PAIR-WISE MAXIMUM MARGIN MATRIX FACTORIZATION

Submitted in partial fulfilment of the requirements of the degree of

(Bachelor of Technology)
by

Satya Avinash Gamidi (177118)
Pottanigari Nikhil Reddy (177146)
Vamshi Krishna Korra (177130)

Supervisor (s):

Dr. Venkateswara Rao Kagita
Associate Professor,Department of CSE, NIT Warangal

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL

(2021)

APPROVAL SHEET

This Project Work entitled **Pair-Wise Maximum Matrix Factorization** by **Satya Avinash, Nikhil Reddy, Vamshi Krishna** is approved for the degree of Bachelor of Technology in Computer Science and Engineering at National Institute of Technology Warangal during the year 2020-2021

Examiners

____

____

____

Supervisor (s)

[illegible]

____

Dr. Venkateswara Rao Kagita
Associate Professor, CSE Department

Chairman

____

Dr. P. Radha Krishna
Head of Department, CSE
NIT Warangal

Date: 12/05/2021

Place: Warangal

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/ data/ fact/ source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

G.S.Avinash

(Signature)
Gamidi Satya Avinash
177118

[illegible]

(Signature)
Pottanigari Nikhil Reddy
177146

[illegible]

(Signature)
Vamshi Krishna Korra
177130

Date: 12/05/2021

NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Project work entitled **“PAIR-WISE MAXIMUM MARGIN MATRIX FACTORIZATION”** is a bonafide record of work carried out by Satya Avinash Gamidi (177118), Pottanigari Nikhil Reddy (177146), Vamshi Krishna Korra (177130), submitted to the faculty of Computer Science and Engineering Department in fulfillment of the requirements for the award of the degree of
Bachelor of Technology in Computer Science and Engineering at National Institute of Technology, Warangal during the academic year 2020-2021.

Dr. Venkateswara Rao Kagita
Project Guide
Department of CSE
NIT Warangal

Dr. P. Radha Krishna
Head of Department
Department of CSE
NIT Warangal

ACKNOWLEDGEMENT

We consider it as a great privilege to express our deep gratitude to many respected personalities who guided, inspired and helped us in the successful completion of our project.

We would like to express our deepest gratitude to our guide **Dr. Venkateswara Rao Kagita** and **Dr. Vikas Kumar,** Department of Computer Science and Engineering, National Institute of Technology,Warangal, for their constant supervision, guidance, suggestions and invaluable encouragement during the project.They have been a constant source of inspiration and helped in each stage.

We are grateful to **Dr. P. Radha Krishna**, Head of Department ,Computer Science and Engineering, National Institute of Technology,Warangal for his moral support to carry out this project.

We are very thankful to the project evaluation committee for the strenuous efforts to evaluate our project.

We wish to thank all the staff members of the department for their kind cooperation and support given throughout our project work.We are also thankful to all our friends who have given their valuable suggestions and help in all stages of the development of the project.

Satya Avinash Gamidi (177118)

Pottanigari Nikhil Reddy (177146)

Vamshi Krishna Korra (177130)

ABSTRACT

Maximum Margin Matrix Factorization (MMMF) has been a successful learning method in collaborative filtering research. For a partially observed ordinal rating matrix, the focus is on determining low-norm latent factor matrices U (of users) and V (of items) so as to simultaneously approximate the observed entries under some loss measure and predict the unobserved entries. When the rating matrix contains only two levels (±1), rows of V can be viewed as points in k-dimensional space and rows of U as decision hyperplanes in this space separating +1 entries from −1 entries. When hinge/smooth hinge loss is the loss function, the hyperplanes act as maximum-margin separators. In MMMF, rating matrix with multiple discrete values is treated by specially extending hinge loss function to suit multiple levels. We view this process as analogous to extending a two-class classifier to a unified multi-class classifier. Alternatively, multi-class classifiers can be built by arranging multiple two-class classifiers in a hierarchical manner.

In this paper, we investigate this aspect of collaborative filtering and propose a framework of multiple bi-level MMMFs in a hierarchical manner. The previous approach was based on dividing points by line. We want to consider only the distance between every pair of points, maximizing distance (not considering line division).

Then we compare our proposed method with HMF ,IB, SVD, PMF, MCoC, DsRec , PMMMF , Hern, SCC and TyCo with a benchmark 100k movielen dataset on measures on MAE.

CONTENTS

List of Figures

CHAPTER 1
INTRODUCTION

1.1 Recommender systems

A recommender system or a recommendation system is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item.

Recommender systems are so common now that many of us use them without even knowing it. Because we can't possibly look through all the products or content on a website, a recommendation system plays an important role in helping us have a better user experience, while also exposing us to more inventory we might not discover otherwise.

Some examples of recommender systems in action include product recommendations on Amazon, Netflix suggestions for movies and TV shows in your feed, recommended videos on YouTube, music on Spotify, the Facebook newsfeed and Google Ads.

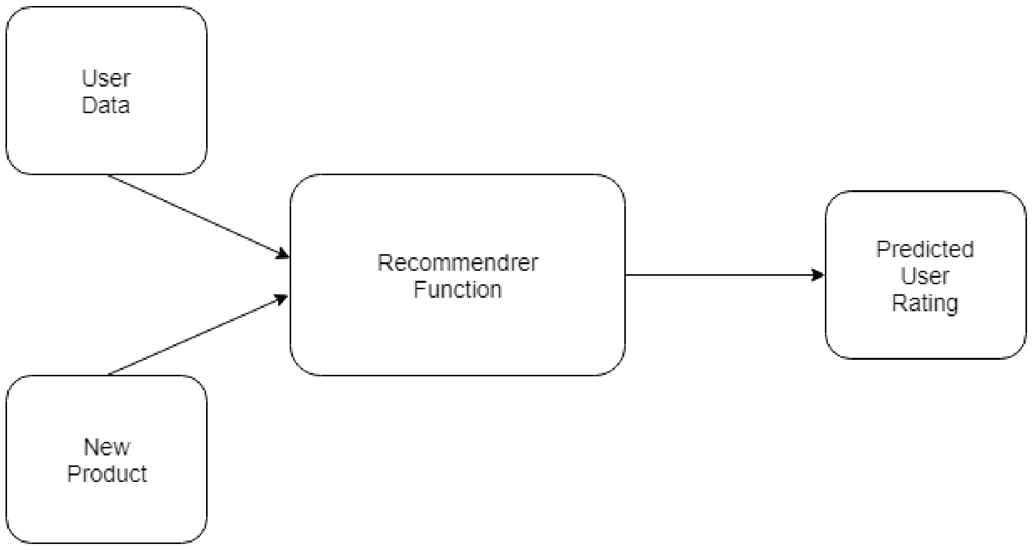


Fig: 1.1 Recommender system

Important component of any of these systems is the recommender function, which takes information about the user and predicts the rating that user might assign to a product, for example. Predicting user ratings, even before the user has actually provided one, makes recommender systems a powerful tool.

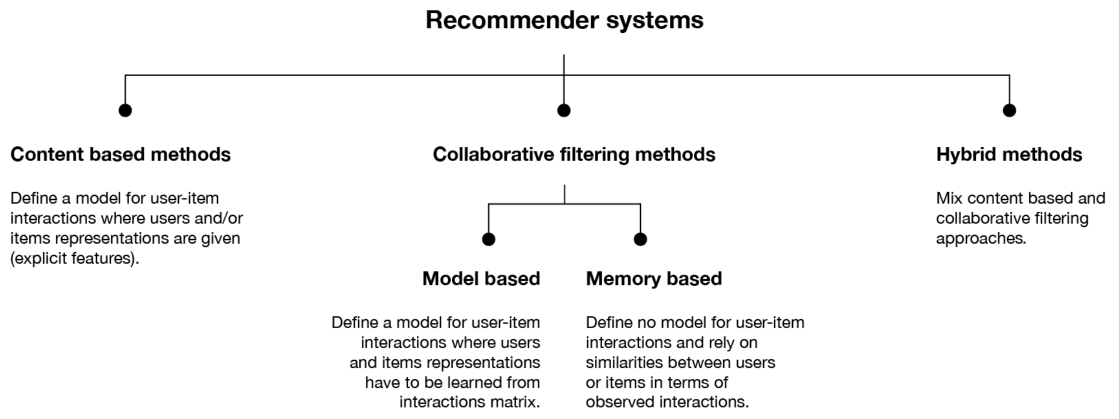

Recommender systems
Content based methods
Define a model for user-item interactions where users and/or items representations are given (explicit features).
Collaborative filtering methods
Hybrid methods
Mix content based and collaborative filtering approaches.
Model based
Define a model for user-item interactions where users and items representations have to be learned from interactions matrix.
Memory based
Define no model for user-item interactions and rely on similarities between users or items in terms of observed interactions.

Fig: 1.2 Classification of recommendation system

1.2 Different types of recommender systems

1. Content Based recommender systems
2. Collaborative Filtering based recommender systems
3. Hybrid based recommender systems (Combination of above both).

