University of Minnesota

Department of CSCI

Recommender Systems

Written by: Roozbeh Ehsani

Instructor: Professor Yousef Saad

1. Motivation:

Recommender systems have become an essential tool for e-commerce websites, social media platforms, and other online services to enhance customer engagement, satisfaction, and loyalty. One of the primary motivations behind the development of recommender systems is to help users overcome the problem of information overload. With the exponential growth of digital data, it has become increasingly difficult for users to find relevant products or content that match their interests and preferences. Recommender systems address this problem by using machine learning algorithms to provide personalized recommendations to users based on their historical behavior and feedback.

Another motivation behind the use of recommender systems is to improve business performance and revenue generation for companies. By providing personalized recommendations, companies can increase customer retention, reduce churn rates, and enhance customer lifetime value. Recommender systems also facilitate cross-selling and up-selling opportunities by suggesting complementary or premium products to users based on their preferences and purchase history. In addition, recommender systems can help companies optimize their inventory management and supply chain operations by predicting demand patterns and identifying popular products.

Recommender systems also have significant implications for research and innovation in various fields, including computer science, data science, and social sciences. By providing insights into user behavior and preferences, recommender systems can contribute to the development of new algorithms, theories, and models that advance the state-of-the-art in these fields. Recommender systems also play a crucial role in facilitating social interaction and information sharing among users, leading to the emergence of new communities, trends, and ideas. Overall, the motivations for using recommender systems are multi-faceted and far-reaching, encompassing economic, social, and scientific domains.

2. Methods:

Recommender systems are utilized in various domains to help users discover new items that match their interests. There are several methods used in recommender systems, including:

1. User-based collaborative filtering: This algorithm recommends items based on similarities between users' past behavior, such as ratings or purchases.

2. Item-based collaborative filtering: This algorithm recommends items based on similarities between items themselves, such as genre, actors, or tags.

3. Matrix factorization: This algorithm involves decomposing a user-item matrix into two lower-dimensional matrices, which can be used to make predictions about users' preferences.

4. Singular value decomposition (SVD): This algorithm is similar to matrix factorization, but involves using the SVD decomposition of the user-item matrix.

5. Content-based filtering: This algorithm recommends items based on similarities between the features of the items themselves, such as genre, actors, or tags.

6. Hybrid recommender systems: These systems combine multiple algorithms to improve recommendations. For example, a hybrid system might use both collaborative filtering and content-based filtering to make recommendations.

In this survey, I will provide an explanation of collaborative filtering and matrix factorization techniques.

3. Dataset

The 100k MovieLens dataset is a well-known benchmark dataset for recommender systems. It consists of 100,000 ratings on a scale of 1-5 provided by 943 users for 1,682 movies. The dataset also includes information about the movies such as their titles, release years, and genres. Additionally, it includes demographic information about the users such as age, gender, and occupation. The dataset was collected by the GroupLens research group at the University of Minnesota (MovieLenz-100k).

The dataset is often used for evaluating collaborative filtering and other recommendation algorithms. It is relatively small compared to other movie recommendation datasets, which makes it easy to work with and analyze. However, it still contains enough data to make meaningful recommendations and evaluate the performance of different recommendation algorithms. Many researchers and companies have used the dataset to develop and test new recommendation techniques.

Overall, the 100k MovieLens dataset is a widely used and well-known benchmark dataset in the field of recommender systems, and it has played an important role in the development and evaluation of many recommendation algorithms. I have used it as an input dataset in order to evaluate models.

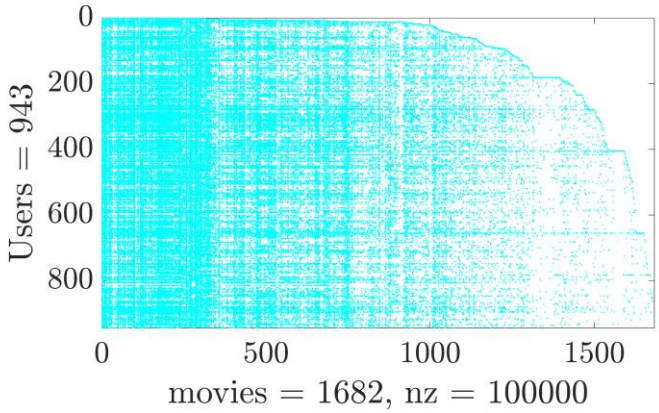

Figure 1: Spy illustration of the rating matrix *R*

For the evaluation of the performance of each method, the dataset is split up into 80 percent training dataset, and 20 percent test dataset. Then after the prediction, the mean of the root square of deviation between the predicted rate ($\hat{r}_{ui}$) and the actual rate (r_{ui}^t) from the test dataset is calculated.

$$RMSE = \sqrt{\frac{1}{n}\sum_{u}\sum_{i:r_{ui}^t>0}\left(\hat{r}_{ui} - r_{ui}^t\right)^2}$$

4. Models

4.1 User-based Collaborative filtering:

User-based collaborative filtering is a popular technique used in recommender systems to make personalized recommendations to users. The idea behind this approach is to identify users who have similar preferences and tastes, and recommend items to a user based on what other similar users have liked.

