STAT 578 STATISTICAL LEARNING IN DATA SCIENCE COURSE PROJECT
CAUSAL INFERENCE FOR RECOMMENDATION

Gizem Tabak
(tabak2)
May 8, 2017

1. INTRODUCTION

Collaborative filtering is one of the two fundamental ideas used in recommender systems. It depends on the user preferences and similarities in their taste, rather than item type and content or user properties and demographics. A collaborative filtering based recommender system can be thought of as a matrix completion problem (Fig. 1), if each row represents the users and each column represents the items in the dataset, and the elements of the matrix represent user's reaction, which can be their rating for movie recommendation, or whether or not they click on a link for an advertisement system, and so on. This sparsely known matrix completion problem is addressed by Salakhutdinov et. al with probabilistic matrix factorization (PMF) [1], which assumes latent Gaussian models on the user preferences (user intrinsic scores) and item properties (item intrinsic scores), and learns those latent model parameters with gradient methods in order to reconstruct the matrix and fill up the unknown elements.

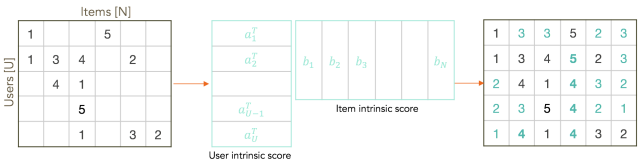


Fig. 1. Movie recommender system problem posed as matrix completion problem

One important drawback of probabilistic matrix factorization, also encountered in many other data science applications, is being a driven approach. Hence, the results reflect the bias of the data used in the problem. While it is easy to obtain unbiased data in some applications, it might not naturally be possible in some others. For example, in a movie recommendation problem, it is obvious that "popular" movies are being rated more than the "unpopular" ones, and hence they are encountered more frequently in the dataset. Or a user is more likely to click on an ad link if he is exposed to it more than others. As a result, more popular or more exposed items have bigger influence on the learning process than the unpopular or less exposed ones, and hence the resulting recommender system obtains a bias in favor of those. However, in order to construct an unbiased recommender system, the data should be as if it is coming from a randomized experiment, since the ultimate purpose of the recommender system is to discover user's true preference of an item independent of its popularity or of its exposure to the user. However, it is hard to perform a randomized experiment in many of these applications since it will be hard and time consuming.

There has been several works that aim eliminating this bias in the dataset [2], [3]. The main difference between these two works is Schnabel et. al [3] debias the data using empirical risk minimization and assuming a self-selection model based on user's preferences, whereas Liang et. al [2] use a Bayesian exposure model. In this project, I focused on the work of Liang et. al [2] since their motivation seemed more intuitional to me, and I am more familiar with Bayesian settings than empirical risk minimization.

In their recent work, Liang et. al [2] introduced a causal inference approach to obtain results as if they are obtained through a randomized experiment without actually performing a real randomized experiment. They proposed to weight the data samples according to different "exposure" models of an item to a user. For example, they inversely weight items based on their popularity, and then they performed probabilistic matrix factorization using this weighted dataset so that the resulting ratings would be as if the users have been exposed to items with the same probability, and this prevents the bias caused by item popularity in their ratings. Details of this approach as well as probabilistic matrix factorization will be given in Section 2.

Using weighted samples with different exposure models, Liang et. al [2] show improvement in the prediction accuracy compared to the standard unweighted PMF scheme. However, they did not evaluate the system in terms of hidden gem discovery. Since the weighting process yields an unbiased recommender, it is expected the system to not favor popular or more exposed items, but recommend unpopular items just as likely as the popular ones to the appropriate users. This debiasing scheme can be used in recommending movies to users with different preferences in terms of the popularity and rating of the user.

In this report, first I will explain probabilistic matrix factorization method and causal inference approach to this method in Section 2. Then, I will demonstrate the results of Liang et. al's causal approach with weighted samples [2] on MovieLens dataset and compare it to unweighted PMF

method, as well as some basic baselines in Section **??**. Lastly, I will display different recommendation modes and hidden gem discovery using different recommendation schemes on top of the causal approach. While doing so, I will demonstrate the results on different datasets and explain how some data analysis and preprocessing methods yield improvements over others.

2. METHODS

2.1. Notation

Notations in this section are as below:

- $u \in U = \{1, 2, ..., N_u\}$: User u demonstrated with an id in U where N_u is number of users.
- $i \in M = \{1, 2, ..., N_m\}$: Item i demonstrated with an id in M where N_m is number of items.
- $\mathcal{O}$: Dataset that contains user-item-rating triplets (u, i, r_{ui})
- $\mathbf{R} \in \mathbb{R}^{U \times M}$: Matrix equivalent of the dataset whose $(r, i)-$th element r_{ui} is the rating (or action) of the user u to item i.
- $\theta_u \in \mathbb{R}^L$: User intrinsic score vector for user u, where L is the number of latent factors.
- $\beta_i \in \mathbb{R}^L$: Item intrinsic score vector for item i.
- p_{ui} : Propensity score of user u for item i.
- λ_θ : Inverse std for zero-mean Gaussian user preference prior model
- λ_β : Inverse std for zero-mean Gaussian item preference prior model

