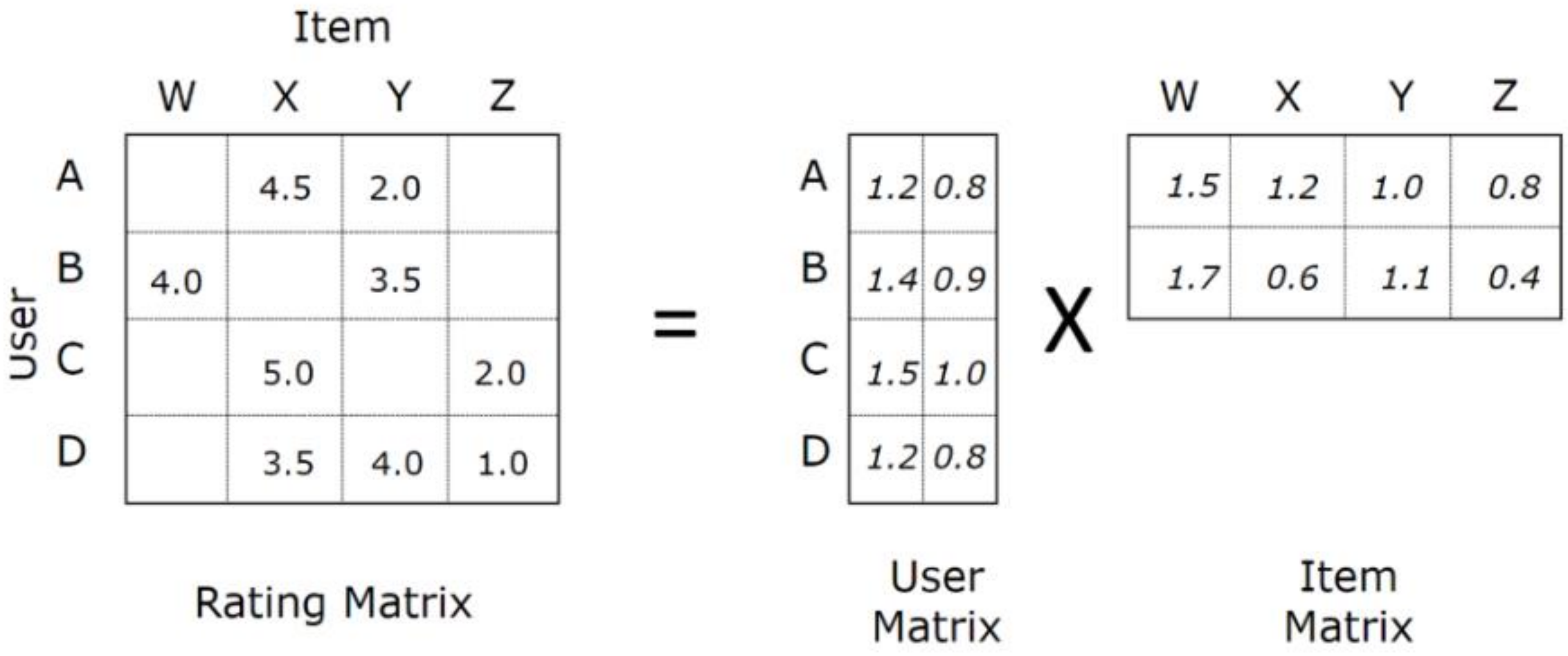
Probabilistic Matrix Factorization

Ruslan Salakhutdinov and Andriy Mnih / NeurIPS

Jaeheong Jeon

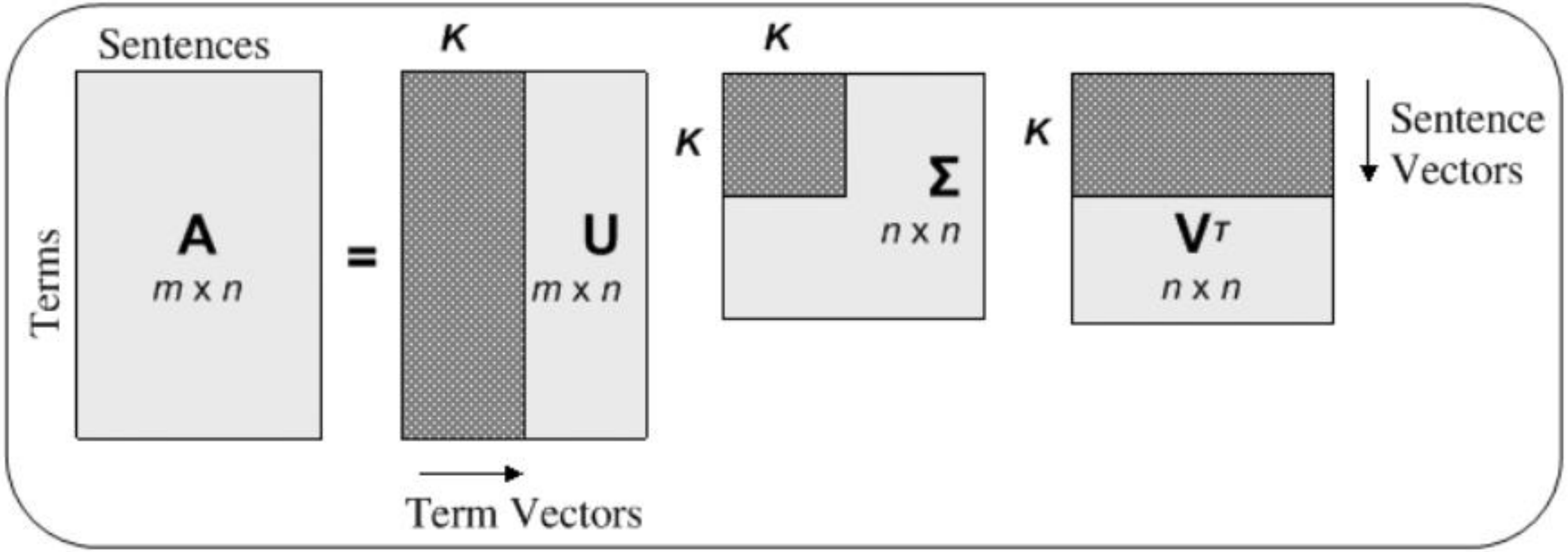
Introduction [collaborative filtering using Matrix factorization]

- Matrix factorization techniques are a class of widely successful Latent Factor models that attempt to find weighted low-rank approximations.