To implement user-based collaborative filtering, a user-item matrix R with the size of $R \in \mathbb{R}^{m\times n}$ based on the training dataset is constructed where each row represents a user (m users), and each column represents an item (n items). The entries in the matrix indicate the ratings that a user has given to the items.

$$R = \begin{bmatrix} r_{11} & \cdots & r_{1i} \\ \vdots & \ddots & \vdots \\ r_{u1} & \cdots & r_{ui} \end{bmatrix}_{m\times n}$$

Similarity between users is then measured using a similarity metric, such as cosine similarity or Pearson correlation. Here I used cosine similarity between rating vector of two users.

$$Sim(UB)_{(u,u')} = \frac{\sum_i^n r_{ui}\, r_{u'i}}{\sqrt{\sum_i^n r_{ui}^2}\sqrt{\sum_i^n r_{u'i}^2}}$$

Once the similarities between users are calculated, the system identifies a set of similar users to the target user and uses their ratings to predict the target user's rating for an item. Figure 2 shows the contour plot of cosine similarity between the users $Sim(UB)$ which its size is $(m \times m)$. On the diagonal entries, all the values are one which shows maximum similarity. For identifying set of similar users, a limited number of most similar users are elected which are called "Neighbors" to our target (i.e., K). Based on the similarity of neighboring users u' to our target user u, the rating for the specific item i is calculated using weighted averaging. The predicted rating $\hat{r}_{ui}$ can then be used to make recommendations to the user.

$$\hat{r}_{ui} = \sum_{u'}^{K} Sim(UB)_{(u,u')} r_{u'i} \Big/ \sum_{u'}^{K} Sim(UB)_{(u,u')}$$

The number of neighbors K is changed iteratively to find most appropriate one. Here the number of neighbors is varying from 1 to 20 (i.e., $K \in [1\ 20]$). The effect of number of neighbors on the RMSE will be discussed later on.

While user-based collaborative filtering is a simple and intuitive approach, it can suffer from the cold-start problem, where new users have no ratings and thus no similar users can be found.

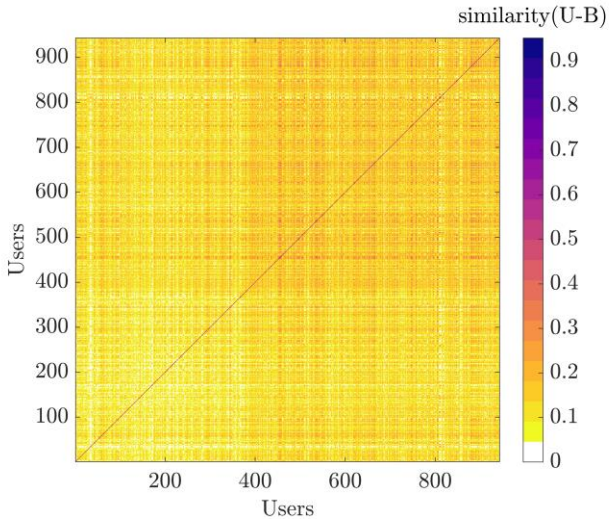

Figure 2: Cosine similarity between the users based on the training dataset. The diagonal entries are all 1 which shows the maximum similarity of every user with itself.

4.2 Item-based Collaborative filtering:

Item-based collaborative filtering is a recommendation technique that focuses on finding the similarity between items based on the ratings given by users. Once the similarity between items is calculated, the ratings of similar items are used to predict the rating of users for items that they have not yet rated.

To implement item-based collaborative filtering, the first step is to create an item-item similarity matrix. This matrix represents the similarity between all pairs of items in the dataset based on the ratings given by the users. The similarity between items can be calculated using various methods such as cosine similarity, Pearson correlation, or Jaccard similarity. Here I used cosine similarity between rating vector of two items.

$$Sim(IB)_{(i,i')} = \frac{\sum_u^m r_{ui}\, r_{ui'}}{\sqrt{\sum_u^m r_{ui}^2}\sqrt{\sum_u^m r_{ui'}^2}}$$

The contour plot of cosine similarity between items is presented in Figure 3, where the matrix size $Sim(IB)$ is $(n \times n)$.

