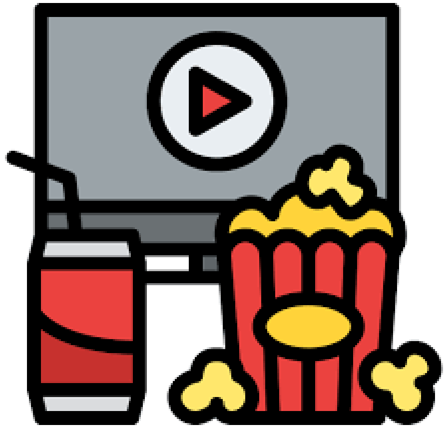
Probabilistic Matrix Factorization and Recommendation System

김현철

2023.07.04

Contents

Recommendation System



Real–life situation

– CGV wants to draw more audience to sell more tickets.
– Thus, want to build a recommendation system to promote attention to other films.
– For what basis should we recommend ?

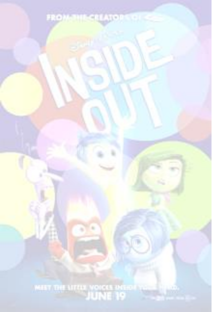
Recommendation System

[Content based filtering]

: Recommend new films that are related to the movies that user have seen

[Collaborative filtering]

: Recommend movies based on the taste of the user



Recommendation System

[Content based filtering]

: Recommend new films that are related to the movies that user have seen

[Collaborative filtering]

: Recommend movies based on the taste of the user

Recommendation System

[Content based filtering]

: Recommend new films that are related to the movies that user have seen

[Collaborative filtering]

: Recommend movies based on the taste of the user

Collaborative Filtering

	Movie 1	Movie 2	Movie 3	…	Movie M
User 1	5	?	4		1
User 2	2	1	?		5
User 3	3	1	4		5
…					
User N	?	2	3		?

Given a table of each user's ratings on the movies,
Estimate the unknown (possibly unseen movies) values and recommend if it is high

$$Y \approx UV^T$$

Y : Ratings table, n x m
U : User coefficient matrix, n x k
V : Factor matrix, m x k

(k : hyper-paramter)

Low rank approximation

Assume Y is fully known, we want to find
$X = UV^T$ such that $\|Y - X\|$ is minimum

By Eckart - Young - Mirsky theorem, X can be analytically found

1. Perform SVD to Y, such that

$$Y = \widetilde{U}\Sigma\widetilde{V}^T \quad , \widetilde{U}: \text{ n x d matrix}$$

$$\widetilde{V}: \text{ m x d matrix, and d} > \text{k}$$

2. Choose k vectors corresponding to k largest singular values

Proof of Eckart–Young–Mirsky theorem：https://en.wikipedia.org/wiki/Low-rank_approximation

Low rank approximation

However, Y is not fully known.
Thus, first fill out missing entries by

- Zero filling
- Row averaging
- Column averaging
- Entry averaging

Then, perform low rank approximation.
After estimating U and V, calculate missing values by matrix multiplication