Introduction [SVD]

- SVD finds the matrix Rˆ = U^T V of the given rank which minimizes the sum-squared distance to the target matrix R.



Introduction

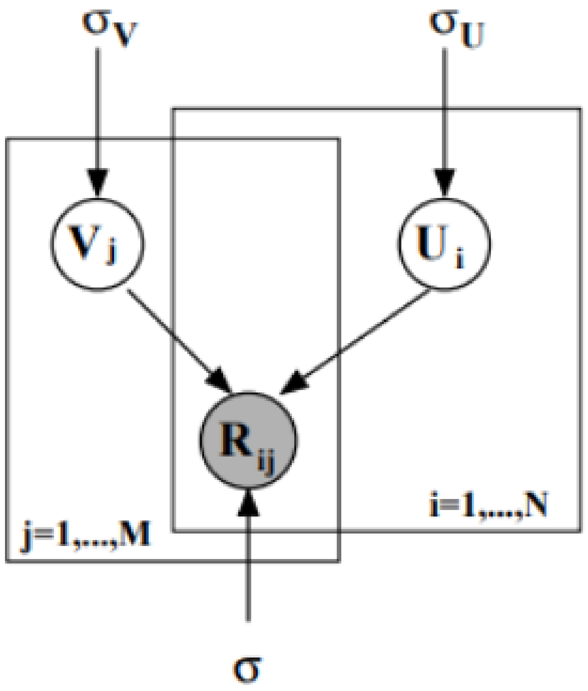
Two main problem of traditional approach

- First, none of the mentioned approaches, except for the matrix-factorization-based ones, scale well to large datasets.