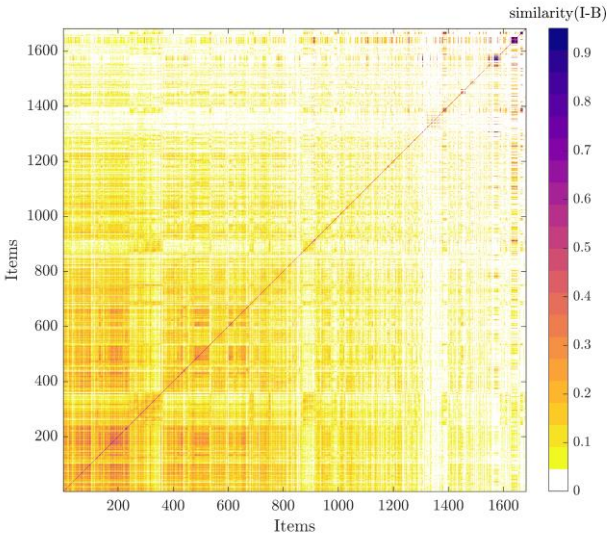

Figure 3: Cosine similarity between the items based on the training dataset. The diagonal entries are all 1 which shows the maximum similarity of every item with itself.

Once the item-item similarity matrix is calculated, the ratings of similar items are used to predict the rating of users for items that they have not yet rated. Similar to the user-based collaborative filtering matrix, for the item-based similarity matrix, the diagonal entries are one. In order to identify similar items, limited number of items for a user are selected which are called "Neighbors" to our target (i.e., K). Based on the similarity of neighboring items i' to our target item i, the rating for a user for the specific item i is calculated using weighted averaging. The predicted rating $\hat{r}_{ui}$ can then be used to make recommendations to the user.

$$\hat{r}_{ui} = \sum_{i'}^{K} Sim(IB)_{(i,i')} r_{ui'} \Big/ \sum_{i'}^{K} Sim(IB)_{(i,i')}$$

The number of neighbors is changed iteratively to find most appropriate one. Here the number of neighbors is varying from 1 to 20 (i.e., $K \in [1\ 20]$). The effect of number of neighbors on the RMSE for both models is studied. RMSE shows the square difference between the predicted rates $\hat{r}_{ui}$ base on the training dataset with the existing rates in the test dataset $r_{ui}^{t} > 0$. Figure 4 shows the value of RMSE for different number of neighbors. As the number of neighbors increases, the magnitude of error is decreasing. The result shows that the performance of item-based collaborative filtering is better than user-based collaborative filtering because:

1. Items are simpler, users have multiple tastes.

2. People are more complex than objects.

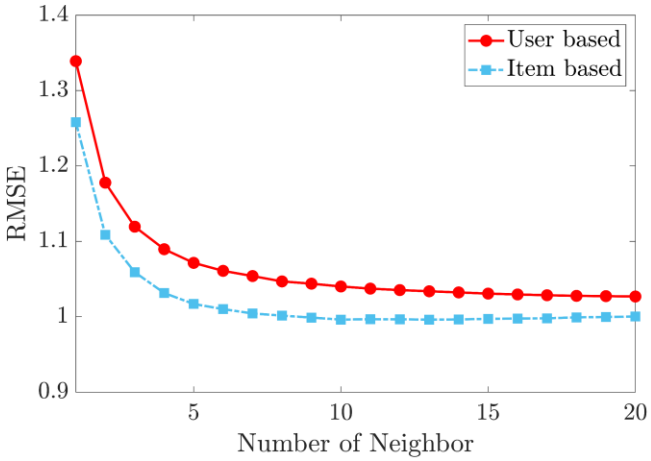

Figure 4: The value for RMSE for different number of neighbors. Red circles correspond to User-based collaborative filtering and blue squares belong to Item-based collaborative filtering.

Some challenges of collaborative filtering

Data Sparsity: User in general rate only a limited number of items.

Cold star: Difficulty in recommendation to new users or new items.

Scalability: Increase in number of users of items results in decreasing the performance of the system in computing the similarity.

To resolve these issues: Hybrid recommender systems.

Codes for these two methods in Matlab script are available in last pages

4.3 Matrix factorization:

Matrix factorization is a technique used in recommender systems to predict user preferences or ratings for items. This model is a generalization of the SVD model, which intends to find a low-rank matrix factorization to approximate user-item rating matrix R. Suppose that f- dimensional vector is associated with each user u (i.e., user factor vector $x_u \in \mathbb{R}^f$) and f-dimensional vector is associated with each item i (i.e., item factor vector $y_i \in \mathbb{R}^f$). The prediction is done by taking an inner product (i.e., $\hat{r}_{ui} = x_u^T y_i$). The row of matrix X contains all the x_u^T vectors associated with each user (i.e., $X \in \mathbb{R}^{m \times f}$), and the row of matrix Y contains all the y_i^T vectors associated with each item (i.e., $Y \in \mathbb{R}^{n \times f}$). The purpose of this model is to find user-factor matrix X and item-factor matrix Y to predict the ratings (i.e., $\hat{R} = XY^T$) such that:

$$J = \min_{X_*, Y_*} \|R - XY^T\|_F^2$$

One possible issue is that minimizing the Frobenius norm may increase the likelihood of overfitting, as the algorithm may prioritize fitting the known entries too closely. In order to avoid overfitting the user-item rating matrix $\hat{R}$, the l_2 norm regularized matrix factorization method is employed.

