequently rated movies.

4. Discussion

In general, our matrix factorization models are decently good at predicting movie ratings, with about 0.93 average rating deviance from the actual values for both models. In reality, similar models often achieve even higher accuracy, as they have larger datasets for reference and more types of implicit feedback that could better capture user-item interactions. The high accuracy, however, raises potential ethical concerns around issues like data privacy and feed manipulation.

In the past years, recommender systems have been developed focusing mostly on realizing firms' commercial objective to attract users and earn more profits. Companies that employ such models, mainly online businesses and services, will endeavor to increase the models' prediction powers by all means to increase their potential profits. This may lead to serious data privacy issues: as we can see from our matrix factorization models, having more implicit feedback may significantly increase the prediction accuracy, so the firms will have a strong incentive to gather whatever data they can find about the users in order to provide more accurate feed to them. This balance between collecting more private data to increase prediction performance while reducing the amount of privacy violations is important to discuss for any recommender systems. Assuming that most firms are profit-oriented, they often do not have incentives to stop their endless collections of data. Certain regulations are strongly needed to protect user data from being excessively exploited.

Another ethical issue is around feed manipulation. Most recommender systems will filter out some options according to certain algorithms. In most cases, this involves excluding information that does not align with a user's preferences or past behavior; however, since this information filtering and selection is mostly not controlled by users, deliberate manipulations could happen to fulfill certain parties' interests. In sectors such as healthcare and insurance plans, this process can be particularly morally significant and ethically complex, as it relates to an individual's health and safety. (Milano et al., 2020) In our example of making movie recommendations, the ethical stakes are much lower, but the models are still manipulable in many ways - for example, the company can feed a movie to more users just because the movie's director provided it extra money. Regulations of recommender systems, therefore, should demand more transparency of the models to detect not only privacy violations but also potential feed manipulations.

There are many more potential ethical topics related to recommender systems: is it moral to trap users forever with similar products? Should the users have the right to turn on or off the recommender system? While designing recommender systems, computer science researchers should seriously consider these ethical questions and adjust their algorithms accordingly.

5. Conclusions

In this paper, we implemented and tested the performance of two matrix factorization algorithms, SVD and SVD++.

We tuned three hyperparameters of each model: number of latent factors, learning rate, and regularization factor. Despite our hypothesis that SVD++ should perform better than SVD, we do not see a significant difference between SVD++ and SVD for our dataset. However, having more latent factors indeed increases the performance, as indicated by both models.

Due to computational constraints, our study has limited results that could lead to an inaccurate estimation. To thoroughly study the performance differences between the two algorithms, we need more testing with different datasets as well as different hyperparameters. To more accurately decide on the best hyperparameter values, we could vary more values for the hyperparameters. Just like methods in the surprise library, we could also apply a different learning rate or regularization factor for each value in the prediction function to fine-tune a better accuracy.

To further improve the accuracy of the model, we could incorporate more implicit feedback, such as temporal or spatial differences. Matrix factorization algorithms are scalable in implementation and allow multiple implicit feedback to be simultaneously included. Future works are needed to explore the accuracy performance after increasing the types of implicit feedback.

Acknowledgments

We would like to thank Swarthmore College and Professor Ben Mitchell for supporting our CPSC 066: Machine Learning Final Project. We would also like to thank GroupLens for providing the MovieLens dataset, and several previous works on matrix factorization models for providing valuable insight into implementation and hyperparameter tuning. ChatGPT has also been very helpful with plotting. Finally, we would like to thank the scikit-surprise library for providing the necessary functions.

References

GroupLens Research, None. Movielens 100k dataset. GroupLens, 1998. URL https://grouplens.org/datasets/movielens/. Available at: https://grouplens.org/datasets/movielens/.

Hyndman, Rob J. and Koehler, Anne B. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2006.03.001. URL https://www.sciencedirect.com/science/article/pii/S0169207006000239.

Koren, Yehuda. Factorization meets the neighborhood: A multifaceted collaborative filtering model. pp. 426–434, 08 2008. doi: 10.1145/1401890.1401944.

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

Milano, Silvia, Taddeo, Mariarosaria, and Floridi, Luciano. Recommender systems and their ethical challenges. *AI & Society*, 35, pages957–967:957–96, 2020.

Ricci, Francesco, Rokach, Lior, Shapira, Bracha, and Kantor, Paul B. *Recommender Systems Handbook*. Springer, 1st edition, 2010.

Salakhutdinov, Ruslan and Mnih, Andriy. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems 20*, 2008. URL https://papers.nips.cc/paper/3208-probabilistic-matrix-factorization.pdf.

Appendix

Appendix 1: Heatmaps for SVD++ Hyperparameter Tuning Results

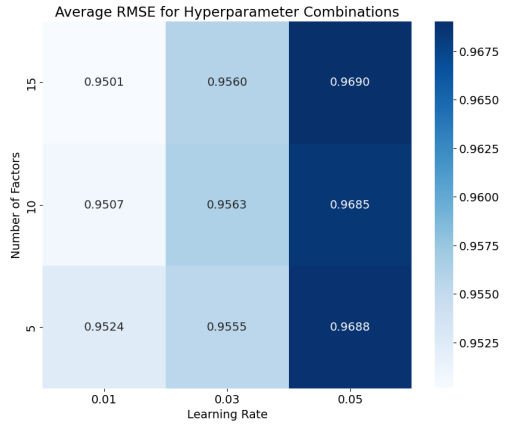

Figure 8. num_factors - learning_rate RMSE Heatmap for SVD++

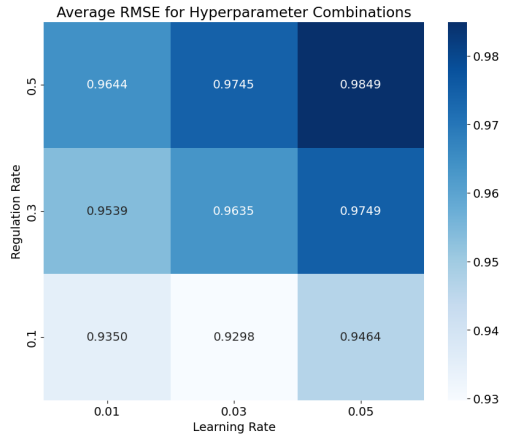

Figure 9. num_factors - reg RMSE Heatmap for SVD++

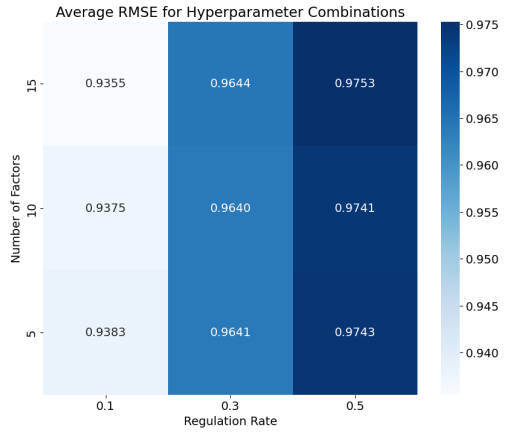

Figure 10. reg - learning_rate RMSE Heatmap for SVD++