1.3 Content Based Recommender systems

The idea of content based methods is to try to build a model, based on the available "features", that explain the observed user-item interactions.
Content based methods suffer far less from the cold start problem than collaborative approaches: new users or items can be described by their characteristics (content) and so relevant suggestions can be done for these new entities.Example is to predict using additional information like age,gender..etc to recommend a movie

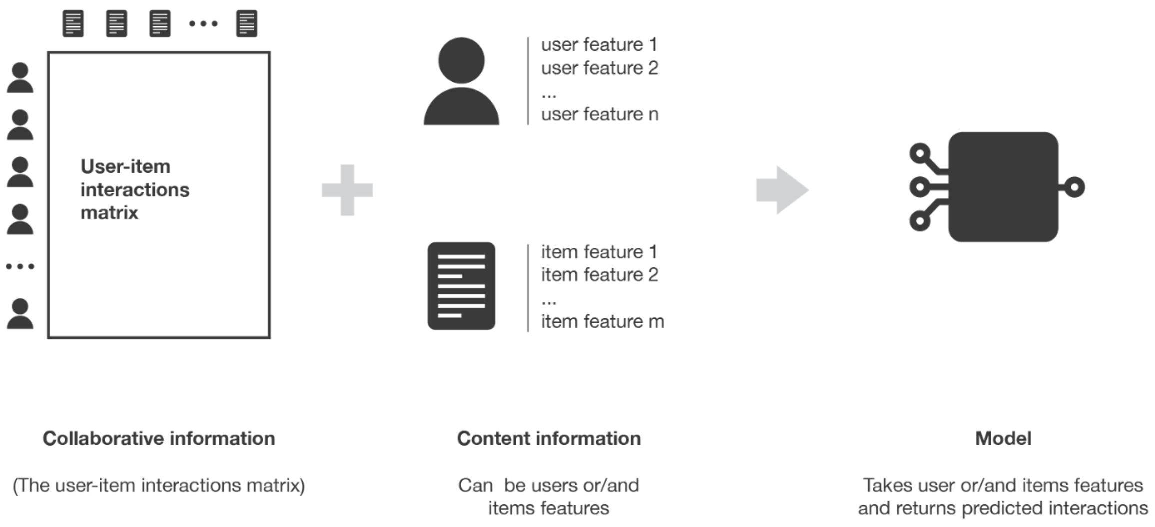
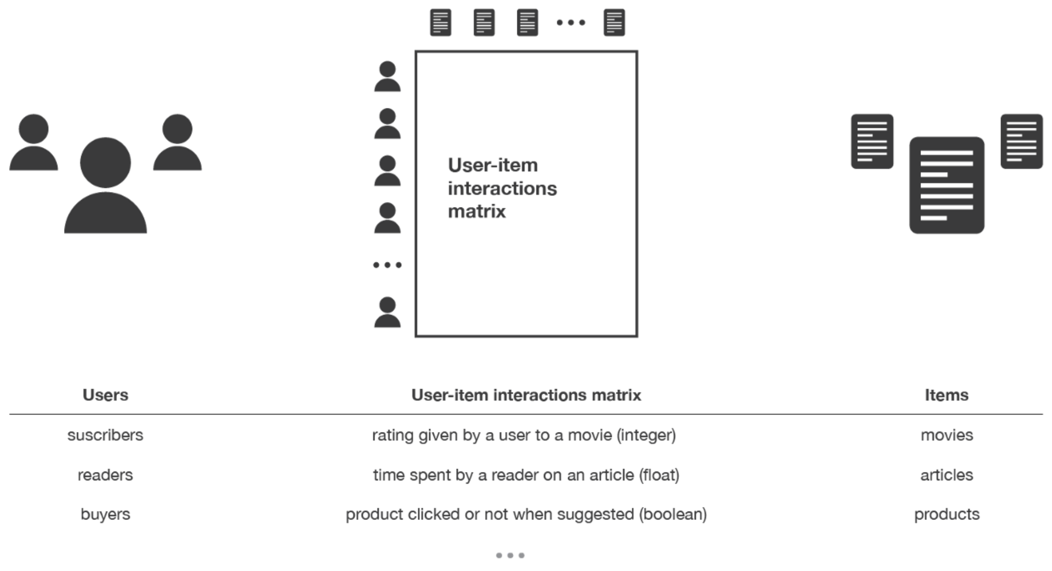

User-item interactions matrix
user feature 1
user feature 2
...
user feature n
item feature 1
item feature 2
...
item feature m
Collaborative information
(The user-item interactions matrix)
Content information
Can be users or/and items features
Model
Takes user or/and items features and returns predicted interactions

Fig: 1.3 Content Based Recommender model

1.4 Collaborative Filtering Recommender systems

Collaborative methods for recommender systems are methods that are based solely on the past interactions recorded between users and items in order to produce new recommendations. These interactions are stored in the so-called "user-item interactions matrix".



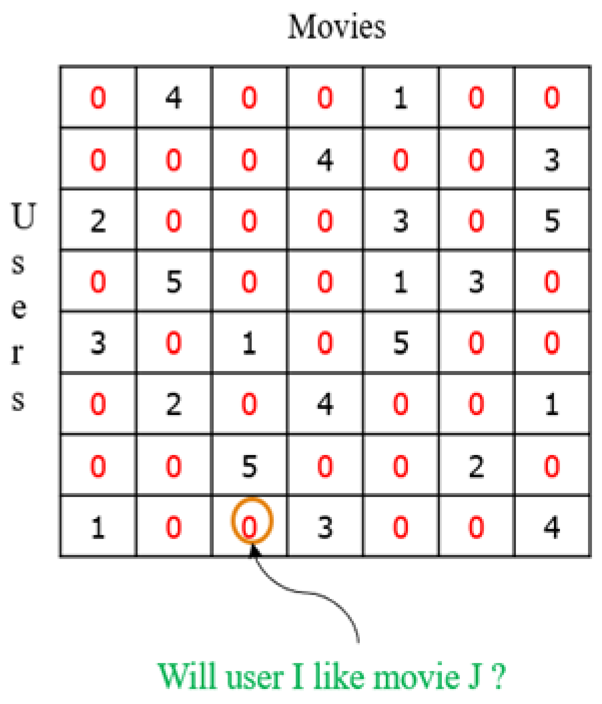
Users	User-item interactions matrix	Items
suscribers	rating given by a user to a movie (integer)	movies
readers	time spent by a reader on an article (float)	articles
buyers	product clicked or not when suggested (boolean)	products

Fig: 1.4 Collaborative Filtering Recommender model

Then, the main idea that rules collaborative methods is that these past user-item interactions are sufficient to detect similar users and/or similar items and make predictions based on these estimated proximities.

1.5 DataSet

In recommender systems, we take rating matrix as our dataset where $Y = [\, y\_ij \,]$ is a N × M user/items rating matrix where N is the number of users and M is the numbers of items. Each element $y\_ij \in \{\mathbf{0}, \mathbf{1}, \ldots, \boldsymbol{R}\}$, where $\boldsymbol{R}$ is the total level of rating and $\mathbf{0}$ indicate unknown rating. The goal is to predict the unknown ratings, represented by $y\_ij = \mathbf{0}$.


Movies
Users
Will user I like movie J ?

0	4	0	0	1	0	0
0	0	0	4	0	0	3
2	0	0	0	3	0	5
0	5	0	0	1	3	0
3	0	1	0	5	0	0
0	2	0	4	0	0	1
0	0	5	0	0	2	0
1	0	0	3	0	0	4

Fig: 1.5 Example rating matrix

1.6 Matrix Factorization

Matrix factorization is a class of collaborative filtering algorithms used in recommender systems. Matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices.

Let’s consider an interaction matrix M (n x m) of ratings where only some items have been rated by each user (most of the interactions are set to None to express the lack of rating). We want to factorise that matrix such that

$$M \approx X.Y^T$$

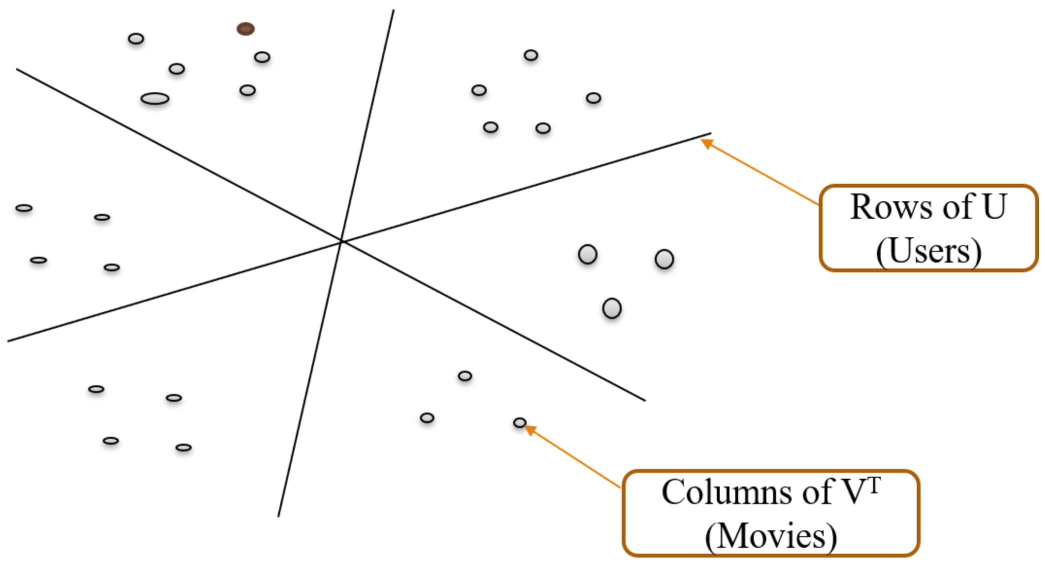


Fig: 1. 6 Geometrical Interpretation of matrix factorization

where X is the “user matrix” (n x l) whose rows represent the n users and where Y is the “item matrix” (m x l) whose rows represent the m items: (1*l) item (1*l)

$$user_i \equiv X_i \qquad \forall i \in \{1, ..., n\}$$
$$item_j \equiv Y_j \qquad \forall j \in \{1, ..., m\}$$

Chapter 2.