2.2. Model

Although the models introduced in this section can be used in all recommender system applications, for the sake of simplicity, here they will be explained on movie recommender systems where item i is a movie and r_{ui} is the rating of the user u to movie i. Probabilistic matrix factorization [1] assumes a Gaussian model on the observed ratings given user and item intrinsic scores, Gaussian priors on those intrinsic scores, and independence of the observations. Hence, the observation distribution given intrinsic scores is

$$p(\mathbf{R}|\Theta, \mathrm{B}, \sigma^2) = \prod_{u=1}^{N_u} \prod_{i=1}^{N_m} \left[\mathcal{N}(r_{ui}|\theta_u^T \beta_j, \sigma^2)\right]^{I_{ui}} \quad (1)$$

where $\mathcal{N}(x|\mu, \sigma^2)$ is the Gaussian distribution of x with mean mu and variance σ^2, $\Theta \in \mathbb{R}^{L \times N_u}$ and $\mathrm{B} \in \mathbb{R}^{L \times N_m}$ are latent user and movie matrices, whose columns are θ_u and β_i respectively, and I_{ui} indicates (is equal to 1) if the rating of user u to movie i is in the dataset. And prior distributions on intrinsic scores are

$$p(\Theta|\lambda_\theta^{-1}) = \prod_{u=1}^{N_u} \mathcal{N}(\theta_u|0, \lambda_\theta^{-1} I_L) \quad (2)$$

$$p(\mathrm{B}|\lambda_\beta^{-1}) = \prod_{i=1}^{N_m} \mathcal{N}(\beta_i|0, \lambda_\beta^{-1} I_L) \quad (3)$$

and the predictions are formed by

$$\hat{r}_{ui} = \theta_u^T \beta_i \quad (4)$$

From the above definition of the model, it is obvious that one needs to estimate vectors θ_u and β_i in order to make predictions. They can be obtained by maximizing the posterior distribution

$$p(\Theta, \mathrm{B}|\mathbf{R}, \lambda_\theta, \lambda_\beta, \sigma^2) \propto p(\Theta, \mathrm{B}) \prod_{u=1} \prod_{i=1} p(r_{ui}|\theta_u, \beta_i) \quad (5)$$

and since the distributions are assumed to be Gaussian, it is equivalent to minimizing the objective function

$$\mathcal{L} = \sum_{(u,i):(u,i)\in\mathcal{O}} (r_{ui} - \theta_u^T \beta_i)^2 + \lambda_\theta \sum_{u:(u,i)\in\mathcal{O}} \|\theta_u\|^2 + \lambda_\beta \sum_{i:(u,i)\in\mathcal{O}} \|\beta_i\|^2 \quad (6)$$

Note that without loss of generality, λ_θ and λ_β can be scaled by σ^2, so σ^2 is not included here for simplicity.

As it is discussed in Section 1, since this estimation of latent vectors is driven by data samples in $\mathcal{O}$, items exposed more to the users has more influence, hence bias, on the resulting parameter estimations and consecutively recommender system than the less exposed ones. Causal inference method of Liang et. al [2] steps in here. In order to eliminate this influence, they propose to generate an intervention dataset by weighting each user-item pair so that it will downweight the influence of popular items and up-weight the influence of unpopular items. The exposure of a user to an item is modeled as a Bernouilli random variable $Ber(p_i)$ and the propensity score, i.e. weights, can be estimated as the proportion of the users that has rated the movie i

$$p_{ui} = \hat{p}_i = \frac{\text{\# of users watched movie } i}{N_u} \quad (7)$$

The weighting scheme used here depends solely on item popularity. However, it can be generalized to models that include both user and item properties in the exposure model. By weighting the probabilities in Eq. (5) by this propensity score, the new intervened posterior distribution becomes

$$p(\Theta, \mathrm{B}|\mathbf{R}, \lambda_\theta, \lambda_\beta) \propto p(\Theta, \mathrm{B}) \prod_{u=1} \prod_{i=1} p(r_{ui}|\theta_u, \beta_i)^{\frac{1}{p_{ui}}} \quad (8)$$

and the objective function becomes

$$\mathcal{L} = \sum_{(u,i):(u,i)\in\mathcal{O}} \frac{1}{p_{ui}}(r_{ui} - \theta_u^T \beta_i)^2 + \lambda_\theta \sum_{u:(u,i)\in\mathcal{O}} \|\theta_u\|^2 + \lambda_\beta \sum_{i:(u,i)\in\mathcal{O}} \|\beta_i\|^2 \quad (9)$$

and by setting the derivative of the objective function with respect to θ_u and β_i to 0 gives the vector updates as

$$\theta_u \leftarrow \left(\sum_{i:(u,i)\in\mathcal{O}} \frac{1}{p_{ui}} \beta_i \beta_i^T + \lambda_\beta I_K \right)^{-1} \sum_{i:(u,i)\in\mathcal{O}} \frac{1}{p_{ui}} r_{ui} \beta_i \quad (10)$$

$$\beta_i \leftarrow \left(\sum_{u:(u,i)\in\mathcal{O}} \frac{1}{p_{ui}} \theta_u \theta_u^T + \lambda_\theta I_K \right)^{-1} \sum_{u:(u,i)\in\mathcal{O}} \frac{1}{p_{ui}} r_{ui} \theta_u \quad (11)$$

and the complete algorithm for estimating vectors θ_u and β_i can be found in Fig. 2.