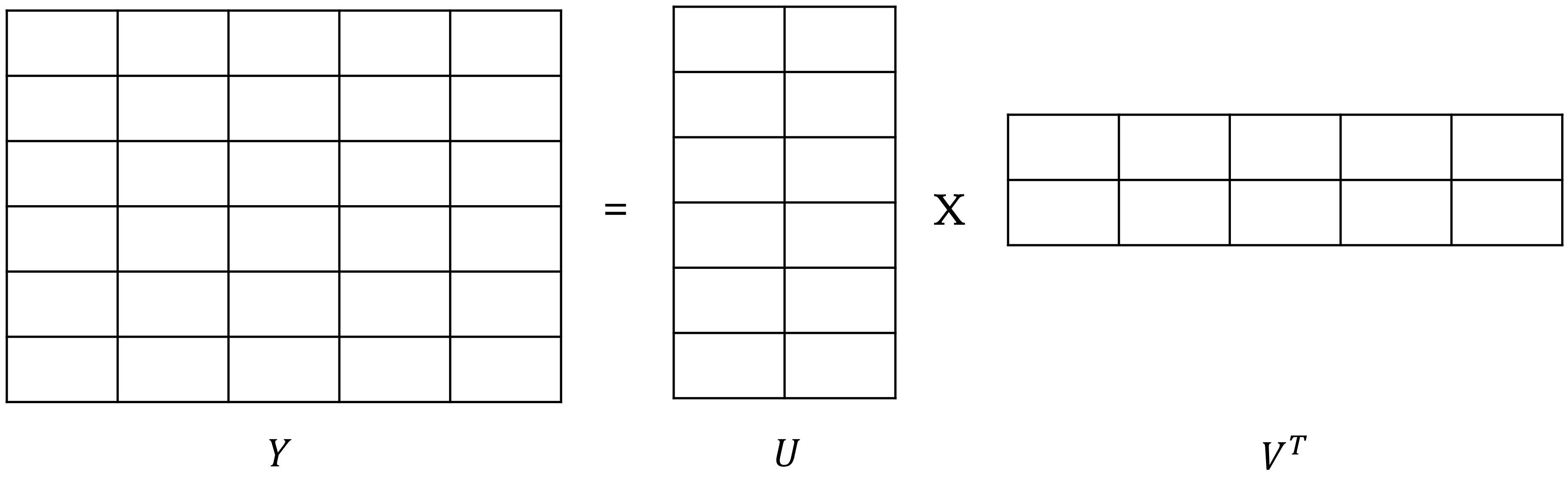
→ Works reasonably well when there are few missing values

Maximum Margin Matrix Factorization

Low rank approximation method does not work when

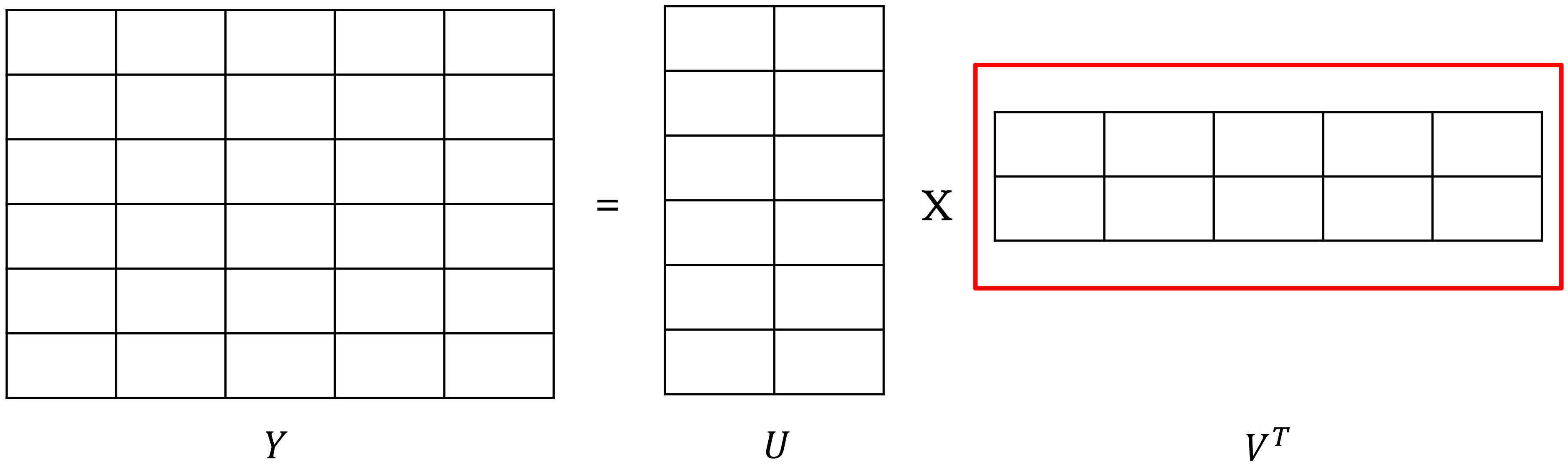
1. Many missing entries
2. Entries are discrete values