Related work

2.1 Maximum Margin Matrix Factorization

Maximum Margin Matrix factorization(MMMF) is a breakthrough in Rating based Recommender systems.
Maximum Margin Matrix Factorization (MMMF) is an effective collaborative filtering approach. MMMF suffers from the data sparsity problem, i.e., the number of items rated by the users are very small as compared to the very large item space. Recently, techniques like cross-domain collaborative filtering (transfer learning) is suggested for addressing the data sparsity problem.

For a partially observed ordinal rating matrix, the focus is on determining low-norm latent factor matrices U (of users) and V (of items) so as to simultaneously approximate the observed entries under some loss measure and predict the unobserved entries. When the rating matrix contains only two levels (±1), rows of V can be viewed as points in k-dimensional space and rows of U as decision hyperplanes in this space separating +1 entries from −1 entries.

When hinge/smooth hinge loss is the loss function, the hyperplanes act as maximum-margin separators.

$$M \approx X.Y^T$$

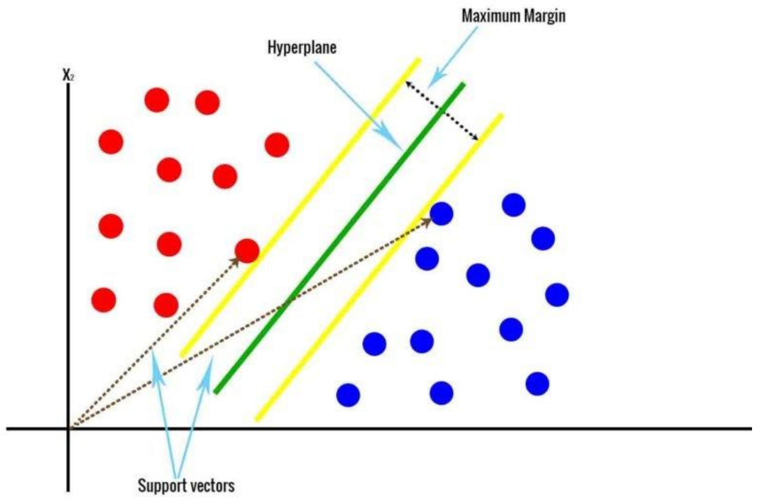


Fig: 2.1 Using support vector in matrix factorization for getting maximum margin

2.2 Hierarchical Matrix Factorization

Collaborative filtering(HMF) using multiple binary maximum margin matrix factorizations
Authors:- Vikas Kumar, Arun K. Pujari, Sandeep Kumar Sahu, Venkateswara Rao Kagita,Vineet Padmanabhan

HMF is a novel stage-wise matrix completion technique that makes use of several bi-level MMMFs in a hierarchical fashion.
Uses a bi-level MMMF.
The approach is to decompose a multiclass problem into multiple, independently trained, binary classification problems and to combine them appropriately so as to form a multiclass classifier.
At every stage of HMF, we predict the entries of the matrix of a specific ordinal value. In other words, at Stage q, HMF attempts to complete only those unobserved entries of the rating matrix where the rating q is predicted. Thus for a R-level ordinal rating matrix, we use R − 1 stages.
We search for matrices U and V which yield a minimum trace norm of X = UV’ and approximate Y on entries in training data .
The entries in the rating matrix Y have two levels, {±1} with +1 for likes and −1 for dislikes.

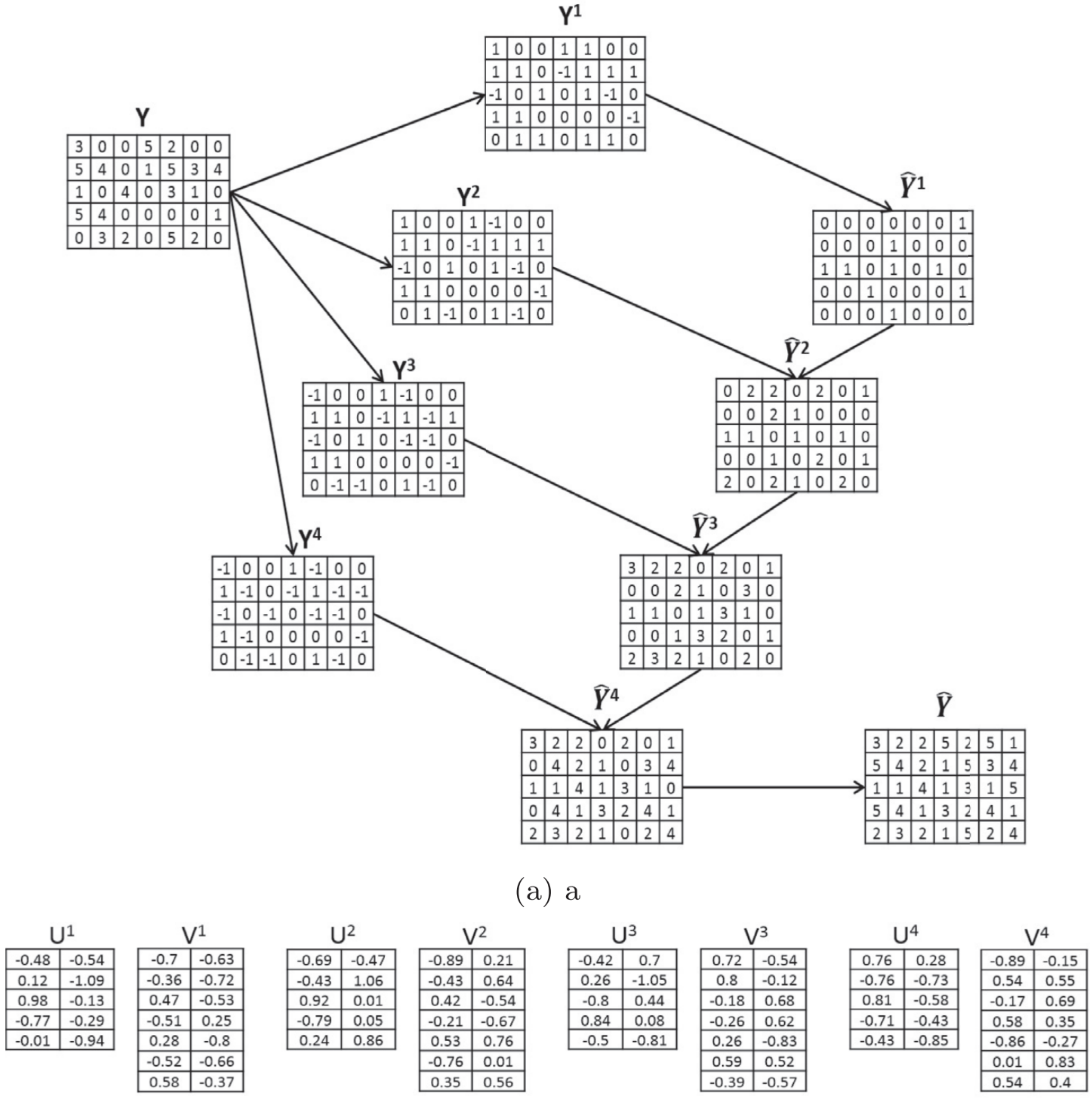
Which uses smooth hinge loss h(z) and its derivative h'(z).

$$h(z) = \begin{cases} 0 & \text{if } z \geq 1 \\ \frac{1}{2}(1-z)^2 & \text{if } 0 < z < 1 \\ \frac{1}{2} - z & \text{otherwise} \end{cases} \qquad h'(z) = \begin{cases} 0 & \text{if } z \geq 1 \\ z-1 & \text{if } 0 < z < 1 \\ -1 & \text{otherwise} \end{cases}$$

In each stage, a bi-level MMMF is employed and for this purpose, the R-level ordinal rating matrix Y is converted to a bi-level rating matrix Y^q by the following process.

$$y_{ij}^q = \begin{cases} -1 & \text{if } (i,j) \in \Omega \wedge y_{ij} \leq q \\ +1 & \text{if } (i,j) \in \Omega \wedge y_{ij} > q \\ 0 & \text{if } (i,j) \notin \Omega \end{cases}$$

We employ bi-level MMMF described in (Section 4 has to include some link) for factorization of Y^q. The outcome of bi-level MMMF at Stage q are low rank matrices Uq and Vq approximating Y^q and are used for prediction at Stage q. The unobserved entry (i, j) is predicted to be q if (i, j) is not predicted till the previous stage (Stage q − 1) and the sign of the $(U^q V^{q\,T})_{ij}$ is negative. At the last stage, Stage R − 1, after employing the above rule to predict entries with values R − 1, we complete the matrix by assigning R for all remaining unobserved entries.



(a) a

(b) b

(c) c

(d) d

(e) d

Fig: 2.2 HMF tree structure

Chapter 3.

Proposed work

3.1 Outline of Proposed work

After going through the HMF model we observed that different approaches can be used as an improvement based on observed bias problem. This bias problem arises due to the large difference in the number of training samples between two classes of both +1 and -1 labels, formed while using binary MMMF as subprocess in HMF.
For example if the difference between the number of training examples of -1 label and +1 label is high there is a tendency for bias towards higher number of training examples. There is a likely chance our classifier classifies all label points as points of higher number of training examples.