Input: Exposed entires in the click matrix $\{r_{ui} : (u,i) \in \mathcal{O}\}$, regularization parameters λ_θ and λ_β
Output: User latent factors $\theta_{1:U}$ and item latent factors $\beta_{1:I}$
Fit the exposure model to compute the propensity score
Randomly initialize $\theta_{1:U}$, $\beta_{1:I}$
while *not converged* **do**
  for $u \leftarrow 1$ ***to*** U **do**
    Update user factor θ_u (Eq. 10)
  end
  for $i \leftarrow 1$ ***to*** I **do**
    Update item factor β_i (Eq. 11)
  end
end
return $\theta_{1:U}$, $\beta_{1:I}$

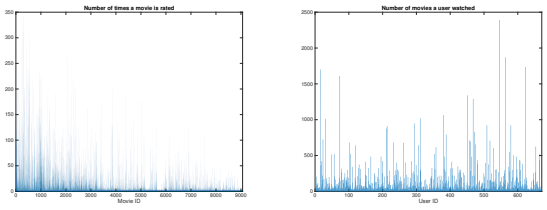
Fig. 2. PMF algortihm with causal inference approach [2]

3. NUMERICAL STUDIES

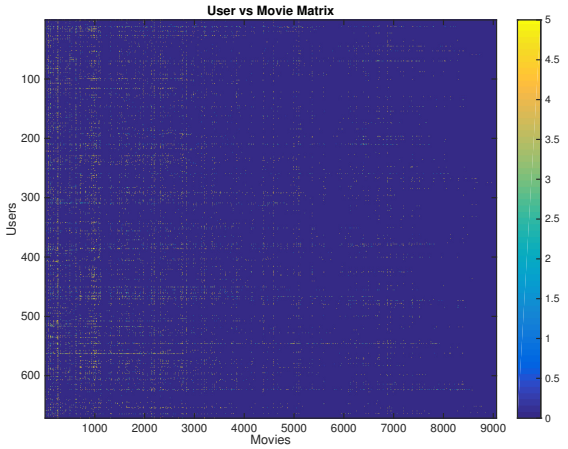
3.1. Dataset

In this project, I first used the latest updated version of MovieLens 100k dataset [4]. It is released in 2016, and it contains 100004 ratings from 671 users on 9066 movies, with sparsity of 1.64%. Plots that show a summary of the dataset can be seen in Fig. 3.

As it can be seen in Fig. 4(a), number of ratings a movie get, which can be thought as popularity, varies heavily. There are some movies that are rated only once. Similarly, there are also some users that rated only one single movie.



(a) Number of users rated a movie (b) Number of movies a user rated



(c) Ratings of users for the movies, **R** matrix

Fig. 3. MovieLens 100k dataset

Then, I used MovieLens 1M dataset, which has 1M ratings from 6040 users on 3706 movies with sparsity of 4.47%. One critical difference between two datasets is 1M dataset has more users than movies, i.e. the data matrix to be completed is a thin matrix.

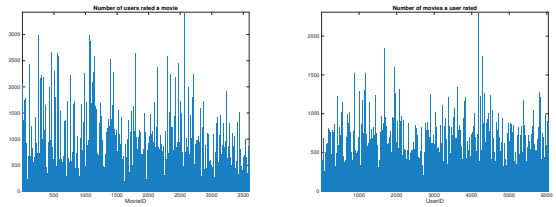
3.2. Data Analysis

In their paper, Liang et. al [2] used binary predictions for their model. In movie recommender systems, this binary representation corresponds to whether a user liked a movie or not. In this project, I applied a threshold on the rating to determine if a user liked a movie or not,

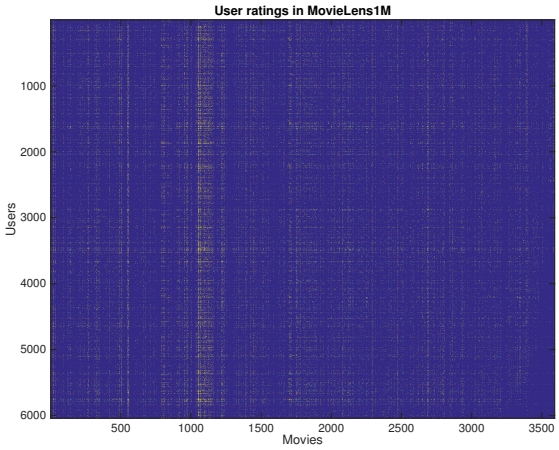
$$r_{ui} = \begin{cases} 1 & \text{if } r_{ui} > 3 \\ 0 & \text{if } r_{ui} \leq 3 \end{cases}$$

and the histogram of the data labels before and after applying the threshold for both datasets can be seen in Fig. 5.

When I apply probabilistic matrix factorization, I divided the datasets into three parts as training, validation and test. One critical issue to consider was how to divide datasets into these three appropriately. First, I blindly divided them as the order they appear in the dataset, which is ordered by first the user id's and then movie id's within users. By doing so, I happened to divide the datasets into three so that each subset



(a) Number of users rated a movie (b) Number of movies a user rated



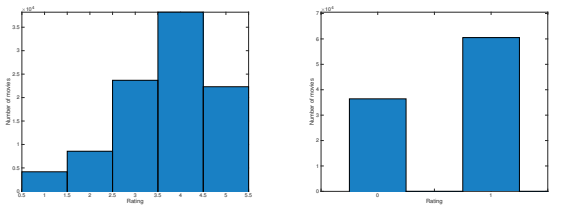
(c) Ratings of users for the movies, **R** matrix

Fig. 4. MovieLens 1M dataset

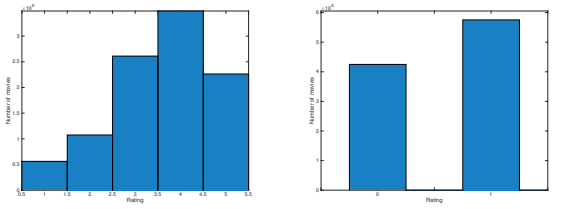
consist of different subjects, and some movies in the validation and test sets are never seen in the training set. This corresponds to completely empty rows (or columns) in **R** matrix in the matrix completion problem, and it is not possible to fill in completely empty rows or column. Since it was not possible for the system to recommend movies it is not aware of to the users it did not encounter, this affected the accuracy. So, I divided the dataset more wisely, by making sure each movie in the validation and test sets is also encountered in training set, and also each user appears in the training set if she is present in the validation and test sets. Different divisions of the datasets can be seen in Fig. 6.