Instead of constraining the dimension, constrain the norm of U and V



Srebro, N., Rennie, J., & Jaakkola, T. (2004). Maximum-margin matrix factorization. Advances in neural information processing systems, 17.

Maximum Margin Matrix Factorization



Suppose that V^T is fixed,
Then, learning U = learning the linear projection of the column vectors of V^T to R^n.
Thus, U can be learned by minimizing $\frac{1}{2}\|U\|_{Fro}^2$
(Recall the learning process of SVM)

Srebro, N., Rennie, J., & Jaakkola, T. (2004). Maximum-margin matrix factorization. Advances in neural information processing systems, 17.

Maximum Margin Matrix Factorization

So, learning U and V can be done by minimizing

$$\frac{1}{2}\|U\|_{Fro}^2 + \frac{1}{2}\|V\|_{Fro}^2$$

Also, minimizing such term is equivalent to minimizing

$$\|X\|_\Sigma : \mathit{the\ sum\ of\ the\ singular\ values\ of\ } X$$

Therefore, we can directly learn X by Semi−Definite Programming

→ No need to constrain the dimension k

[Fast MMMF]
- SDP method is impractical
- Learns X by Stochastic Gradient Descent
- Faster, but can be suboptimal

Srebro, N., Rennie, J., & Jaakkola, T. (2004). Maximum−margin matrix factorization. Advances in neural information processing systems, 17.

Probabilistic Matrix Factorization

However, MMMF (or Fast MMMF) does not work with unbalanced dataset
– Ex) Only few people rate their movies, and they tend to rate every movies they have seen

Propose a method that
1. Scales linearly
2. Robust to unbalanced dataset

Maximize posterior probability $p(U,V|R,\sigma^2,\sigma_u^2,\sigma_v^2)$, $U \in \mathbb{R}^{D\times N}$, $V \in \mathbb{R}^{D\times M}$
where the prior distributions are assumed to follow spherical Gaussian

$$p(U|\sigma_U^2) = \prod_{i=1}^{N} \mathcal{N}(U_i|0,\sigma_U^2\mathbf{I}), \qquad p(V|\sigma_V^2) = \prod_{j=1}^{M} \mathcal{N}(V_j|0,\sigma_V^2\mathbf{I})$$

$$p(R|U,V,\sigma^2) = \prod_{i=1}^{N}\prod_{j=1}^{M}\left[\mathcal{N}(R_{ij}|U_i^TV_j,\sigma^2)\right]^{I_{ij}}$$

$\sigma^2,\sigma_u^2,\sigma_v^2$: hyper–parameters

Mnih, A., & Salakhutdinov, R. R. (2007). Probabilistic matrix factorization. Advances in neural information processing systems, 20.

Probabilistic Matrix Factorization

$$p(U,V \mid R, \sigma^2, \sigma_V^2, \sigma_U^2)$$

$$= \frac{p(R \mid U,V,\sigma^2)\, p(U,V \mid \sigma_U^2, \sigma_V^2)}{p(R \mid \sigma^2)} \quad (\because \text{Bayes Rule})$$

$$= \frac{p(R \mid U,V,\sigma^2)\, p(U \mid \sigma_U^2)\, p(V \mid \sigma_V^2)}{p(R \mid \sigma^2)} \quad (\because \text{Independence})$$