In our project, We want to address this issue using a different approach,which is pair-wise MMMF.

So our main goal is to segregate points effectively such that for every user the points will be on forming two clusters one of +1 label and other -1 label based on the data given by the user.
The approach is to increase the difference directly between the product values of (U.V’) of pairs.

If we can increase the difference between pairs, the distance between low-norm latent factor matrices V (of items) increases so the division will be such that users can divide points without the problem of bias.

Here the computation for calculating distance between all pairs is high. So, we take some subset of pairs randomly for reducing the computation, which can be achieved by implementing methods like random sampling.

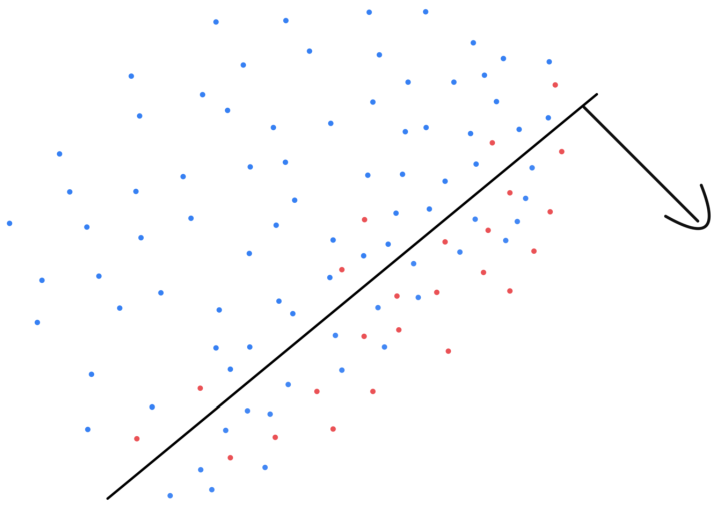


Fig: 3.1 Showing bias problem

The reason for choosing this approach is that the division with respective lines may be biased by the number of points on each side. If we want to increase the separation between the points we can increase the distance between the product of U.V’ which in turn increases the separation between the pairs which are on opposite sides of lines(which are users) so that the user can easily divide the likes(+1) and dislikes(-1). We think that the approach can nullify the bias and increase the accuracy of the recommender system.

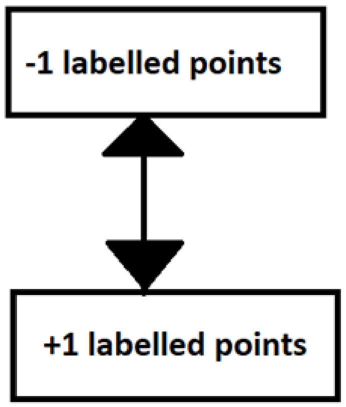


Fig: 3.2 Image showing to increase distance between -1 and +1 points

3.2 Details of algorithm

3.2.1 Data

Y - Data given is matrix of shape (n x m)
n users as rows and m users as columns.
Y_{ij} - Rating given by user i to item j.
If Y_{ij} is zero it means not predicted.

3.2.2 How we divide data

The matrix has values ranging from 0 to R.
0 implies rating was not predicted.
1-R means the user has given a rating.
So the ratings present are divided into test and train.
Train matrix contains ratings which are selected in the division process as mentioned above and all other values as zeros.
Test matrix contains ratings which are selected in the division process as mentioned above and all other values as zeros.

3.2.3 What is a stage and How many stages will be present for rating matrix with range 1-R..

For ratings ranging from 1 to R - (R-1) stages are applied.
In Each stage a binary rating matrix is calculated as mentioned below.
This binary matrix is used for training this stage.

For example there is a rating matrix with range from 1 to R.
The stages executed are
Stage 1 - (1) vs (2,3,4,5)
Stage 2 - (1,2) vs (3,4,5)
Stage 3 - (1,2,3) vs (4,5)
Stage 4 - (1,2,3,4) vs (5)

3.2.4 How a binary matrix calculated from matrix containing ratings 1-R for each stage

Here, we take 1-R rating matrix and calculate the binary matrix by finding 1 vs 2,3,....,R as -1 vs +1 by taking all ratings which are rated as 1 as -1 and rest of 2,3,....,R ratings as +1 and this is stage one.In similar fashion stage two as 1,2 vs 3,4,...,R and stage three as 1,2,3 vs 4,5,....R and soon. We have (R-1) stages for each stage one matrix. So, we get R-1 binary matrix's.

$$T_{ij}^{r} = \begin{cases} +1, & for\ r \geq Y_{ij} \\ -1, & for\ r < Y_{ij} \end{cases}$$

3.2.5 What is a pair

A pair is represented as (i1,i2)
which implies i1 as -1 and i2 as +1 in the binary matrix.
Some users may support this pair and some not.
Each user has a different set of pairs based on its ratings.

for example the binary ratings of a user for 5 items are

items -	1	2	3	4	5
ratings -	1	0	-1	-1	1

Pairs for this user are - (3,1),(3,5),(4,1),(4,5).

3.2.6 Generating pairs

The pairs are generated based on the binary matrix which is constructed for a level mentioned above.
So all combination of pairs are taken (i1,i2)
i1 - range 1 to R
i2 - range 1 to R
Then we find users who support this pair and a list is created.
If no users support a pair then it is discarded.

3.2.7 What is a batch

A batch is used in the process of training in the process of mini batch update.
Batch is a value signifying the number of pairs used in epoch.

3.2.8 Selecting Pairs for training

Every pair cannot be selected because if items are n, the possible pairs for each user are n(n-1)/2. This will increase the running time of the algorithm, so we used a random sampling process of selecting some pairs while training for each epoch in stage which is nothing but a mini batch update process.
After doing some observation we observed that after selecting some number of pairs there is no improvement in the model. So there is an optimal value for selecting no of pairs for the dataset.
The applied process is - Initially all pairs in a stage are recorded and then while training some set of points were taken and used for that epoch.

3.2.9 Hinge Loss Function

This function is used because we want to maximize the distance between pairs so we want a function which gives output higher for lower input and lower for higher input. Hinge Loss Function does this job for input greater than or equal to 1 output is zero and for input less than 1 the output is higher.

$$h(z) = \begin{cases} 0 & \text{If } z \geq 1 \\ \frac{(1-z)^2}{2} & \text{If } 0 < z < 1 \\ \frac{1}{2} - z & \text{If } z \leq 0 \end{cases}$$

We are calculating the difference between the product of U.V' of +1 label and product of U.V' of -1 label for all pairs. Since we need to separate all items with respect to each user, gradients now will be modifying both U and V matrices for every epoch.

S_{+1}^{i} *Randomly choosen indices of* $+1$ *label for user* i *in* T_{ij}^{r}
S_{-1}^{i} *Randomly choosen indices of* -1 *label for user* i *in* T_{ij}^{r}

Loss Function

$$\min_{U,V} J(U,V) = \sum_{i} \sum_{j \in S_{+1}^{i}} \sum_{k \in S_{-1}^{i}} h(U_i V_j^T - U_i V_k^T) + \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2)$$

3.2.10 Gradients

Gradients are derived using gradient descent algorithm.

- $\frac{\partial J}{\partial U_i} = \lambda U_i + \sum_{j \in S_{+1}^{i}} \sum_{k \in S_{-1}^{i}} h'\left(U_i V_j{}^T - U_i V_k{}^T\right).(V_j - V_k)$
- $\frac{\partial J}{\partial V_j} = \lambda V_j + \sum_{i} \sum_{k \in S_{-1}^{i}} h'\left(U_i V_j{}^T - U_i V_k{}^T\right).(U_i)$ for $j \in S_{+1}^{i}$
- $\frac{\partial J}{\partial V_k} = \lambda V_k + \sum_{i} \sum_{j \in S_{+1}^{i}} h'\left(U_i V_j{}^T - U_i V_k{}^T\right).(-U_i)$ for $k \in S_{-1}^{i}$

Gradients

Hinge Loss function derivative h’(z) is also defined.

$$h'(z) = \begin{cases} 0 & \text{if } z \geq 1 \\ z - 1 & \text{if } 0 < z < 1 \\ -1 & \text{otherwise} \end{cases}$$

3.2.11 How we convert real values of U.V’ to binary

NeagtiveMean - For a user the mean of -1 points.
PositiveMean - For a user mean of +1 points.
Based on the number of negative and positive points the threshold is calculated.
The points above threshold are labelled as +1 and below are labelled as -1
This procedure is applied for all users.

3.2.12 How we combine U.V’ values from each stage to final matrix

Initially a Matrix of all zeros is taken.
For each stage the Result Matrix filled by the last stage Result Matrix and this stage (U.V’) which is converted into binary and if -1 labelled position is zero in the Result Matrix filled by last stage then the position is filled with present stage value. +1 labelled position in this stage binary Matrix is ignored as those will be filled by next stages.
So after stage 1 - (R-1) the ratings, 1 to (R-1) are filled.
At last the Result Matrix with values 0 are replaced with Rating R.

3.2.13 What parameters are used in this model

Here we use U and V parameters which are low-norm latent factor matrices and shape of (n x p) and (m x p) respectively.
The rating matrix has a shape (n x m).
The rating matrix is found using U and V as U.V’ (V’ implies V transpose and it is a matrix multiplication).