3.3. Experiment Results

In this section, I compare results of probabilistic matrix factorization with and without causal approach on MovieLens 100k and 1M datasets. For probabilistic matrix factorization, I used the Matlab code provided with the paper [1], and modified it heavily in order to apply causal approach. In the attached code files, `pmf_original.m` is the original probabilistic matrix factorization code provided by [1] with some small modifications for printing and plotting the results. `pmf_causal.m` is the code that implements causal inference for recommendations, and also computes and displays movie recommendations. In order to keep consistency, although the



(a) Ratings-100k Dataset (b) Binary ratings-100k Dataset



(c) Ratings-1M Dataset (d) Binary ratings-1M Dataset

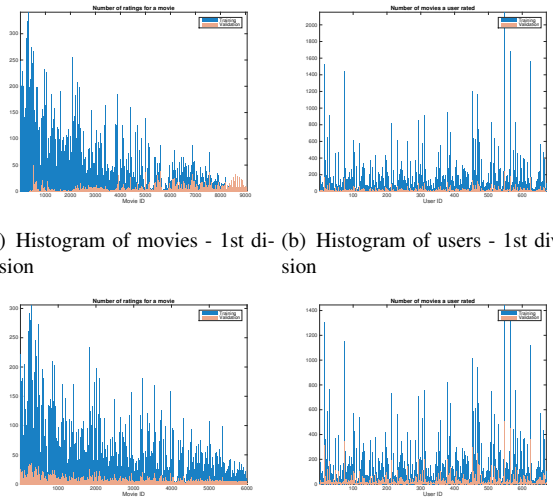
Fig. 5. Ratings before and after applying threshold in both datasets

notation and most of the variables are the same with the previous one, I wrote most of this part from scratch on my own.

As evaluation metric, root mean square error (RMSE) is used. Since this is a binary prediction problem, RMSE is highly correlated with accuracy. In order to find the best performance of each method, a grid search is performed over experiment parameters (learning rate (LR), momentum (mom), regularization coefficient (λ) and number of latent factors (L) for noncausal PMF and inverse prior variances ($\lambda = \lambda_\theta = \lambda_\beta$) and number of latent factors (L) for causal PMF), and the results of two different recommendation methods (NC: noncausal PMF, C: causal PMF) are compared on two datasets (MovieLens 100k and MovieLens 1M). As it can be seen from this comparison, introducing causal approach decreases RMSE and hence increases accuracy for both datasets. When training errors are compared, it is seen that noncausal approach has smaller training error compared

		MovieLens 100k		MovieLens 1M	
		NC	C	NC	C
Optimum Parameters	LR	10	NA	50	NA
	Mom	0.8	NA	0.8	NA
	λ	0.001	100	0.01	50
	L	30	30	30	30
Runtime		17 s	90 s	90 s	25 m
RMSE	Training	0.3473	0.4152	0.3757	0.5440
	Test	0.6363	0.5301	0.6273	0.5780

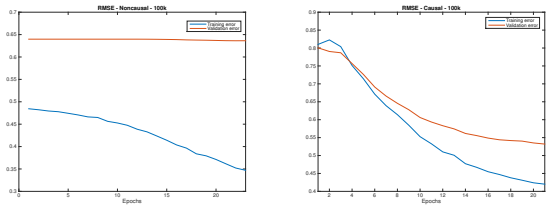
Table 1. Comparison of different methods (NC: noncausal PMF, C: causal PMF) on MovieLens 100k and MovieLens 1M datasets



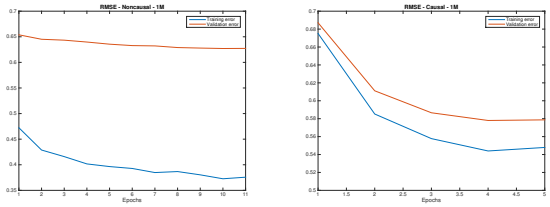
(a) Histogram of movies - 1st division (b) Histogram of users - 1st division

(c) Histogram of movies - 2nd division (d) Histogram of users - 2nd division

Fig. 6. Ratings before and after applying threshold in both datasets



(a) RMSE of NC on 100k (b) RMSE of C on 100k



(c) RMSE of NC on 1M (d) RMSE of C on 1M

Fig. 7. Training and validation RMSE over epochs of causal (C) and noncausal (NC) methods on 100k and 1M data

to causal approach, although its test error is larger. One possible explanation to this is that noncausal approach overfits the data, which makes sense when we think about the influence of the frequent data points discussed as in Section 1.

Training and validation errors over epochs can be seen in Fig. 7. For each simulation, vector updates are performed until the validation error starts to increase. Thus, simulations reached their optimum performance at different epochs. Since it inculdes matrix inversion (Eq. 10), causal approach takes more time compared to noncausal PMF approach but it gives much better results (Table 1).