Here,

$$p(R \mid U,V,\sigma^2) = \prod_{i=1}^{N} \prod_{j=1}^{M} \left[\frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{1}{2\sigma^2} (U_i^T V_j - R_{ij})^2 \right) \right]^{I_{ij}}$$

$$p(U \mid \sigma_U^2) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\,\sigma_U} \exp\left(-\frac{1}{2\sigma_U^2} U_i^T U_i \right)$$

$$p(V \mid \sigma_V^2) = \prod_{j=1}^{M} \frac{1}{\sqrt{2\pi}\,\sigma_V} \exp\left(-\frac{1}{2\sigma_V^2} V_j^T V_j \right)$$

$$\therefore \ln p(U,V \mid R, \sigma^2, \sigma_V^2, \sigma_U^2)$$

$$= \ln p(R \mid U,V,\sigma^2) + \ln p(U \mid \sigma_U^2) + \ln p(V \mid \sigma_V^2)$$

$$- \ln p(R \mid \sigma^2)$$

Mnih, A., & Salakhutdinov, R. R. (2007). Probabilistic matrix factorization. Advances in neural information processing systems, 20.

Probabilistic Matrix Factorization

$$\ln p(R|U,V,\sigma^2) = \ln\left(\prod_{i=1}^{N}\prod_{j=1}^{M}\left[\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2\sigma^2}(U_i^TV_j-R_{ij})^2\right)\right]^{I_{ij}}\right)$$

$$= \sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}\ln\left(\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2\sigma^2}(U_i^TV_j-R_{ij})^2\right)\right)$$

$$= -\frac{1}{2\sigma^2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}(U_i^TV_j-R_{ij})^2 + \sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}\ln\left(\frac{1}{\sqrt{2\pi}\sigma}\right)$$

$$= -\frac{1}{2\sigma^2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}(U_i^TV_j-R_{ij})^2 - \frac{1}{2}\left(\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}\right)\ln\sigma^2$$

$$-\frac{1}{2}\left(\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}\right)\ln(2\pi)$$

Mnih, A., & Salakhutdinov, R. R. (2007). Probabilistic matrix factorization. Advances in neural information processing systems, 20.

Probabilistic Matrix Factorization

$$\ln p(U,V|R,\sigma^2,\sigma_V^2,\sigma_U^2) = -\frac{1}{2\sigma^2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}(R_{ij}-U_i^T V_j)^2 - \frac{1}{2\sigma_U^2}\sum_{i=1}^{N} U_i^T U_i - \frac{1}{2\sigma_V^2}\sum_{j=1}^{M} V_j^T V_j$$
$$-\frac{1}{2}\left(\left(\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}\right)\ln\sigma^2 + ND\ln\sigma_U^2 + MD\ln\sigma_V^2\right) + C, \qquad (3)$$

Final loss term that depends on U and V

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

, where $\lambda_U = \sigma^2/\sigma_U^2$, $\lambda_V = \sigma^2/\sigma_V^2$

Loss is backpropagated to optimize U and V by SGD

Mnih, A., & Salakhutdinov, R. R. (2007). Probabilistic matrix factorization. Advances in neural information processing systems, 20.

Probabilistic Matrix Factorization

A1. Automatic Complexity Control for PMF Models

Instead of using hyper-parameters, the machine adaptively modifies them while training

1. Gaussian Priors
   : Find optimal hyper-parameters by closed form solution

2. Mixture of Gaussian Priors
   : Find optimal hyper-parameters by expectation maximization

$$\ln p(U, V, \sigma^2, \Theta_U, \Theta_V | R) = \ln p(R|U, V, \sigma^2) + \ln p(U|\Theta_U) + \ln p(V|\Theta_V) + \\ \ln p(\Theta_U) + \ln p(\Theta_V) + C,$$

Mnih, A., & Salakhutdinov, R. R. (2007). Probabilistic matrix factorization. Advances in neural information processing systems, 20.