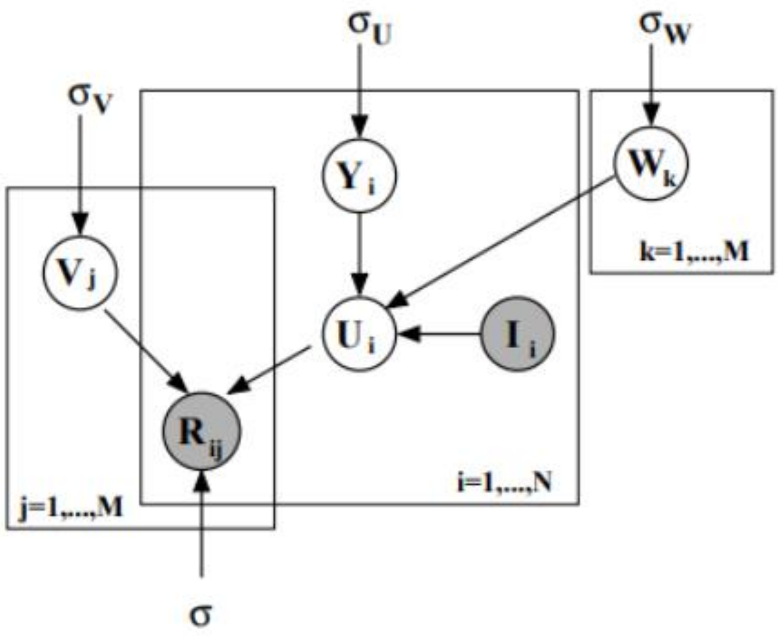
- Second, most of the existing algorithms have trouble making accurate predictions for users who have very few ratings.

Introduction

- Probabilistic Matrix Factorization (PMF)



- Constrained PMF



Proposed Method [Probabilistic Matrix Factorization (PMF)]

- Suppose we have M movies, N users, and integer rating values from 1 to K .
- Let R_ij represent the rating of user i for movie j, U and V be latent user and movie feature matrices

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

- We also place zero-mean spherical Gaussian priors on user and movie feature vectors:

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

Proposed Method [Probabilistic Matrix Factorization (PMF)]

- By Bayes's rule

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

- The log of the posterior distribution over the user and movie features

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

Proposed Method [Probabilistic Matrix Factorization (PMF)]

- The log of the posterior distribution over the user and movie features

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

- Maximizing the log-posterior is equivalent to minimizing the sum-of-squared-errors objective function

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

Proposed Method [Automatic Complexity Control for PMF Model]

- Capacity control is essential to making PMF models generalize well.
- The simplest way is by changing the dimensionality of feature vectors. However, when the dataset is unbalance, this approach fail
- Regularization parameters provide a more flexible approach.

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

- The simplest way to find suitable values is to consider a set of reasonable parameter values, train a model for each setting, and choose the best one.
- The main drawback of this approach is that it is computationally expensive

Proposed Method [Automatic Complexity Control for PMF Model]

- Introducing priors for the hyperparameters and maximizing the log-posterior allows model complexity to be controlled automatically.

$$\begin{aligned}\ln p(U, V, \sigma^2, \Theta_U, \Theta_V | R) = \\ &\ln p(R|U, V, \sigma^2) + \ln p(U|\Theta_U) + \ln p(V|\Theta_V) + \\ &\ln p(\Theta_U) + \ln p(\Theta_V) + C,\end{aligned}$$

- When the prior is Gaussian, using steepest ascent for updating (the optimal hyperparameters can be found in closed form)

- When the prior is a mixture of Gaussians, using EM.

Proposed Method [Constrained PMF]

- The way of constraining user-specific feature vectors that has a strong effect on infrequent users.

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

- Let $W \in R^{D \times M}$ be a latent similarity constraint matrix
- Y_i can be seen as the offset added to the mean

$$p(W|\sigma_W) = \prod_{k=1}^{M} \mathcal{N}(W_k|0, \sigma_W^2 \mathbf{I})$$

Proposed Method [Constrained PMF]

- The way of constraining user-specific feature vectors that has a strong effect on infrequent users.