3.4. Hidden Gem Discovery

In Liang et. al's work [2], a causal approach with probabilistic matrix factorization is presented with a binary rating scheme. In this section, I used their method not in a binary setting, but with ratings from 1 to 5, which means not applying the threshold function in Eq. 3.2, and I took advantage of the popularity score calculated in previous section (Eq. 7). After predicting the ratings, instead of presenting the user the movie with highest predicted rating, I generated three different recommendation modes, namely "Popular recommendation", "Hidden gem recommendation" and "Highest rating recommendation".

A comparison of these three recommendation modes for a randomly selected user can be seen in Table 2. In popular recommendation mode, although the rating of the user for that movie is less than others, he is recommended *Cast Away* because apparently (I checked number of votes manually on IMDb website) it is a popular movie. So, by sacrificing a little bit from the taste, the user gets to see a very popular movie. If he chooses to see a hidden gem, a less-known but still likable movie, he is recommended *Return to Me*. Again, checking its IMDb votes, it seems like it is a less known movie indeed, and by looking at the user's rating, it seems user did indeed enjoyed the movie. And finally, if the user wants to see the movie he would like the best, he is recommended *Man on the Moon*, and since he gave 5 to this movie, we can again say that the recommender system did a good job by recommending a movie in alignment with user's preference.

4. CONCLUSION

In this project, first I implemented a causal approach presented in [2] with probabilistic matrix factorization [1] on MovieLens data for movie recommendation. Thanks to this causal approach, the bias occured from the user exposure of the movies could be eliminated by weighting the data samples by the popularity of the movies. I implemented both causal and noncausal approaches on MovieLens 100k and MovieLens 1M datasets. Although causal approach took more time than the noncausal approach, more accurate recommendations could be made (Table with causal approach 1). Then, I generated a recommender system with three different recommendation modes, namely "Popular recommendation", "Hidden gem recommendation" and "Highest rating recommendation". Based on user's preference, I weighted movies to be recommended with their popularity appropriately and recommended the movie that has the best score in this weighted scheme. As a result, comparing the recommended movies with user's ratings and IMDb votes, recommender system appeared to be performing in alignment with user's discovery preferences.

	Recommendation	User rating	# of IMDb votes	IMDb rating
Popular	Cast Away	3	417216	7.7
Hidden Gem	Return to Me	4	16978	6.9
Highest	Man on the Moon	5	103797	7.4

Table 2. Different recommendation modes

5. REFERENCES

[1] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization," in *Proceedings of the 20th International Conference on Neural Information Processing Systems*, USA, 2007, NIPS'07, pp. 1257–1264, Curran Associates Inc.

[2] D. Liang, L. Charlin, and D. M. Blei, "Causal inference for recommendation," in *Workshop on Causation: Foundation to Application, 32nd Conference on Uncertainty in Artificial Intelligence*, 2016.

[3] T. Schnabel, A. Swaminathan, A. Singh, N. Chandak, and T. Joachims, "Recommendations as treatments: Debiasing learning and evaluation," in *Proceedings of The 33rd International Conference on Machine Learning*, Maria Florina Balcan and Kilian Q. Weinberger, Eds., New York, New York, USA, 20–22 Jun 2016, vol. 48 of *Proceedings of Machine Learning Research*, pp. 1670–1679, PMLR.

[4] F. Maxwell Harper and Joseph A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 19:1–19:19, Dec. 2015.