Probabilistic Matrix Factorization

A2. Constrained PMF

PMF model will tend to make users with few ratings near the mean values
Thus, add $W \in \mathbb{R}^{D \times M}$: latent similarity matrix to add some bias

$$U_i = Y_i + \frac{\sum_{k=1}^{M} I_{ik} W_k}{\sum_{k=1}^{M} I_{ik}}.$$

$$p(R|Y,V,W,\sigma^2) = \prod_{i=1}^{N}\prod_{j=1}^{M}\left[\mathcal{N}(R_{ij}|g([Y_i + \frac{\sum_{k=1}^{M} I_{ik} W_k}{\sum_{k=1}^{M} I_{ik}}]^T V_j), \sigma^2)\right]^{I_{ij}}$$

$$\begin{aligned} E &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}\left(R_{ij} - g([Y_i + \frac{\sum_{k=1}^{M} I_{ik} W_k}{\sum_{k=1}^{M} I_{ik}}]^T V_j)\right)^2 \\ &+ \frac{\lambda_Y}{2}\sum_{i=1}^{N} \| Y_i \|_{Fro}^2 + \frac{\lambda_V}{2}\sum_{j=1}^{M} \| V_j \|_{Fro}^2 + \frac{\lambda_W}{2}\sum_{k=1}^{M} \| W_k \|_{Fro}^2, \end{aligned}$$

Mnih, A., & Salakhutdinov, R. R. (2007). Probabilistic matrix factorization. Advances in neural information processing systems, 20.

Probabilistic Matrix Factorization

Experiments

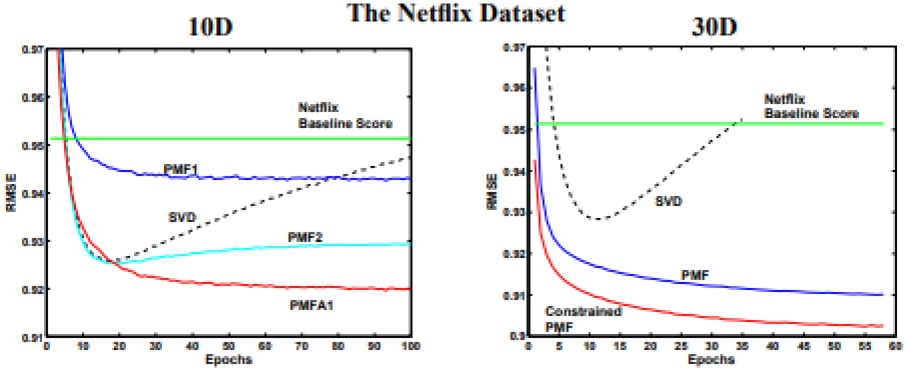


Figure 2: Left panel: Performance of SVD, PMF and PMF with adaptive priors, using 10D feature vectors, on the full Netflix validation data. Right panel: Performance of SVD, Probabilistic Matrix Factorization (PMF) and constrained PMF, using 30D feature vectors, on the validation data. The y-axis displays RMSE (root mean squared error), and the x-axis shows the number of epochs, or passes, through the entire training dataset.

Implementation

0. Data preparation

```
with open("u1.base") as i_file:
    for line in i_file:
        id, *others = map(int, line.split())
        user[id] += 1
        if id > 50:
            base_file.write(line)
        elif user[id] <= 5:
            base_file.write(line)
        else:
            lines.append(line)

with open("u1.test") as i_file:
    for line in i_file:
        id, *others = map(int, line.split())
        if id <= 50:
            lines.append(line)
        else:
            test2.write(line)
```

```
import torch
from torch.utils.data import Dataset

class movie_lens(Dataset):
    def __init__(self, path, split='train'):
        self.path = path
        self.read_meta()

    def read_meta(self):
        self.all_users = []
        self.all_movies = []
        self.all_ratings = []

        with open(self.path, 'r') as f:
            for line in f:
                user, movie, rating, *others = map(int, line.split())
                self.all_users += [user]
                self.all_movies += [movie]
                self.all_ratings += [(rating-1)/4]

    def __len__(self):
        return len(self.all_ratings)

    def __getitem__(self, idx):
        sample = {
            'user': self.all_users[idx],
            'movie': self.all_movies[idx],
            'rating': self.all_ratings[idx]
        }

        return sample
```