3.2.14 What hyper-parameters are used

Hyperparameters used are p for low-norm latent factor matrices
lambda regularization factor.
epochs for a stage and batch size for random sampling.

3.2.15 How can parallelization can be applied

In the proposed method, we can apply parallelization to run different functions and subsequently the running time decreases. for generating pairs, we convert all ratings into converting -1 and +1 label points stage wise and get pairs. We can run each stage in parallel and get the resultant matrix for each stage and club them accordingly.

3.2.16 What Model Does

Final matrix is the resultant predicted matrix by PWMMMF.
Initially it is all values are zero
All Stages are executed, after each stage the Final matrix is updated with the stage rating.
In Each stage:

i. Calculate the binary level Matrix for the present stage rating value.
ii. Calculate Pairs and initialize parameters U and V.
iii. Calculate the objective and update parameters for specified epochs.
iv. Calculate the stage matrix X using U.V’
v. Converting real valued X to binary valued X(-1 and +1).
vi. Combining X values with Final Matrix.

The above process is done for R-1 stages if there are R ratings.
At last all the zeros will be replaced with the highest rating value.
For example:
if there are 5 ratings(1,2,3,4,5)
Stage 1 : Updating 1 values in Final Matrix
Stage 2 : Updating 2 values in Final Matrix
Stage 3 : Updating 3 values in Final Matrix
Stage 4 : Updating 4 values in Final Matrix
At last all zeros in Final Matrix are replaced with rating value 5.

Chapter 4.

Coding And Experimentation setup

4.1 Designed code

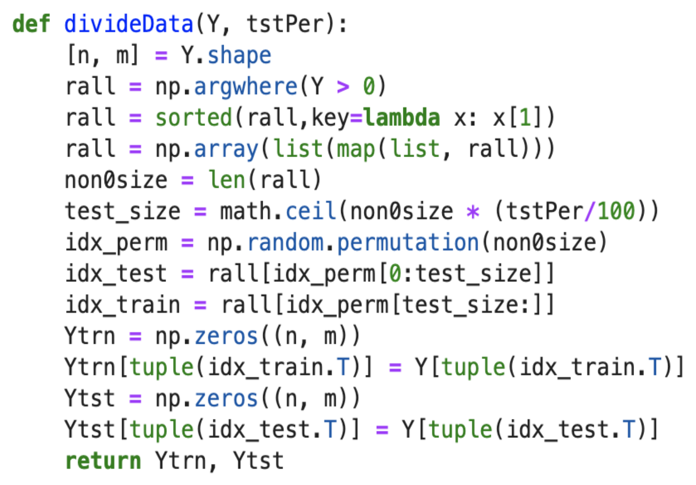
Here we enlist all the relevant code fragments used by us in the implementation along with necessary input and expected output parameters.

4.1.1 DivideData :

Input - Rating Matrix(**Y**) and test percentage (**tstPer**).

Output - Train Matrix(**Ytrn**) and Test Matrix(**Ytst**).

We divide data by taking test percent = tstPer%. So (100-tstPer)% ratings will be in the training matrix Ytrn and tstPer% of ratings will be in the testing matrix Ytst.



```
def divideData(Y, tstPer):
    [n, m] = Y.shape
    rall = np.argwhere(Y > 0)
    rall = sorted(rall,key=lambda x: x[1])
    rall = np.array(list(map(list, rall)))
    non0size = len(rall)
    test_size = math.ceil(non0size * (tstPer/100))
    idx_perm = np.random.permutation(non0size)
    idx_test = rall[idx_perm[0:test_size]]
    idx_train = rall[idx_perm[test_size:]]
    Ytrn = np.zeros((n, m))
    Ytrn[tuple(idx_train.T)] = Y[tuple(idx_train.T)]
    Ytst = np.zeros((n, m))
    Ytst[tuple(idx_test.T)] = Y[tuple(idx_test.T)]
    return Ytrn, Ytst
```

4.1.2 LevelWiseBinaryMatrix :

We convert the rating matrix Y to level wise binary matrices according to number of rating levels.

```
while levelNo < L :
    RcapTrn = np.zeros((n, m));
    RcapTst = np.zeros((n, m));

    RcapTrn[(Ytrn <= levelNo) & ( Ytrn != 0 )] = -1;
    RcapTst[(Ytst <= levelNo) & ( Ytst != 0 )] = -1;

    RcapTrn[(Ytrn > levelNo) & ( Ytrn != 0 )] = 1;
    RcapTst[(Ytst > levelNo) & ( Ytst != 0 )] = 1;
```

4.1.3 GeneratePairs :

We generate pairs of +1 labeled and -1 labeled points with respect to each user.

Input : Binary rating matrix (**Y**) and minimum number of users supporting a pair (**perItemPairs**)
Output : Two dictionaries **pairWiseUser** and **itemInPair**.

A notation is used in this process - ‘i1_i2’ a string which signifies a pair i1 and i2 which are opposite(-1 and +1)
itemInPair - dictionary with key - string(‘i1_i2’) and value - list([i1 and i2])
To get i1 and i2 from notation specified above we use itemInPair dictionary
pairWiseUser - dictionary with key - string(‘i1_i2’) and value - list(list of users supporting pair ‘i1_i2’).
Where we find the pairs which are opposite w.r.t users and supported by a minimum set of users (specified by perItemPairs).

```
def generatePairs(Y, perItemPairs):
    minUsersPairs = 1
    pairWiseUser = {}
    itemInPair = {}
    n, m = Y.shape
    for itemNo in range(0, m):
        items = np.array(list(set(range(0, m)) - {itemNo}))
        itemsIdxrandPerm = np.random.permutation(items)  # items

        pairs = np.zeros((m, 2))
        selectedIdx = 0
        usersPairs = {}
        for pairNo in range(0, m-1):
            usersWithThisPair = np.where(
                ((Y[:, itemNo] == -1) & (Y[:, itemsIdxrandPerm[pairNo]] == 1)) == True)[0]
            if len(usersWithThisPair) >= minUsersPairs:
                pairs[selectedIdx, :] = [itemNo, itemsIdxrandPerm[pairNo]]
                usersPairs[selectedIdx] = list(usersWithThisPair.T)
                selectedIdx = selectedIdx + 1
            if selectedIdx >= perItemPairs:
                break
        for pairNo in range(0, selectedIdx):
            tmp = str(int(pairs[pairNo, 0]))+"_"+str(int(pairs[pairNo, 1]))
            if tmp in pairWiseUser:
                pairWiseUser[tmp] = pairWiseUser[tmp] + usersPairs[pairNo]
            else:
                pairWiseUser[tmp] = usersPairs[pairNo]
                itemInPair[tmp] = pairs[pairNo, :].astype('int64')
    return pairWiseUser, itemInPair
```

4.1.4 DrawSample :

Return only some pairs from all pairs randomly for computational efficiency.

Input - List of pairs in the form ‘i1_i2’ (**Pairs)** and value specifying maximum no of batches to be selected (**maxBatch).** Each batch contains 1000 pairs.This batch was done to increase parallelization.

Output - Dictionary(**batchWisePairs**) and **maxBatch**(No of batches selected).

batchWisePairs : key - integer(batch number) and value - list(list of pairs in the form string ‘i1_i2’).

```
def drawsample(pairs, maxBatch):
    maxBatchSize = 1000
    batchWisePairs = {}
    randPairsIdx = np.random.choice(pairs.shape[0], maxBatch*maxBatchSize, replace=False)
    for batchNo in range(maxBatch):
        if ((batchNo+1)*maxBatchSize) > len(randPairsIdx) :
            batchWisePairs[batchNo] = pairs[randPairsIdx[(batchNo)*maxBatchSize:]]
            maxBatch = batchNo+1;
            break;
        else :
            batchWisePairs[batchNo] = pairs[randPairsIdx[(batchNo)*maxBatchSize:(batchNo+1)*maxBatchSize]]
    return batchWisePairs, maxBatch
```

4.1.5 PairWiseHinge :

Returns the loss values and gradients to be updated.

Input : Features(**v**) and the parameters required for training in the form of a dictionary(**parameter**).

parameter - dictionary containing the lambda value, pairs data, train sample(Y).