$$J = \min_{X_*, Y_*} \|R - XY^T\|_F^2 + \lambda(\|X\|_F^2 + \|Y\|_F^2)$$

The term $\lambda(\|X\|_F^2 + \|Y\|_F^2)$ is necessary for regularizing the model such that it will not overfit the training dataset. Note that if we fix either the user-factor X or the item-factor Y, the cost function J becomes quadratic, and we can easily compute its global minimum. This observation leads to an optimization process called *alternating least squares*, where we alternate between re-computing the user-factor and item-factor. At each step, the value of the cost function is guaranteed to decrease, making the convergence to the global minimum more efficient. To find the value of X or Y at each step, the gradient of cost function with respect to that parameter should be equal to zero (i.e., $\frac{\partial J}{\partial X} = 0$, $\frac{\partial J}{\partial Y} = 0$).

$$J = \|R - XY^T\|_F^2 + \lambda(\|X\|_F^2 + \|Y\|_F^2) \rightarrow \|R\|_F^2 + \|XY^T\|_F^2 - 2tr(R^T XY^T) + \lambda(\|X\|_F^2 + \|Y\|_F^2)$$

$$J = tr(R^T R) + tr((XY^T)^T XY^T) - 2tr(R^T XY^T) + \lambda(tr(X^T X) + tr(Y^T Y))$$

$$\frac{\partial J}{\partial X} = dtr((XY^T)^T XY^T) - 2dtr(R^T XY^T) + \lambda(dtr(X^T X))$$

$$\frac{\partial J}{\partial X} = tr((dXY^T)^T XY^T) + tr((XY^T)^T dXY^T) - 2tr(R^T dXY^T) + \lambda(tr(dX^T X) + tr(X^T dX))$$

$$\frac{\partial J}{\partial X} = tr(YdX^T XY^T) + tr(YX^T dXY^T) - 2tr(R^T dXY^T) + \lambda(tr(dX^T X) + tr(X^T dX)) \xrightarrow{tr(AA^T)=tr(A^T A)}$$

$$\frac{\partial J}{\partial X} = tr(YX^T dXY^T) + tr(YX^T dXY^T) - 2tr(R^T dXY^T) + \lambda(tr(X^T dX) + tr(X^T dX)) \xrightarrow{tr(ABC)=tr(CAB)}$$

$$\frac{\partial J}{\partial X} = 2tr(Y^T YX^T dX) - 2tr(Y^T R^T dX) + \lambda\big(2tr(X^T dX)\big) = 0$$

$$Y^T YX^T - Y^T R^T + \lambda X^T = 0$$

$$X^T = (Y^T Y + \lambda I)^{-1} Y^T R^T$$

where I is an $f \times f$ identity matrix. The user-factors vector for each user is:

$$x_u = (Y^T Y + \lambda I)^{-1} Y^T r_u^T$$

In which r_u is rating row vector of user u in the training dataset (i.e., $r_u \in \mathbb{R}^{1\times n}$). After doing same procedure for $\frac{\partial J}{\partial Y} = 0$ we have:

$$\frac{\partial J}{\partial Y} = 2tr(X^T XY^T dY) - 2tr(X^T RdY) + \lambda\big(2tr(Y^T dY)\big) = 0$$

$$Y^T = (X^T X + \lambda I)^{-1} X^T R$$

The item-factors vector for each item is:

$$y_i = (X^T X + \lambda I)^{-1} X^T r_i$$

In which r_i is rating column vector of item i in the training dataset (i.e., $r_i \in \mathbb{R}^m$). This model is implemented for different number of factors. Figure 5 shows the RMSE for the different number of factors. As the number of factors increases, the RMSE decreases because the model embeds more detail about the characteristics of users and items and can predict the rating better.

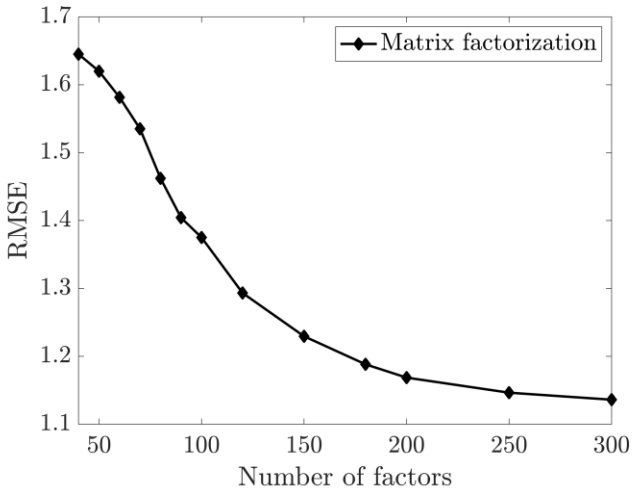

Figure 5: RMSE for the different number of factors for matrix factorization method.