Implementation

1. PMF / CPMF

```
V = torch.rand(size=(args.D, args.M), device=device, requires_grad=True)
Y = torch.rand(size=(args.D, args.N), device=device, requires_grad=True)
if args.constrained == False:
    W = torch.zeros_like(V, device=device)
else:
    W = torch.rand(size=(args.D, args.M), device=device, requires_grad=True)
```

$$U_i = Y_i + \frac{\sum_{k=1}^{M} I_{ik} W_k}{\sum_{k=1}^{M} I_{ik}}.$$

2. PMF / PMFA

```
if args.adaptive and i%args.N_a == 0:
    lu.requires_grad = True
    lv.requires_grad = True
    V.requires_grad = False
    Y.requires_grad = False
    if args.constrained:
        W.requires_grad = False
else:
    lu.requires_grad = False
    lv.requires_grad = False
    V.requires_grad = True
    Y.requires_grad = True
    if args.constrained:
        W.requires_grad = True
```

Implementation

3. Calculate loss

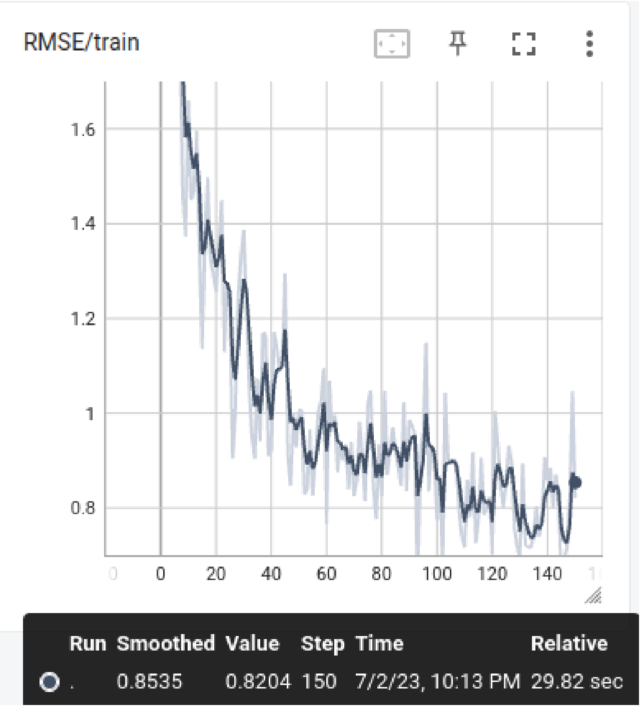
$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}\left(R_{ij} - U_i^T V_j\right)^2 + \frac{\lambda_U}{2}\sum_{i=1}^{N} \| U_i \|_{Fro}^2 + \frac{\lambda_V}{2}\sum_{j=1}^{M} \| V_j \|_{Fro}^2,$$

```
ratings_loss = lambda x, y: .5 * torch.sum((x - y)**2)
```