Output : Objective Value(**obj**) and gradients for updating(**grad**).

```python
def pairWiseHinge(v, parameter):
    Y = parameter['Y']
    lam = parameter['lambda']
    p = parameter['p']
    pairWiseUser = parameter['pairWiseUser']
    itemInPair = parameter['itemInPair']
    batchWisePairs = parameter['batchWisePairs']
    maxBatch = parameter['maxBatch']
    alpha = parameter['alpha']
    n, m = Y.shape
    U = v[0, :n*p].reshape(n, p)
    V = v[0, n*p:].reshape(m, p)
    Unorm = np.sum(np.square(U))
    Vnorm = np.sum(np.square(V))
    regobj = (lam/2)*(Unorm+Vnorm)
    dU = lam*U
    dV = lam*V
    X = np.dot(U, V.T)
    lossObj = 0
    for batchNo in range(0, maxBatch):
        batch = batchWisePairs[batchNo]
        noOfSample = batch.shape[0]
        UVminusUV = np.zeros((n, noOfSample))
        VminusV = np.zeros((noOfSample, p))
        userInSamplesFlag = np.zeros((n, noOfSample))
        itemInSamplesFlag = np.zeros((noOfSample, m))
        for sampleNo in range(0, noOfSample):
            key = batch[sampleNo]
            itemPair = itemInPair[key]
            UVminusUV[:, sampleNo] = X[:, itemPair[1]] - X[:, itemPair[0]]
            VminusV[sampleNo, :] = V[itemPair[1], :] - V[itemPair[0], :]
            userVec = pairWiseUser[key]
            userInSamplesFlag[userVec, sampleNo] = 1
            itemInSamplesFlag[sampleNo, itemPair[0]] = -1
            itemInSamplesFlag[sampleNo, itemPair[1]] = 1
        lossInBatch = np.sum((h(UVminusUV - alpha) * userInSamplesFlag))
        lossObj = lossObj + lossInBatch
        dH = hprime(UVminusUV - alpha) * userInSamplesFlag
        dU = dU + np.dot(dH, VminusV)
        tmp = np.dot(dH, itemInSamplesFlag)
        dV = dV + np.dot(tmp.T, U)
    obj = regobj + lossObj
    grad = np.concatenate((dU, dV), axis=None)
    return obj, grad
```

4.1.6 SmoothHingeLoss :

```
def h(z):
    zin01 = (z > 0).astype('int64') - (z >= 1).astype('int64')
    zle0 = (z < 0).astype('int64')
    ret = (((zin01/2-zin01*z) + (((zin01*z)**2)/2)) + (zle0/2)) - zle0 * z
    return ret


def hprime(z):
    zin01 = (z > 0).astype('int64') - (z >= 1).astype('int64')
    zle0 = (z < 0).astype('int64')
    ret = zin01 * z - zin01 - zle0
    return ret
```

4.1.7 PredicitonAtLevel :

This function fills the final prediction matrix using the mean centroid method, confusion matrix and already filled final prediction matrix from the previous stage.

RFinal - Is the final matrix predicted after levelNo of stages.

- Initial this will be all zeros.
- After the last stage this contains ratings ranging from 1 to R.

Input -

RFinal : Matrix filled by previous stage(in case of stage 1 it is filled with zeros),
X : Present stage matrix found using U.V’
RcapTrn : Bi-level Train matrix
Margin : used to move threshold value.
confusingEntriesConfi,confusingEntriesLabel - Used for updating some doubtful points and converting some doubtful points to fixed.
levelNo : Used to specify the present stage level number in RFinal matrix.

Output -
RFinal - Updated RFinal matrix using this stage X.
confusingEntriesConfi,confusingEntriesLabel - Updated values after this stage.

```python
def predictionAtLevel(RFinal,X,RcapTrn, margin, confusingEntriesConfi,confusingEntriesLabel, levelNo):
    [n,m] = RFinal.shape;
    ttlNegative = np.sum(RcapTrn==-1,1);
    ttlPositive = np.sum(RcapTrn==1,1);
    negativeMean = np.sum(X*(RcapTrn==-1),1)/ttlNegative;
    negativeMean[np.isnan(negativeMean)] = -1000
    positiveMean = np.sum(X*(RcapTrn==1),1)/ttlPositive;
    positiveMean[np.isnan(positiveMean)] = 1000
    ttlMargin = positiveMean-negativeMean;
    theta = negativeMean + (ttlMargin/(ttlPositive+ttlNegative))*ttlNegative;
    theta = theta.reshape(-1,1)
    T = np.dot(theta,np.ones((1,m)))
    y = (X < T)
    T = T-margin
    y1 =(X < T)
    RFinal[np.logical_and(RFinal==0,y1)] = levelNo;
    confuseGainConfi = confusingEntriesConfi!=0 & y1;
    confusingEntriesConfi[confuseGainConfi] = 0;
    confusingEntriesLabel[confuseGainConfi] = 0;
    confusingEntries = y.astype('int64') - y1.astype('int64');
    confusingEntries = np.logical_and(confusingEntries, RFinal==0)
    confidence = abs(T - X)*confusingEntries;
    tmp = confidence > confusingEntriesConfi;
    confusingEntriesConfi[tmp] = confidence[tmp];
    confusingEntriesLabel[tmp] = levelNo;

    return RFinal, confusingEntriesConfi, confusingEntriesLabel
```

4.1.8 Metrics :

We used three metrics in our project :

1) Zero one error
2) Mean average error
3) Root mean squared error

```python
# Zero-one error
def zoe(x, y):
    res = sum(sum((x*(y != 0)) != y))/np.count_nonzero(y)
    return res

# Mean absolute error
def mae(a, b):
    c =  np.sum(abs(a*(b != 0)-b))/np.count_nonzero(b)
    return c


def RMSE(X, Y):
    tmp = ((Y - X) * (Y != 0)) ** 2
    error = np.sqrt(sum(sum(tmp)) / sum(sum(Y != 0)))
    return error
```

4.2 Model tunning

4.2.1 Varying Regularization parameter

For finding optimal regularization parameter, we found out the ZOE value for each stage with different lambda values in the range of [10 - 15].

Level No : 1 optimal : 125, Level No : 2 optimal : 150

Level No : 3 optimal : 20, Level No : 4 optimal : 10

Stage - 1		Stage - 2		Stage - 3		Stage - 4	
Lambda	ZOE	Lambda	ZOE	Lambda	ZOE	Lambda	ZOE
10	0.06545	10	0.19155	10	0.32775	10	0.2290833
20	0.06431	20	0.1877	20	0.32045	20	0.2438165
30	0.06546	30	0.1891	30	0.32545	30	0.2734836
40	0.0662	40	0.1904	40	0.330467	40	0.3107167
50	0.06573	50	0.1929	50	0.336567	50	0.3427834
60	0.06429	60	0.1898	60	0.3441	60	0.3657496
70	0.06403	70	0.1915	70	0.3487833	70	0.3842165
80	0.06276	80	0.18686	80	0.35255	80	0.4002167
90	0.06208	90	0.1844	90	0.358767	90	0.4112833
100	0.06123	100	0.1822	100	0.3589834	100	0.4279167
125	0.061	125	0.179	125	0.365	125	0.4603665
150	0.0618	150	0.1773	150	0.3852504	150	0.53435
175	0.06186	175	0.1789	175	0.4133	175	0.6163
200	0.06184	200	0.1815	200	0.446594	200	0.716233

4.2.2 Varying parameters of model

Here, we are taking different sizes of pairs randomly for decreasing computation and for finding optimal values.

Epochs For Each Stage	Sample Size	Train			Test		
		ZOE	MAE	RMSE	ZOE	MAE	RMSE
10	1000	0.66531	1.00826	1.38458	0.6761	1.01515	1.38125
50	1000	0.69134	1.0916	1.48019	0.69165	1.0939	1.48233
10	5000	0.52446	0.71131	1.08306	0.61205	0.82605	1.16312
50	5000	0.50087	0.67992	1.05972	0.60515	0.8133	1.15261
10	10000	0.38115	0.49178	0.86999	0.5872	0.76505	1.08989
50	10000	0.31028	0.39745	0.77811	0.57315	0.7337	1.0566
10	15000	0.27291	0.34084	0.70756	0.57725	0.7337	1.05205
50	15000	0.19595	0.24088	0.58743	0.5669	**0.7076**	1.01961
10	20000	0.22772	0.27689	0.62511	0.5781	0.73125	1.04506
50	20000	0.13965	0.1669	0.47828	0.56215	**0.7014**	1.01346
10	30000	0.18042	0.21365	0.53784	0.57665	0.729	1.04115
50	30000	0.10929	0.12829	0.41399	0.5758	0.7211	1.02917

Here, we are getting optimal value at sample size **20000**

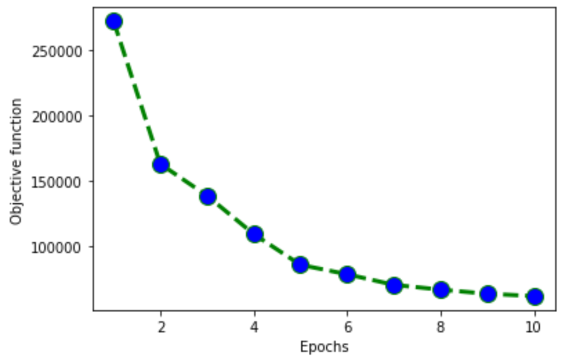
4.3 Graphs of Model varying over iteration

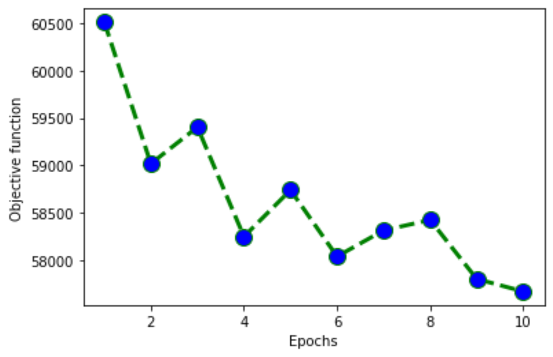
Here, we are calculating values objective values over iteration for each stage and as well ZOE error for each stage over iterations.