Codes for Matrix factorization model in Matlab script is available in last pages

The input dataset could have different types. Most convenient is the high-quality ***explicit feedback***, which includes explicit input by users regarding their interest in product. For example, Netflix collects star rating for movie or 100k MovieLens dataset contains ratings of users on a scale of 1 to 5. The user-item rating matrix R that we have used in previous sections is explicit dataset. However, explicit feedback is not always available.

4.4 Matrix factorization (Implicit Feedback Dataset):

Recommender systems have the ability to provide users with suggestions based on the ***implicit feedback***, which is more abundant and indirectly reflects the user's opinion through their observed behavior. The input data associate users and items through r_{ui} values, is the "observation". For instance, r_{ui} can indicates the number of times user u bought item i or how many times user u fully watched movie i. $r_{ui} = 0.6$ means that user u watched 60 percent of movie i. In order to use implicit dataset as our input dataset for the model, two other user-item matrices should be introduced. First, I should introduce set of binary variables p_{ui}, which indicates the preference of user u, to item i.

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

In other words, if user u, purchased or watched item i (i.e., $r_{ui} > 0$), the value for $p_{ui} = 1$ regardless of value of r_{ui} otherwise $p_{ui} = 0$. The size of the matrix P is same as the size of rating matrix ($P \in \mathbb{R}^{m \times n}$). Not watching or purchasing item i by the user u, could stem from different reasons such as the user u might be unaware of the existence of the item i or unable to buy it because of its price. On the other hand, watching or purchasing item i, could not strictly mean that user u, have preference for the item. User could purchase item for his/her friend or maybe watch a TV show just because user stayed on the channel of the previously watched show. In general, as r_{ui} increases, we have a stronger indication that the user u like the item i. As a result, the second set of variables should be introduced c_{ui} which measures our confidence in observing p_{ui}.

$$c_{ui} = 1 + \alpha r_{ui}$$

In this way, we may have more confidence in the value of p_{ui} for every user-item pairs. The size of the matrix C is same as the size of matrix P ($C \in \mathbb{R}^{m \times n}$). In the experiment, $\alpha = 40$ may result in good results. Still, our purpose is to find user-factor vectors x_u for each user and item-factor vectors y_i for each item in order to predict preferences of a user to an item ($\hat{p}_{ui} = x_u^T y_i$). Accordingly, the factors are computed by minimizing the following cost function:

$$J = \min_{x_*, y_*} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2)$$

The alternative lease square optimization process is used to find for the value of factor matrices. The first step is computing all user-factor. In order to do so, for each user a diagonal $n \times n$ matrix C^u where $C^u_{ii} = c_{ui}$ and the vector $p(u) \in \mathbb{R}^n$ that contain all the preferences by user u is defined. The derivative of cost function with respect to x_u ($\frac{\partial J}{\partial x_u} = 0$) is:

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

Calculating the matrix $Y^T C^u Y$ is computationally expensive $O(f^2 n)$ and this matrix for each user could be calculated in an alternative efficient way. Before looping through each user, the matrix $Y^T Y$ is calculated which is the same for all users and it can be calculated once in each step with the cost of $O(f^2 n)$. Then a remarkable speedup is achieved by using the fact that $Y^T C^u Y = Y^T Y + Y^T (C^u - I) Y$. $Y^T Y$ is independent of users u and already is precalculated. For $Y^T (C^u - I) Y$, the middle term $(C^u - I)$ has only n_u non-zero elements, where n_u is number of items for which $r_{ui} > 0$ for the user u for different items i ($n_u \ll n$). Similarly, $C^u p(u)$ contains n_u number non-zero elements. For inversion of $(Y^T C^u Y + \lambda I)^{-1}$, $O(f^3)$ is assumed. Consequently, computation of x_u for each user in time is $O(f^2 n_u + f^3)$ and for all users will give rise to $O(f^2 \mathcal{N} + f^3 m)$ in which $\mathcal{N}$ is overall number of non-zero observations ($\mathcal{N} = \Sigma_u n_n$).

Similarly for item-factor vector for each item we have:

$$y_i = \left(X^T C^i X + \lambda I\right)^{-1} X^T C^i p(i)$$

Where C^i is diagonal $m \times m$ matrix $C^i_{uu} = c_{ui}$ for each item and $p(i) \in \mathbb{R}^m$ contains all the preferences for i. Likewise, $X^T C^i X = X^T X + X^T (C^i - I) X$ can be computed in efficient way. $X^T X$ is independent of items i and already is precalculated. The running time of this step would be $O(f^2 \mathcal{N} + f^3 n)$. The whole

process scales linearly with the size of the data. After computing the user- and item-factor, we recommend to user, The K available items with the largest value of $\hat{p}_{ui} = x_u^T y_i$.