Appendix: Code

pmf_causal.m

```
clear all;
tic
for num_feat = 30;%[10, 30, 50]
    for lambda = 100;%[0.1, 1, 10, 100]
        for causal = 1%0:1
          epoch=1;
          maxepoch=20;
          load moviedata100k_movies % Triplets: {user_id, movie_id, rating}
          mean_rating = mean(train_vec(:,3));
          pairs_tr = length(train_vec); % training data
          pairs_pr = length(probe_vec); % validation data

          N=50000; % number of training triplets per batch
          numbatches = floor(pairs_tr/N); % Number of batches
          num_m = length(unique([train_vec(:,2); probe_vec(:,2)]));  % Number of movies
          num_p = length(unique([train_vec(:,1); probe_vec(:,1)]));  % Number of users

          w1_M1     = randn(num_m, num_feat+1); % Movie feature vectors
          w1_P1     = randn(num_p, num_feat+1); % User feature vecators
          if causal == 1
              w1_M1 = w1_M1*sqrt(1/lambda);
              w1_P1 = w1_P1*sqrt(1/lambda);
          end
          w1_M1_inc = zeros(num_m, num_feat+1);
          w1_P1_inc = zeros(num_p, num_feat+1);
          p_score = zeros(num_m,num_p);
          items = unique(train_vec(:,2));
          p_score(items,:) = repmat(arrayfun(@(i)sum(train_vec(:,2) == i), items), 1, num_p);
          p_score = p_score/num_p;
        epoch = 1;
        flag = true;
        f = [];
        err_train_batch = [];
            while flag
              rr = randperm(pairs_tr);
              train_vec = train_vec(rr,:);
              clear rr
              for batch = 1:numbatches

                aa_p   = double(train_vec((batch-1)*N+1:batch*N,1));
                aa_m   = double(train_vec((batch-1)*N+1:batch*N,2));
                rating = double(train_vec((batch-1)*N+1:batch*N,3));
                rating = (rating > 3);

                %%%%%%%%%%%%%% Compute Predictions %%%%%%%%%%%%%%%%%%
                pred_out = sum(w1_M1(aa_m,:).*w1_P1(aa_p,:),2);
                vec_ind = sub2ind(size(p_score), aa_m, aa_p);
                p_score_vec = p_score(vec_ind);
                w1_P1_old = w1_P1;
                for u = 1:num_p
                    if sum(aa_p == u) > 0
                    idx_mu = aa_m(aa_p == u);
                    out_P1 = bsxfun(@times, permute(w1_M1(idx_mu,:), [2 3 1]), ...
                        permute(w1_M1(idx_mu,:), [3 2 1 ]));
                    V_P1 = sum(bsxfun(@times, out_P1, reshape(1./p_score(idx_mu, ...
                        u),1,1,numel(p_score(idx_mu, u)))), 3) + lambda*eye(num_feat+1);
                    Y_P1 = sum(((rating(aa_p == u)./p_score(idx_mu, ...
                        u))*ones(1,num_feat+1)).*w1_M1(idx_mu,:), 1)';
                    w1_P1(u, :) = (inv(V_P1)*Y_P1)';
                    end
                end
                for i = 1:num_m
                    if sum(aa_m == i) > 0
```

```
                    idx_pm = aa_p(aa_m == i);
                    out_M1 = bsxfun(@times, permute(w1_P1_old(idx_pm,:), [2 3 1]), ...
                        permute(w1_P1_old(idx_pm,:), [3 2 1 ]));
                    V_M1 = sum(bsxfun(@times, out_M1, reshape(1./p_score(i, ...
                        idx_pm),1,1,numel(p_score(i, idx_pm)))), 3) + lambda*eye(num_feat+1);
                    Y_M1 = sum(((rating(aa_m == i)./p_score(i, ...
                        idx_pm)')*ones(1,num_feat+1)).*w1_P1_old(idx_pm,:), 1)';
                    w1_M1(i, :) = (inv(V_M1)*Y_M1)';
                    end
                end
                if causal == 1
                f(end+1) = sum( (1./p_score_vec).*(pred_out - rating).^2 + ...
                    0.5*lambda*( sum( (w1_M1(aa_m,:).^2 + w1_P1(aa_p,:).^2),2)));
                else
                    f(end+1) = sum( (pred_out - rating).^2 + ...
                    0.5*lambda*( sum( (w1_M1(aa_m,:).^2 + w1_P1(aa_p,:).^2),2)));
                end
                err_train_batch = [err_train_batch sqrt(sum((pred_out- ...
                    rating).^2)/length(pred_out));];
            end
            %%%%%%%%%%%%%% Compute Predictions after Parameter Updates %%%%%%%%%%%%%%%%%%
            pred_out = sum(w1_M1(aa_m,:).*w1_P1(aa_p,:),2);
            sum(pred_out>0 == rating)/length(pred_out)
            if causal == 1
            f_s = sum( (1./p_score_vec).*(pred_out - rating).^2 + ...
                  0.5*lambda*( sum( (w1_M1(aa_m,:).^2 + w1_P1(aa_p,:).^2),2)));
            else
            f_s = sum( (pred_out - rating).^2 + ...
                  0.5*lambda*( sum( (w1_M1(aa_m,:).^2 + w1_P1(aa_p,:).^2),2)));
            end
            err_train_loss(epoch) = sqrt(f_s/N);
            err_train(epoch) = sqrt(sum((pred_out- rating).^2)/length(pred_out));
            %%% Compute predictions on the validation set %%%%%%%%%%%%%%%%%%%%%%%
            NN=pairs_pr;

            aa_p = double(probe_vec(:,1));
            aa_m = double(probe_vec(:,2));
            rating = double(probe_vec(:,3));
            rating = (rating > 3);

            pred_out = sum(w1_M1(aa_m,:).*w1_P1(aa_p,:),2);
            err_valid(epoch) = sqrt(sum((pred_out- rating).^2)/NN);
            ff = find(pred_out>0); pred_out(ff)=1; % Clip predictions
            ff = find(pred_out<=0); pred_out(ff)=0;
            if ( epoch > 1 && err_valid(epoch-1) < err_valid(epoch)) || epoch > maxepoch || ...
                isnan(err_valid(end))
                %
                flag = false;
            else
                epoch = epoch + 1;
            end
        end
    end
end
toc
%%
figure,plot(1:length(err_train), err_train, 1:length(err_train), err_valid);
xlabel('Epochs');legend('Training error', 'Validation error');
figure,plot(1:length(f), f);xlabel('Epochs');
%%
%%%%%% Recommend one movie to users (pairs that are present in valid dataset) %%%%%%%%%%%%%%%%%%%%%%%
present_m = probe_vec(:,2);
present_p = probe_vec(:,1);
rec_user = [];
for random_user = randi(num_p) % 1:num_p
missing_m_user = present_m(present_p == random_user);
pred_user = w1_M1(missing_m_user,:)*w1_P1(random_user,:)' + mean_rating;
```

```
pred_user(pred_user>5)=5; % Clip predictions
pred_user(pred_user<1)=1;
idx_user = sub2ind(size(p_score), missing_m_user, random_user*ones(size(missing_m_user)));
[¬, idx_gem_user] = max(pred_user.*(1./p_score(idx_user)));
[¬, idx_pop_user] = max(pred_user.*p_score(idx_user));
[¬, idx_rat_user] = max(pred_user);
rec_user = [rec_user; random_user, missing_m_user(idx_pop_user), missing_m_user(idx_gem_user), ...
    missing_m_user(idx_rat_user)];
end
%% %% Display random recommendations   %%%%%%%
movies = readtable('movies.csv');
randu = random_user;
fprintf('For user %d \n Popular recommendation: %s %.1f \nHidden gem recommendation: %s ...
    %.1f\nHighest rating recommendation: %s %.1f\n',...
    randu, movies{rec_user(1, 2), 2}{1}, probe_vec(probe_vec(:,1) == randu & probe_vec(:,2) == ...
        rec_user(1,2), 3),...
           movies{rec_user(1, 3), 2}{1}, probe_vec(probe_vec(:,1) == randu & probe_vec(:,2) == ...
               rec_user(1,3), 3),...
           movies{rec_user(1, 4), 2}{1}, probe_vec(probe_vec(:,1) == randu & probe_vec(:,2) == ...
               rec_user(1,4), 3));
```