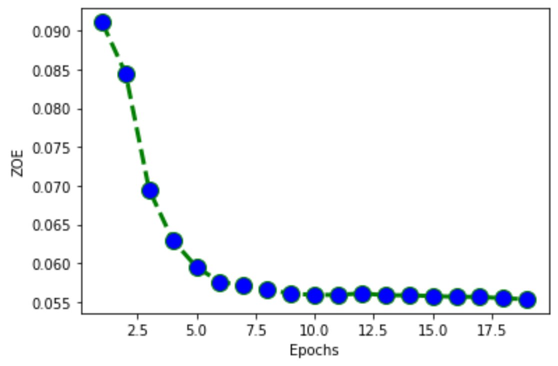
For every stage, we have three graphs:

1) Objective vs Epochs for first 10 iterations
2) Objective vs Epochs for second 10 iterations
3) ZOE vs Epochs for 20 iterations

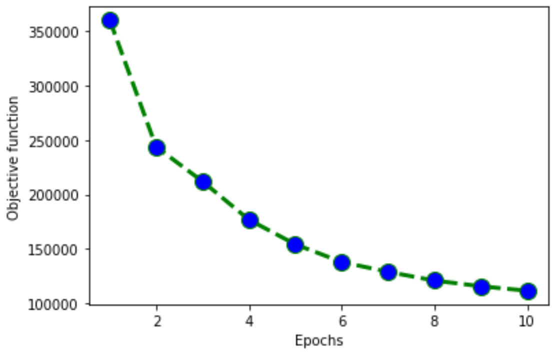
<u>Stage -1</u>

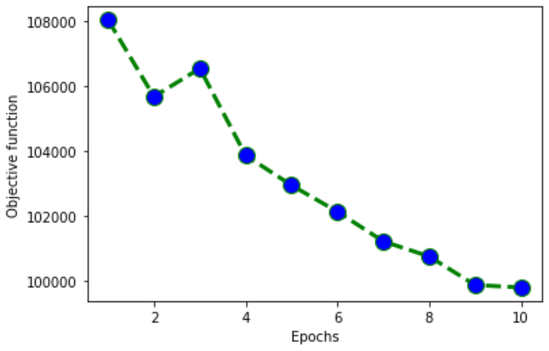


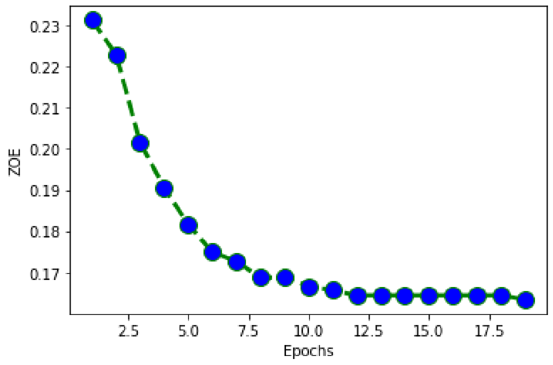




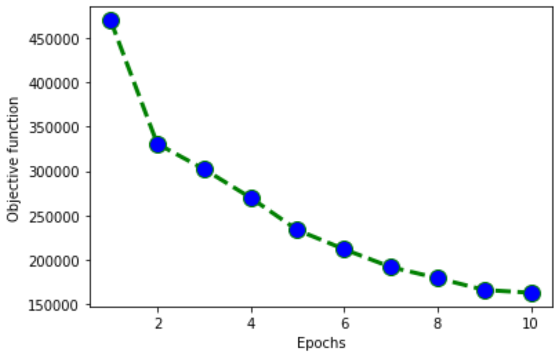
Stage-2

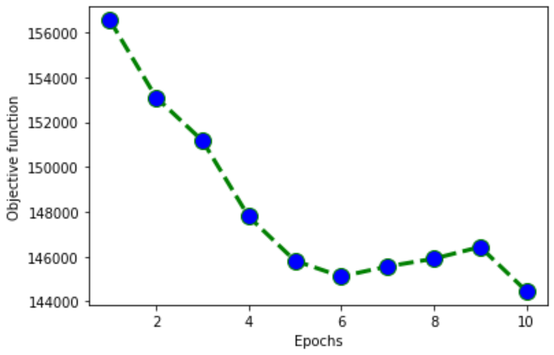


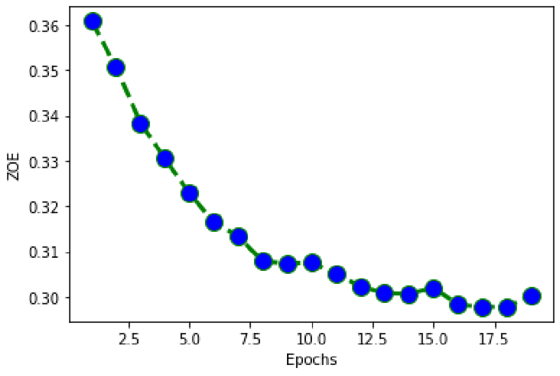




Stage-3

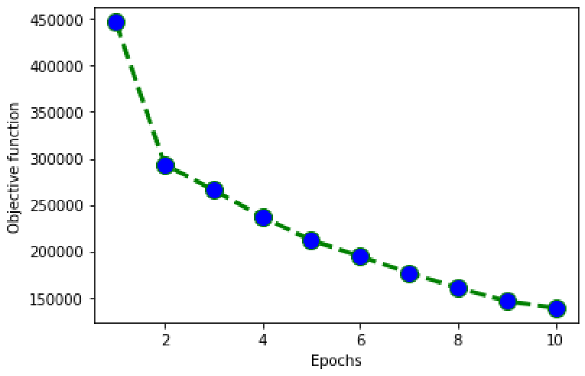




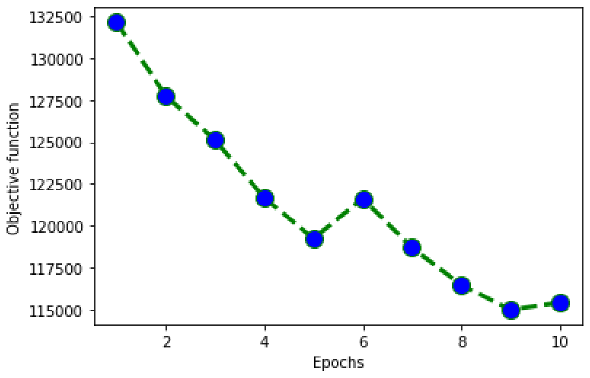


ZOE
Epochs

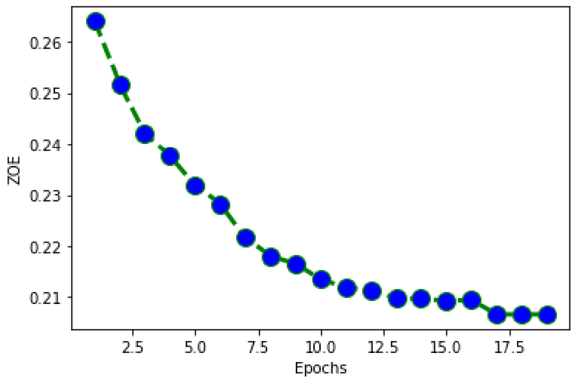
Stage-4



Objective function
Epochs



Objective function
Epochs

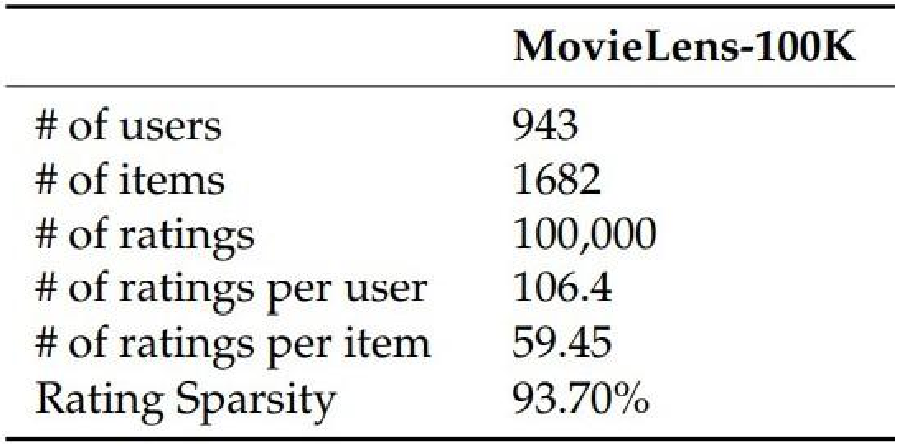


ZOE
Epochs

4.4 Experimentation setup

4.4.1 Dataset

In this section we analyse the performance of PWMMMF by taking into account factors such as accuracy. We use a benchmark dataset for our experiments, namely MovieLens 100k which is standard in the matrix factorization community. The MovieLens 100K movie ratings is a Stable benchmark dataset of 100,000 ratings from 1000 users on 1700 movies. There are 5 possible rating values, {1, 2 . . ., 5}.



	MovieLens-100K
# of users	943
# of items	1682
# of ratings	100,000
# of ratings per user	106.4
# of ratings per item	59.45
Rating Sparsity	93.70%

Fig: 4.1 The statistics of the dataset

4.4.2 Evaluation Measure

The proposed method is first to predict the user's rating score, and then to make recommendations based on the prediction. Therefore, we adopt the Mean Absolute Error (MAE) [31] method, which is commonly used in the field of recommendation system to evaluate the performance of the algorithms, and the expression is as follows:

$$MAE = \frac{1}{N}\sum_{i,j} |R_{ij} - \hat{R}_{ij}|,$$

where N is the number of test data, Rij represents the rating score on the j-th item by the i-th user in the test data, and Rˆij represents the predicted score obtained by using the recommendation algorithm.
Obviously, the lower the MAE value, the better the predicted results coincident with the user’s real
situation, which means better performance of the algorithm.