The foundation of this technique is defined but one can have some modifications in defining the parameters such as $p_{ui}, c_{ui}, \hat{p}_{ui}$ ..... For example, confidence matrix could be changed into $c_{ui} = 1 + \alpha \log(1 + r_{ui}/\epsilon)$ or predicted preference matrix could be defined as $\hat{p}_{ui} = \sum_{j:r_{uj}>0} s_{ij}^u c_{uj}$. Where $s_{ij}^u = y_i^T W^u y_j$ is item-item similarity matrix for user u ($S^u \in \mathbb{R}^{n \times n}$), and $W^u = (Y^T C^u Y + \lambda I)^{-1}$is considered as a weighting matrix associated with user u ($W^u \in \mathbb{R}^{f \times f}$). Also, one can define different thresholds for calculating the value of p_{ui} based on the r_{ui}.

The dataset is divided into training and test dataset such that the model is trained with 4 successive weeks of data and then predict the preference matrix $\hat{p}_{ui}$ for the users for the ensuing week and then it is compared with the preference matrix of test dataset p_{ui}^t. The preference matrix for test dataset p_{ui}^t is calculated based on user-item observation matrix r_{ui}^t.

For the evaluation of the model, the RMSE cannot be used anymore because we do not have explicit ranking by the users. Also, for $r_{ui}^t = 0$ does not necessarily mean that the user does not like the item and could stem from different reasons. However, watching a program or purchasing an item frequently is an indication of liking it. Consequently, we can just use the predicted preference of users $\hat{p}_{ui}$ that its corresponding values in the test dataset be high p_{ui}^t. In other words, the condition of using $\hat{p}_{ui}$ for model evaluation is having high corresponding value of p_{ui}^t in the test dataset because we are confident that the user likes the item. This approach to the evaluation of the model is recall-oriented measurement ($TP/(TP + FN)$). $rank_{ui}$ is the percentile-ranking of program i within the ordered list of all programs prepared for user u. $rank_{ui} = 0\%$ would mean the program i is predicted to be the most desirable for user u. $rank_{ui} = 100\%$ indicates that program i is predicted to be the least preferred for user u. The overall quality measurement of the model could be expressed as:

$$\overline{rank} = \frac{\sum_{u,i} r_{ui}^t rank_{ui}}{\sum_{u,i} r_{ui}^t}$$

The lower value of $\overline{rank}$ is desirable. Since we just choose the preferences of users $\hat{p}_{ui}$ with high corresponding value of p_{ui}^t, we expect to $rank_{ui}$ for those users to be low value and as a result, overall rank is desired to be low ($\overline{rank} \approx 0$). If $\overline{rank}$ for the model be greater than 50% indicates that the model is not better than random model.

[3] implemented this model for the implicit data set for different number of factors (f), ranging from 10 to 200 and compare it with two other model. The first model is recommending shows base on the popularity of models which is independent of factors. The second model is neighborhood based (item-item) which is also factor independent. Figure 6 shows the measured values of $\overline{rank}$ with different number of factors. We can see that the recommender model just based on the popularity is not personalized and its $\overline{rank}$ is equal to 16.46%. For the item-item neighborhood model, we have remarkable improvement $\overline{rank} = 10.47\%$. The better results are achieved by factor model $\overline{rank} = 8.35\%$ for 200 factors.

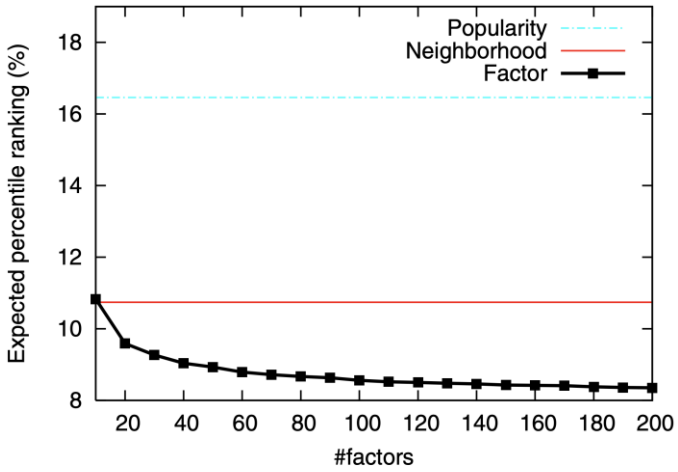

Figure 6: Comparing factor model with popularity ranking and neighborhood model [3].

5. Conclusion

Different models of recommender systems are discussed. The 100k MovieLens dataset is used for training and evaluation of model's performance. Both user-based and item-based collaborative filtering models are doing well as the number of neighbors increases. But overall, the item-based model has better performance than the user-based model. Implementation of Matrix factorization model based on two different types of dataset is introduced. Explicit dataset is valuable, but it is not always available. Consequently, the model based on the implicit dataset which shows the watching behavior of users is implemented. The performance of both matrix factorization models is getting better as the number of factors increases since factors embed the users and items characteristics and as the number of factors increases the model would contains more information about users and items.