pmf_original.m

```
% Version 1.000
%
% Code provided by Ruslan Salakhutdinov
%
% Permission is granted for anyone to copy, use, modify, or distribute this
% program and accompanying programs and documents for any purpose, provided
% this copyright notice is retained and prominently displayed, along with
% a note saying that the original programs are available from our
% web page.
% The programs and documents are distributed without any warranty, express or
% implied.  As the programs were written for research purposes only, they have
% not been tested to the degree that would be advisable in any important
% application.  All use of these programs is entirely at the user's own risk.
tic
clear all;close all;
rand('state',0);
randn('state',0);
restart = 1;
if restart==1
  restart=0;
  epsilon=10; % Learning rate
  lambda  = 0.0001; % Regularization parameter
  momentum=0.8;
  epoch=1;
  maxepoch=50;

  load moviedata100k_movies % Triplets: {user_id, movie_id, rating}
  mean_rating = mean(train_vec(:,3));

  pairs_tr = length(train_vec); % training data
  pairs_pr = length(probe_vec); % validation data
  N=10000; % number training triplets per batch

  numbatches = floor(pairs_tr/N); % Number of batches
  num_m = length(unique([train_vec(:,2); probe_vec(:,2)]));  % Number of movies
  num_p = length(unique([train_vec(:,1); probe_vec(:,1)]));  % Number of users
  num_feat = 30; % Rank 10 decomposition

  w1_M1     = 0.1*randn(num_m, num_feat); % Movie feature vectors
  w1_P1     = 0.1*randn(num_p, num_feat); % User feature vecators
  w1_M1_inc = zeros(num_m, num_feat);
  w1_P1_inc = zeros(num_p, num_feat);
end
for epoch = epoch:maxepoch
```

```
  rr = randperm(pairs_tr);
  train_vec = train_vec(rr,:);
  clear rr
  for batch = 1:numbatches
    fprintf(1,'epoch %d batch %d \r',epoch,batch);

    aa_p   = double(train_vec((batch-1)*N+1:batch*N,1));
    aa_m   = double(train_vec((batch-1)*N+1:batch*N,2));
    rating = double(train_vec((batch-1)*N+1:batch*N,3));
    rating = rating > 3;
    mean_rating = mean(rating);
    rating = rating-mean_rating; % Default prediction is the mean rating.
    %%%%%%%%%%%%%% Compute Predictions %%%%%%%%%%%%%%%%%%
    pred_out = sum(w1_M1(aa_m,:).*w1_P1(aa_p,:),2);
    f(epoch) = sum( (pred_out - rating).^2 + ...
        0.5*lambda*( sum( (w1_M1(aa_m,:).^2 + w1_P1(aa_p,:).^2),2)));
    %%%%%%%%%%%%%% Compute Gradients %%%%%%%%%%%%%%%%%%%
    IO = repmat(2*(pred_out - rating),1,num_feat);
    Ix_m=IO.*w1_P1(aa_p,:) + lambda*w1_M1(aa_m,:);
    Ix_p=IO.*w1_M1(aa_m,:) + lambda*w1_P1(aa_p,:);

    dw1_M1 = zeros(num_m,num_feat);
    dw1_P1 = zeros(num_p,num_feat);
    for ii=1:N
      dw1_M1(aa_m(ii),:) =  dw1_M1(aa_m(ii),:) +  Ix_m(ii,:);
      dw1_P1(aa_p(ii),:) =  dw1_P1(aa_p(ii),:) +  Ix_p(ii,:);
    end
    %%%% Update movie and user features %%%%%%%%%%%
    w1_M1_inc = momentum*w1_M1_inc + epsilon*dw1_M1/N;
    w1_M1 =  w1_M1 - w1_M1_inc;
    w1_P1_inc = momentum*w1_P1_inc + epsilon*dw1_P1/N;
    w1_P1 =  w1_P1 - w1_P1_inc;
  end
  %%%%%%%%%%%%%% Compute Predictions after Paramete Updates %%%%%%%%%%%%%%%%%%
  pred_out = sum(w1_M1(aa_m,:).*w1_P1(aa_p,:),2);
  f_s(epoch) = sum( (pred_out - rating).^2 + ...
        0.5*lambda*( sum( (w1_M1(aa_m,:).^2 + w1_P1(aa_p,:).^2),2)));
  err_train(epoch) = sqrt(f_s(epoch)/N);
  %%% Compute predictions on the validation set %%%%%%%%%%%%%%%%%%%%%%
  NN=pairs_pr;

  aa_p = double(probe_vec(:,1));
  aa_m = double(probe_vec(:,2));
  rating = double(probe_vec(:,3));
  rating = rating > 3;

  pred_out = sum(w1_M1(aa_m,:).*w1_P1(aa_p,:),2) + mean_rating;
  ff = find(pred_out>0); pred_out(ff)=1; % Clip predictions
  ff = find(pred_out<=0); pred_out(ff)=0;

  err_valid(epoch) = sqrt(sum((pred_out- rating).^2)/NN);
  fprintf(1, 'epoch %4i batch %4i Training RMSE %6.4f  Test RMSE %6.4f  \n', ...
              epoch, batch, err_train(epoch), err_valid(epoch));
  if ( epoch > 1 && err_valid(epoch-1) < err_valid(epoch))
      break;
  end
end
toc
%%
figure,plot(1:length(err_train), err_train, 1:length(err_train), err_valid);
xlabel('Epochs');title(sprintf('Error - Lr=%.4f, lambda=%.4f, momentum=%.4f, num_feat=%d', ...
epsilon, lambda, momentum, num_feat));
legend('Training error', 'Validation error');
figure,plot(1:length(f), f);
xlabel('Epochs');title(sprintf('Loss - Lr=%.4f, lambda=%.4f, momentum=%.4f, num_feat=%d', ...
epsilon, lambda, momentum, num_feat));
```