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

- Original PMF

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

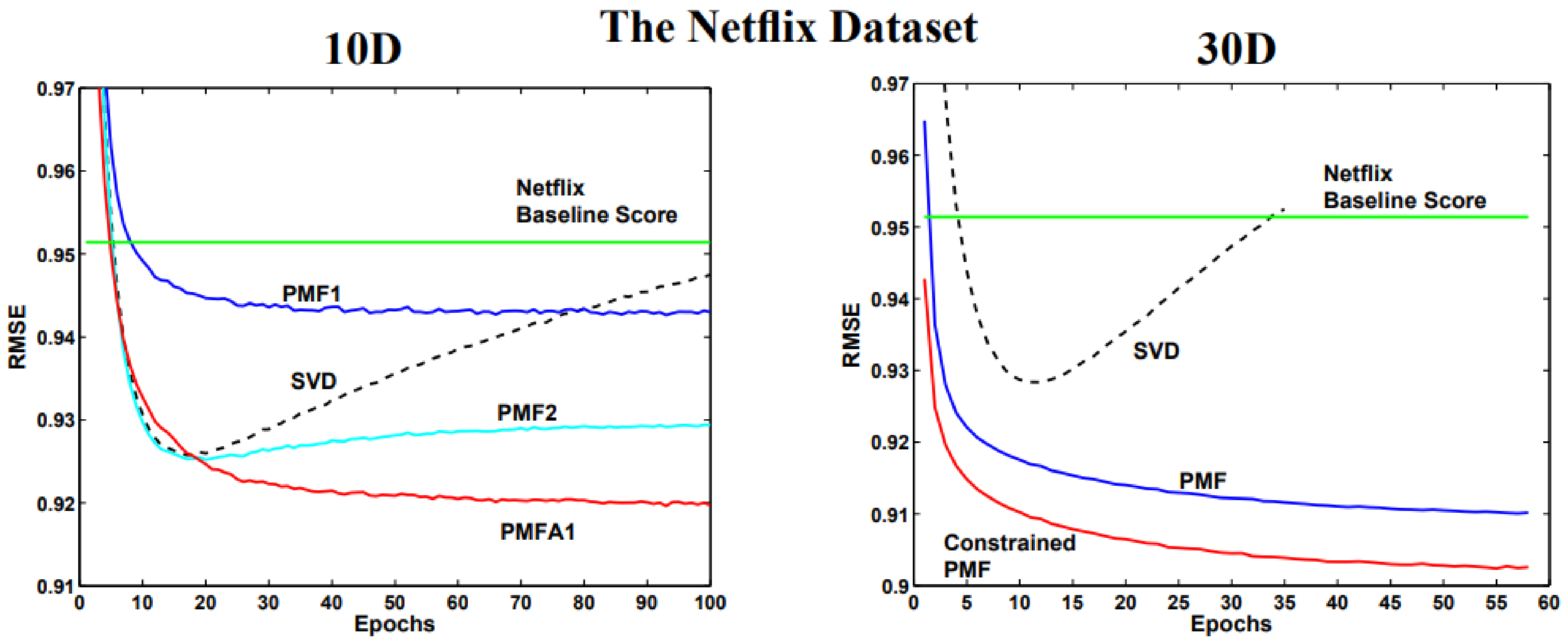
Experiment [Description of the Netflix Data]

- The data were collected between October 1998 and December 2005
- The training dataset consists of 100,480,507 ratings from 480,189 anonymous users on 17,770 movie title

[user_id]	[movie_id]	[rating]	[timestamp]
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596
298	474	4	884182806
115	265	2	881171488
253	465	5	891628467
[illegible]	[illegible]	[illegible]	[illegible]