4.4.3 Compared Methods

In order to verify the effectiveness and accuracy of our method, we choose a variety of recommendation algorithms to compare with ours, which are listed as follows:

Item-Based (IB) : The basic idea of this method is to calculate the similarity between items and then recommend items similar to what is preferred by the active user. In this paper, we use the vector cosine similarity model to compute the item-item similarities.

SVD : This algorithm is based on the Singular Value Decomposition method, wherein the rating matrix is decomposed into two low dimensional matrices, which will be further used to make the prediction.

PMF : A widely used matrix factorization model, based on the Gaussian model for the recommendation. In this paper, we set the regularization parameters by the grid {0.01, 0.05, 0.1, 0.5, 1,5, 10}.

MCoC : The main idea of this method is clustering the users and items into several subgroups, and then making recommendations within each subgroup by using the basis CF method.

DsRec : A hybrid model by combining the basis clustering model and the matrix factorization model to improve prediction accuracy.

PMMMF : This method was proposed by Kumar, who improved the traditional MMMF method by using Proximal Support Vector Machine (PSVM).

Hern : This method was proposed by Hernando, so we use the first author's name to name this algorithm; they use the matrix factorization method based on Bayesian probability model to decompose the user-item rating matrix into two nonnegative matrices, where the values of all elements are between 0 and 1.

SCC : This method is the same as MCoC, which is a recommendation algorithm based on clustering; the difference is that it clusters items by using a self-constructing clustering method.

TyCo : This method borrows ideas of object typicality from cognitive psychology and finds "neighbors" of users based on user typicality degrees in user groups.

HMF : This method was proposed by Kumar, who improved the Hierarchical MMMF method.

4.5 Experimental Results

In our experiments, we divide the public data sets into two parts, respectively, randomly select 80% of the data as the training data, and the remaining 20% as the test data. The dataset is randomly divided three times and the average value of the MAE obtained in all the three tests is regarded as the final result.

	IB	SVD	PMF	MCoC	DsRec	PMMMF	Hern	SCC	TyCo	HMF	**PWMMMF**
1	0.7325	0.7412	0.7248	0.7024	0.7105	0.7221	0.7133	0.7574	0.7247	0.6904	**0.7047**
2	0.7290	0.7342	0.7205	0.7242	0.7201	0.7104	0.7087	0.7546	0.7301	0.6922	**0.70685**
3	0.7432	0.7388	0.7235	0.7189	0.7164	0.7166	0.7080	0.7608	0.7289	0.6946	**0.70715**
avg	0.7356	0.7376	0.7236	0.7197	0.7138	0.7167	0.7086	0.7588	0.7298	0.6924	**0.70623**

Comparison of the MAE Values on MovieLens-100K

The parameters set for our model are lambda = 12 for all stages, p = 100 which is low norm vector dimension, epochs = 50 and sample size = 20000 for mini batch update.

The motive for believing in this model is when we trained our model, the objective function value decreases simultaneously error decreases. So, this means we are going in the right direction.

After tuning the model the results were improved so we think increasing the distance between pairs is a good approach, which can be deep dived to get most out of it.

With our hardware limitations, We were able to work with 100k dataset only and not able to rigour parameter tuning. but, we were able to achieve MAE value which is at most near to HMF and better than other models.

Chapter 5.

Conclusions

In this report, We first introduced about recommender systems, various types of recommender systems and Matrix factorization.

We then discussed Maximum Margin Matrix factorization and Hierarchical Matrix Factorization in detail with formulations.

Later, we discussed the problem in the base paper and proposed a solution and explained about the approach we have chosen and gave insights to the method we have done.

We tried different approaches like restricting values of U.V’ from -1 to 1 before calculating pairwise distances,combining HMF objection function with ours,
Different distances formulas, different epochs for each stage.
Using different improvisations and approaches we could steadily improve our model’s initial accuracy to match and exceed the performance of many novel models and results previously established.

The present model does not perform as expected when compared with HMF. But the performance is good when compared with some recommender systems based on Matrix factorization.

We expect this method to perform better on other datasets but we are limited to work on this dataset only because of hardware limitations and availability of datasets.

We started with an MAE error of 0.80 and improved it to 0.70 in the span of four months.

We came to the conclusion that performance can be improved, as discussed the process is all about testing different approaches so different techniques may lead to better results.

Chapter 6.

References

[1] H. Byun, S.-W. Lee, Applications of support vector machines for pattern recognition: A survey, in: Pattern recognition with support vector machines, Springer, 2002, pp. 213–236.

[2] D. DeCoste, Collaborative prediction using ensembles of maximum margin matrix factorizations, in: Proceedings of the 23rd international conference on Machine learning, ACM, 2006, pp. 249–256.

[3] M. Fazel, H. Hindi, S.P. Boyd, A rank minimization heuristic with application to minimum order system approximation, in: American Control Conference, 2001. Proceedings of the 2001, Volume 6, IEEE, 2001, pp. 4734–4739.

[4] Y. Hu, Y. Koren, C. Volinsky, Collaborative filtering for implicit feedback datasets, in: Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on, IEEE, 2008, pp. 263–272.

[5] N.D. Lawrence, R. Urtasun, Non-linear matrix factorization with Gaussian processes, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 601–608.

[6] D.D. Lee, H.S. Seung, Learning the parts of objects by non-negative matrix factorization, Nature 401 (6755) (1999) 788–791.

[7] D.D. Lee, H.S. Seung, Algorithms for non-negative matrix factorization, in: Advances in Neural information processing systems, 2001, pp. 556–562.

[8] Y. Liu, H.H. Zhang, Y. Wu, Hard or soft classification? large-margin unified machines, J. Am. Stat. Assoc. 106 (493) (2011) 166–177.

[9] B.M. Marlin, Modeling user rating profiles for collaborative filtering, Advances in neural information processing systems, 2003. None

[10] A. Mnih, R. Salakhutdinov, Probabilistic matrix factorization, in: Advances in neural information processing systems, 2007, pp. 1257–1264.

[11] N.S. Nati, T. Jaakkola, Weighted low-rank approximations, in: In 20th International Conference on Machine Learning, AAAI Press, 2003, pp. 720–727.

[12] P. Paatero, Least squares formulation of robust non-negative factor analysis, Chemometrics and intelligent laboratory systems 37 (1) (1997) 23–35.

[13] S. Rendle, C. Freudenthaler, Z. Gantner, L. Schmidt-Thieme, Bpr: Bayesian personalized ranking from implicit feedback, in: Proceedings of the Twenty– Fifth Conference on Uncertainty in Artificial Intelligence, AUAI Press, 2009, pp. 452–461.

[14] J.D. Rennie, N. Srebro, Fast maximum margin matrix factorization for collaborative prediction, in: Proceedings of the 22nd international conference on Machine learning, ACM, 2005, pp. 713–719.

[15] J.D. Rennie, N. Srebro, Loss functions for preference levels: Regression with discrete ordered labels, in: Proceedings of the IJCAI multidisciplinary workshop on advances in preference handling, Kluwer Norwell, MA, 2005, pp. 180–186.

[16] R. Salakhutdinov, A. Mnih, Bayesian probabilistic matrix factorization using markov chain monte carlo, in: Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 880–887.

[17] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, Incremental singular value decomposition algorithms for highly scalable recommender systems, in: Fifth International Conference on Computer and Information Science, Citeseer, 2002, pp. 27–28.

[18] A.P. Singh, G.J. Gordon, A unified view of matrix factorization models, in: Machine Learning and Knowledge Discovery in Databases, Springer, 2008, pp. 358–373.

[19] N. Srebro, J. Rennie, T.S. Jaakkola, Maximum-margin matrix factorization, in: Advances in neural information processing systems, 2004, pp. 1329–1336.

[20] M. Volkovs, G.W. Yu, Effective latent models for binary feedback in recommender systems, in: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR ’15, 2015, pp. 313–322.

[21] M. Weimer, A. Karatzoglou, A.J. Smola, Improving maximum margin matrix factorization, Machine Learning 72 (3) (2008) 263–276.

[22] M. Xu, J. Zhu, B. Zhang, Nonparametric max-margin matrix factorization for collaborative prediction, in: Advances in Neural Information Processing Systems, 2012, pp. 64–72.

[23] M. Xu, J. Zhu, B. Zhang, Fast max-margin matrix factorization with data augmentation, in: Proceedings of the 30th International Conference on Machine Learning (ICML-13), 2013, pp. 978–986.

[24] C. Zhang, Y. Liu, Multicategory large-margin unified machines, The Journal of Machine Learning Research 14 (1) (2013) 1349–1386.

[25] T. Zhang, F.J. Oles, Text categorization based on regularized linear classification methods, Information retrieval 4 (1) (2001) 5–31.

[26] E. Zhong, W. Fan, Q. Yang, Contextual collaborative filtering via hierarchical matrix factorization, in: SDM, SIAM, 2012, pp. 744–755.

[27] Sarwar, B.; Karypis, G.; Konstan, J.; Riedl, J. Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th international conference on World Wide Web, Hong Kong, China, 1–5 May 2001; ACM: New York, NY, USA, 2001; pp. 285–295

[28] Sarwar, B.; Karypis, G.; Konstan, J.; Riedl, J. Application of Dimensionality Reduction in Recommender System-A. Case Study; DTIC Document. Technology Report; Minnesota Univ Minneapolis Dept of Computer Science: Minneapolis, MN, USA, 2000