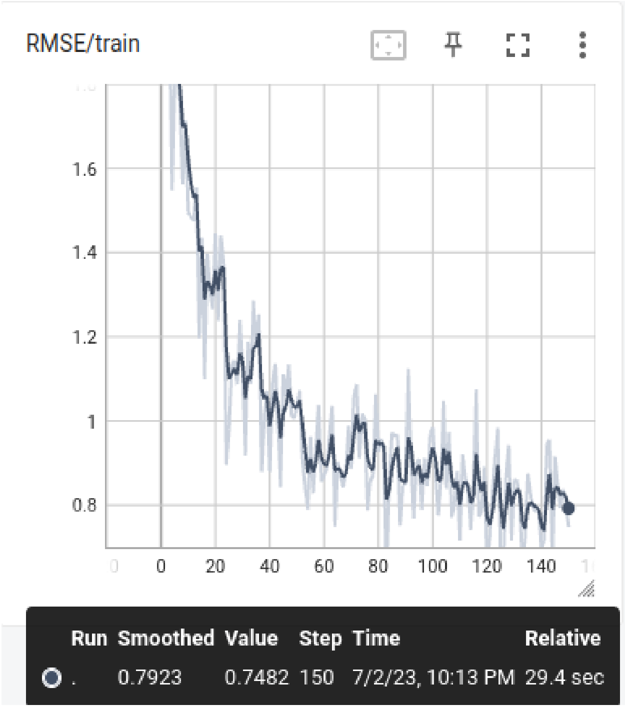
```
for batch in dataloader:
    optimizer.zero_grad()

    users = batch['user'].to(device)
    movies = batch['movie'].to(device)
    ratings = batch['rating'].to(device)

    Y_= Y[:, users]
    W_ = W[:, movies]
    U = Y_ + W_
    U = torch.transpose(U, 0, 1) # U : (batch_size) * D
    V_ = V[:, movies]            # V_ : D * (batch_size)

    UTV = torch.sigmoid(torch.diagonal(torch.mm(U, V_))).to(device)


 loss = ratings_loss(UTV, ratings) + 0.5 * lu * (torch.norm(Y_, p='fro', dim=0)**2).sum() \
                                   + 0.5 * lv * (torch.norm(V_, p='fro', dim=0)**2).sum() \
                                   + 0.5 * lw * (torch.norm(W_, p='fro', dim=0)**2).sum()
 loss.requires_grad_(True)
 loss.backward()
 optimizer.step()
```

Implementation

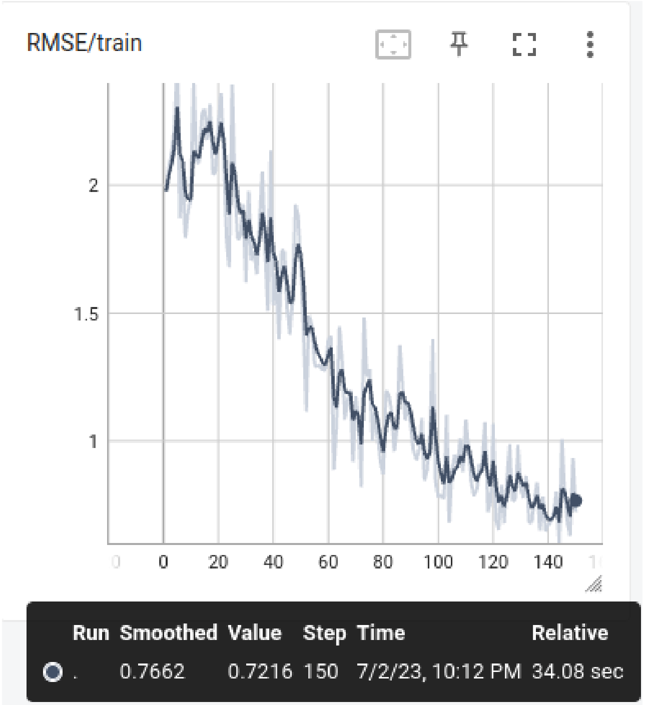
4. Training
Optimizer : torch.optim.SGD, lr = 5e−3, momentum = .9
D = 30