makematrix.m

```
% Version 1.000
%
% Code provided by Ruslan Salakhutdinov

%% Create a matrix of size num_p by num_m from triplets {user_id, movie_id, rating_id}
load moviedata1M_movies
num_m = length(unique([train_vec(:,2); probe_vec(:,2)]));  % Number of movies
num_p = length(unique([train_vec(:,1); probe_vec(:,1)]));  % Number of users
count = zeros(num_p,num_m,'single'); %for Netflida data, use sparse matrix instead.
for mm=1:num_m
 ff= find(train_vec(:,2)==mm);
 count(train_vec(ff,1),mm) = train_vec(ff,3);
end
```

divide_dataset.m

```
function divide_dataset()
data = csvread('ratings1M.csv', 1, 0);
users = unique(data(:,1));
movies = unique(data(:,2));
ratings = data(:,3);
data(:,3) = ratings;
train_vec = [];
probe_vec = [];
m_idx = 1;
%% USERS
for m = movies'
    data(data(:,2) == m, 2) = m_idx;
    m_idx = m_idx+1;
end
for u = users'
    mr_u = data(data(:,1) == u, 2:3); % movie, rating pairs of user u
    nm_u = size(mr_u,1); % number of movies user u rated
    idx = floor(nm_u/10);
    train_vec = [train_vec; [u*ones(nm_u-idx,1) mr_u(1:nm_u-idx, :)]];
    probe_vec = [probe_vec; [u*ones(idx,1) mr_u(nm_u-idx+1:nm_u, :)]];
end
%% Movies
for m = movies'
    if sum(data(:,2) == m) > 1
    data(data(:,2) == m, 2) = m_idx;
    m_idx = m_idx+1;
    else
    data(data(:,2) == m, :) = [];
    end
end
users = unique(data(:,1));
movies = unique(data(:,2));
ratings = data(:,3);
for m = movies'
    n_m = sum(data(:,2) == m);
    if n_m < 10
        n_val_m = floor(n_m / 2);
    else
        n_val_m = ceil(n_m / 10);
    end
    idx_m = find(data(:,2) == m);
    [val_idx_m, val_idx_m_idx] = datasample(idx_m, n_val_m, 'Replace', false);
    train_idx_m = ones(n_m,1);
    train_idx_m(val_idx_m_idx) = 0;
    probe_vec = [probe_vec; [data(val_idx_m, 1:3)]];
    train_vec = [train_vec; [data(idx_m(train_idx_m==1), 1:3)]];
end
save('moviedata1M_movies.mat', 'train_vec', 'probe_vec');
```

movie_data_processing.m

```
load moviedata100k_movies
num_m = length(unique([train_vec(:,2); probe_vec(:,2)]));  % Number of movies
num_p = length(unique([train_vec(:,1); probe_vec(:,1)]));  % Number of users
figure,histogram([train_vec(:,1); probe_vec(:,1)], num_p, 'EdgeColor', 'none', 'FaceAlpha', ...
    0.9);axis tight;title('Number of movies a user rated');xlabel('UserID');
figure,histogram([train_vec(:,2); probe_vec(:,2)], num_m, 'EdgeColor', 'none', 'FaceAlpha', ...
    0.9);axis tight;title('Number of users rated a movie');xlabel('MovieID');
figure,histogram(train_vec(:,2), num_m, 'EdgeColor', 'none', 'FaceAlpha', 0.9);hold ...
    on;histogram(probe_vec(:,2), num_m, 'EdgeColor', 'none', 'FaceAlpha', 0.5);
title('Number of ratings for a movie');xlabel('Movie ID');legend('Training', 'Validation');axis ...
    tight;
figure,histogram(train_vec(:,1), num_p, 'EdgeColor', 'none', 'FaceAlpha', 0.9);hold ...
    on;histogram(probe_vec(:,1), num_p, 'EdgeColor', 'none', 'FaceAlpha', 0.5);
title('Number of movies a user rated');xlabel('User ID');legend('Training', 'Validation');axis tight;
figure,histogram([train_vec(:,3); probe_vec(:,3)], 0.5:1:5.5, 'FaceAlpha', 0.9);axis ...
    tight;ylabel('Number of movies');xlabel('Rating');
figure,histogram(([train_vec(:,3); probe_vec(:,3)] > 3)*1, -0.25:0.5:1.75, 'FaceAlpha', 0.9);axis ...
    tight;ylabel('Number of movies');xlabel('Rating');
```