Future works

Use modified projections methods and preconditioners in order to solve user-factor X and item-factor Y matrices.

References:

[1] Ramlatchan, Andy, et al. "A survey of matrix completion methods for recommendation systems." *Big Data Mining and Analytics* 1.4 (2018): 308-323.

[2] Sarwar, Badrul, et al. *Application of dimensionality reduction in recommender system-a case study*. Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[3] Hu, Yifan, Yehuda Koren, and Chris Volinsky. "Collaborative filtering for implicit feedback datasets." *2008 Eighth IEEE international conference on data mining*. Ieee, 2008.

[4] Saad, Yousef. Numerical methods for large eigenvalue problems: revised edition. Society for Industrial and Applied Mathematics, 2011.

Matlab Code:

```
%% Load data
close all
clc

dataT = readtable('ml-100k/data.csv', 'ReadVariableNames', true);% ratings data
userT = readtable('ml-100k/user.csv', 'ReadVariableNames', true);% user data


data = table2array(dataT);

% Create user-item matrix
R = zeros(max(data(:,1)), max(data(:,2)));
for i = 1:size(data,1)
    R(data(i,1), data(i,2)) = data(i,3);
end

Nuser=size(R,1);
Nmovies=size(R,2);

figure
spy(R,'c.') %1,486,126 zeros
xlabel(sprintf('movies = %d, nz = %d',size(R,2),nnz(R)),'Interpreter','latex');
ylabel(sprintf('Users = %d', size(R,1)),'Interpreter','latex');
set(gca,'TickLabelInterpreter','latex')
set(gca,'FontSize',16)


%% Test and Train dataset

filename = 'ml-100k/u1.base';
dataTrain = load(filename);

filename = 'ml-100k/u1.test';
dataTest = load(filename);

Rtrain = zeros(max(data(:,1)), max(data(:,2)));
for i = 1:size(dataTrain,1)
    Rtrain(dataTrain(i,1), dataTrain(i,2)) = dataTrain(i,3);
end

Rtest = zeros(max(data(:,1)), max(data(:,2)));
for i = 1:size(dataTest,1)
    Rtest(dataTest(i,1), dataTest(i,2)) = dataTest(i,3);
end

%% User-based(UB)


% Compute similarities between users using cosine similarity
simUB = pdist(Rtrain, 'cosine');
simUB = 1 - squareform(simUB);

figure
contourf(simUB,20,'EdgeColor','none')
cmap1 = plasma(20); % Call cbrewer and get colormap
cmap1 = flip(cmap1);         % flip the colorbar
colormap(cmap1);
C1=cmap1;
C1=[1,1,1;C1];
colormap(gca,C1)
hcb1=colorbar;
title(hcb1,'similarity(U-B)','Interpreter','Latex','FontSize',13);
hcb1.TickLabelInterpreter = 'latex';
```

```
ylabel('Users','Interpreter','Latex','FontSize',13);
xlabel('Users','Interpreter','Latex','FontSize',13);
hcb1.Label.Interpreter = 'latex';
axis vis3d
set(gca,'TickLabelInterpreter','latex')
set(gca,'FontSize',13)
set(gca, 'color', 'none');


% Predict ratings for users

N=1:20;
meanMAEUB=zeros();


for k=1:size(N,2)

    PrirateUB = zeros(size(Rtrain,1),size(Rtrain,2));

    K=N(1,k);
    for j=1:size(Rtrain,1)
        [IDXnonrated]=find(~Rtrain(j,:));


        for i=1:size(IDXnonrated,2)

            [IDXrated]=find(Rtrain(:,IDXnonrated(1,i)));
            if size(IDXrated,1)>=K

                [W,I] = sort(simUB(j,IDXrated),'descend');
                W = W(1,1:K);
                PrirateUB(j,IDXnonrated(1,i))=(W*Rtrain(IDXrated(I(1,1:K),1),IDXnonrated(1,i)))/sum(W);
            else
                W=simUB(j,IDXrated);
                PrirateUB(j,IDXnonrated(1,i))=(W*Rtrain(IDXrated,IDXnonrated(1,i)))/sum(W);
            end
            if isnan(PrirateUB(j,IDXnonrated(1,i)))
                PrirateUB(j,IDXnonrated(1,i))=0;
            end
        end
    end


    [r,c]=find(Rtest);

    MAEUB = zeros();

    for i=1:size(r,1)

        MAEUB(1,i) = (Rtest(r(i,1),c(i,1))-PrirateUB(r(i,1),c(i,1)))^2;

    end

    meanMAEUB(1,k) = sqrt(mean(MAEUB,2));

end

%% Item-based(IB)

% Compute similarities between items using cosine similarity
simIB = pdist(Rtrain', 'cosine');
simIB = 1 - squareform(simIB);

figure
contourf(simIB,20,'EdgeColor','none')
cmap1 = plasma(20); % Call cbrewer and get colormap