PMF
lu = 1e−2, lv = 1e−3



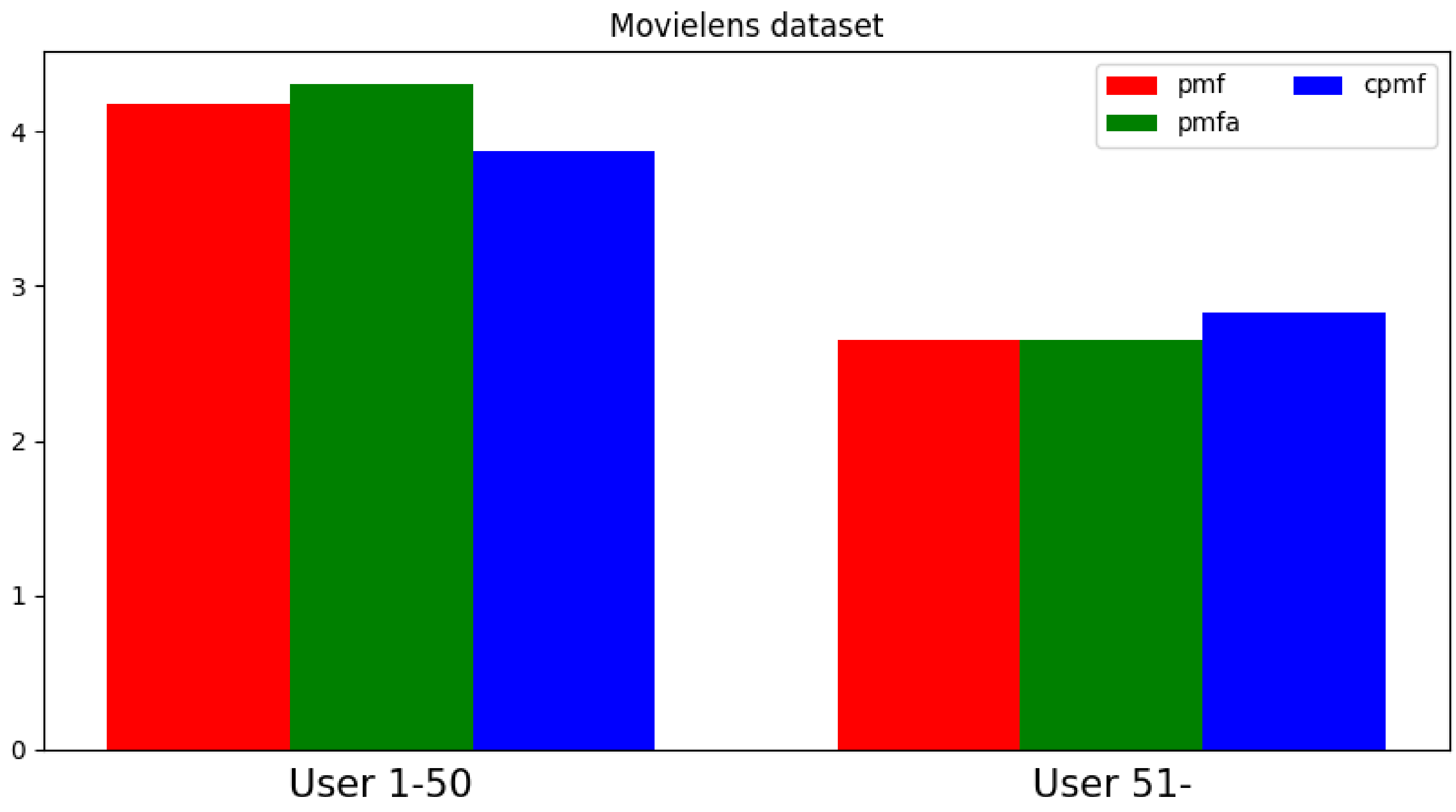
PMFA
lu = 1e−2, lv = 1e−3



CPMF
lu = 2e−2, lv = 2e−2, lw = 2e−2

Implementation

4. Test



Implementation

5. Review

(+)
 – Followed direct implementations (e.g : lr, SGD, Sigmoid …)
 – RMSE similar to the paper

(–)
 – Performed batch–wise training, thus, no I matrix
 – Didn't calculate closed form solution at PMFA