cmap1 = flip(cmap1);         % flip the colorbar
colormap(cmap1);
C1=cmap1;
C1=[1,1,1;C1];
colormap(gca,C1)
hcb1=colorbar;
title(hcb1,'similarity(I-B)','Interpreter','Latex','FontSize',13);
hcb1.TickLabelInterpreter = 'latex';
ylabel('Items','Interpreter','Latex','FontSize',13);
xlabel('Items','Interpreter','Latex','FontSize',13);
hcb1.Label.Interpreter = 'latex';
axis vis3d
set(gca,'TickLabelInterpreter','latex')
set(gca,'FontSize',13)
set(gca, 'color', 'none');


% Predict ratings for users

meanMAEIB=zeros();


for k=1:size(N,2)
    PrirateIB = zeros(size(Rtrain,1),size(Rtrain,2));

    K=N(1,k);


    for j=1:size(Rtrain,1)

        [IDXnonrated]=find(~Rtrain(j,:));


        for i=1:size(IDXnonrated,2)
```

```
            [IDXrated]=find(Rtrain(j,:));

            if size(IDXrated,2)>=K

                [W,I]=sort(simIB(IDXnonrated(1,i),IDXrated),'descend');
                W = W(1,1:K);

                PrirateIB(j,IDXnonrated(1,i))=(W*Rtrain(j,IDXrated(1,I(1,1:K)))')/sum(W);
            else
                W = simIB(IDXnonrated(1,i),IDXrated);
                PrirateIB(j,IDXnonrated(1,i))=(W*Rtrain(j,IDXrated)')/sum(W);
            end
            if isnan(PrirateIB(j,IDXnonrated(1,i)))
                PrirateIB(j,IDXnonrated(1,i))=0;
            end
        end
    end
    [r,c]=find(Rtest);

    MAEIB = zeros();

    for i=1:size(r,1)

        MAEIB(1,i) = (Rtest(r(i,1),c(i,1))-PrirateIB(r(i,1),c(i,1)))^2;

    end

    meanMAEIB(1,k) = sqrt(mean(MAEIB,2));

end

%%

figure
plot(N,meanMAEUB,'r','LineWidth',2,'LineStyle','-',...
    'Marker','o','MarkerSize',8,'MarkerFaceColor','r')
hold on
plot(N,meanMAEIB,'color',[0.30,0.75,0.93],'LineWidth',2,'LineStyle','-.',...
    'Marker','square','MarkerSize',8,'MarkerFaceColor',[0.30,0.75,0.93])
ylim([0.9 1.4])
xlim([1 20])
xlabel('Number of Neighbor','Interpreter','Latex');
ylabel('RMSE','Interpreter','Latex');
legend('User based','Item based','Interpreter','Latex','FontSize',20)
set(gca,'TickLabelInterpreter','latex')
set(gca,'FontSize',20)

%% Matrix Factorization
close all

factors = [40,50,60,70,80,90,100,120,150,180,200,250,300];

meanMAEMF=zeros();
NIter = 20;

for k= 1:size(factors,2)

    f = factors(1,k);

    X = randn(size(Rtrain,1),f);
    Y = randn(size(Rtrain,2),f);
    lambda = 0.1;

    for iter=1:NIter

        % update item factors
        for j = 1:size(Rtrain,2)
            % find the set of users that rated item j
            idx = find(Rtrain(:, j) ~= 0);
            % calculate the matrix Xj from user factors
            Xj = X(idx, :);
            % update item j factor using least squares
            Yj = (Xj' * Xj + lambda * eye(f)) \ (Xj' * Rtrain(idx, j));
            % update the item factor matrix Y
            Y(j, :) = Yj';
        end


        % update user factors
        for i = 1:size(Rtrain,1)
            % find the set of items that user i rated
            idx = find(Rtrain(i, :) ~= 0);
            % calculate the matrix Yi from item factors
            Yi = Y(idx, :);
            % update user i factor using least squares
            Xi = (Yi' * Yi + lambda * eye(f)) \ (Yi' * Rtrain(i, idx)');
            % update the user factor matrix X
            X(i, :) = Xi';
        end


    end

    PrirateMF = X*Y';

    [r,c]=find(Rtest);
```

```
    MAEMF = zeros();

    for i=1:size(r,1)

        MAEMF(1,i) = (Rtest(r(i,1),c(i,1))-PrirateMF(r(i,1),c(i,1)))^2;

    end

    meanMAEMF(1,k) = sqrt(mean(MAEMF,2));

end


figure
plot(factors,meanMAEMF,'k','LineWidth',2,'LineStyle','-',...
    'Marker','d','MarkerSize',8,'MarkerFaceColor','k')
% ylim([0.5 2])
xlim([40 300])
xlabel('Number of factors','Interpreter','Latex');
ylabel('RMSE','Interpreter','Latex');
legend('Matrix factorization','Interpreter','Latex','FontSize',20)
set(gca,'TickLabelInterpreter','latex')
set(gca,'FontSize',20)
```