u.data

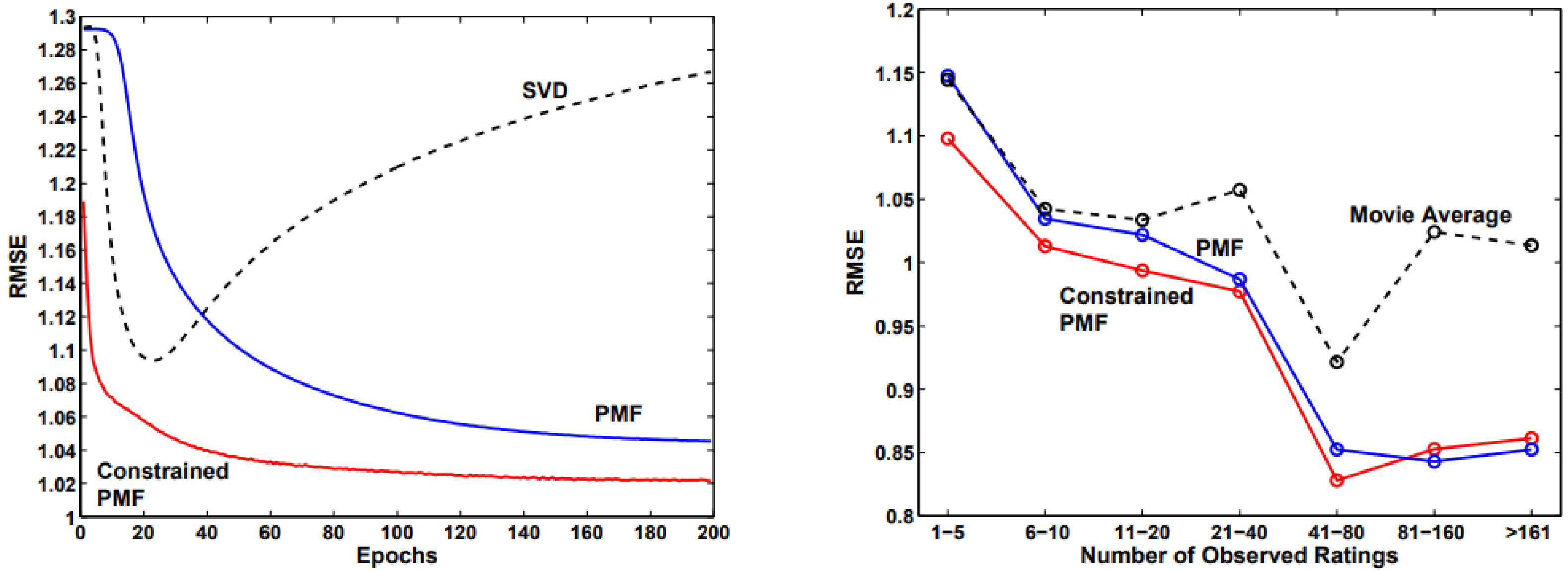
Experiment [Results for PMF with Adaptive Priors]



- PMF1(λu=0.01, λv=0.001)   PMF2(λu=0.001, λv=0.0001)
- PMFA1(λu=0.01, λv=0.001, spherical covariance matrices)
- PMFA2(λu=0.01, λv=0.001, diagonal covariance matrices)

Experiment [Results for Constrained PMF]

Toy Dataset



- Performance of the PMF model for users that have fewer than 5 ratings is almost identical to movie average
- The constrained PMF model performs better on users with few ratings

Conclusion

- This paper present Probabilistic Matrix Factorization (PMF) and its two derivatives: PMF with a learnable prior and constrained PMF.
- Efficiency in training PMF models comes from finding only point estimates of model parameters and hyperparameters
- If we were to take a fully Bayesian approach, we would put hyperpriors over the hyperparameters and resort to MCMC methods

Implementation

```
class PMF():

    def __init__(self, R, val_R, latent_size=50, ld=1e-3, lr=0.001, epochs=200, constrain=False):

        self._R = R
        self._val_R = val_R
        self._N, self._M = R.shape
        self._epochs =epochs
        self._lr = lr
        self._I = copy.deepcopy(self._R)
        self._I[self._I != 0] = 1
        self._val_I = copy.deepcopy(self._val_R)
        self._val_I[self._val_I != 0] = 1
        self._U = np.random.normal(0, 0.1, size=(self._N, latent_size))
        self._V = np.random.normal(0, 0.1, size=(self._M, latent_size))
        self._lambda = ld
        self._constrain = constrain
        if constrain:
            self._W = np.random.normal(0, 0.1, size=(self._M, latent_size))
```

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

Implementation

- Calculate gradient of PMF

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

```python
def get_grad_uncon(self):
    # derivate of U
    grads_u = np.dot(self._I*(self._R-np.dot(self._Y, self._V.T)), -self._V) + self._lambda*self._Y

    # derivate of V
    grads_v = np.dot((self._I*(self._R-np.dot(self._Y, self._V.T))).T, -self._Y) + self._lambda*self._V
    return grads_u, grads_v
```

Implementation

- Calculate gradient of constrain_PMF

$$\frac{\sum_{k=1}^{M} I_{ik} W_k}{\sum_{k=1}^{M} I_{ik}}$$

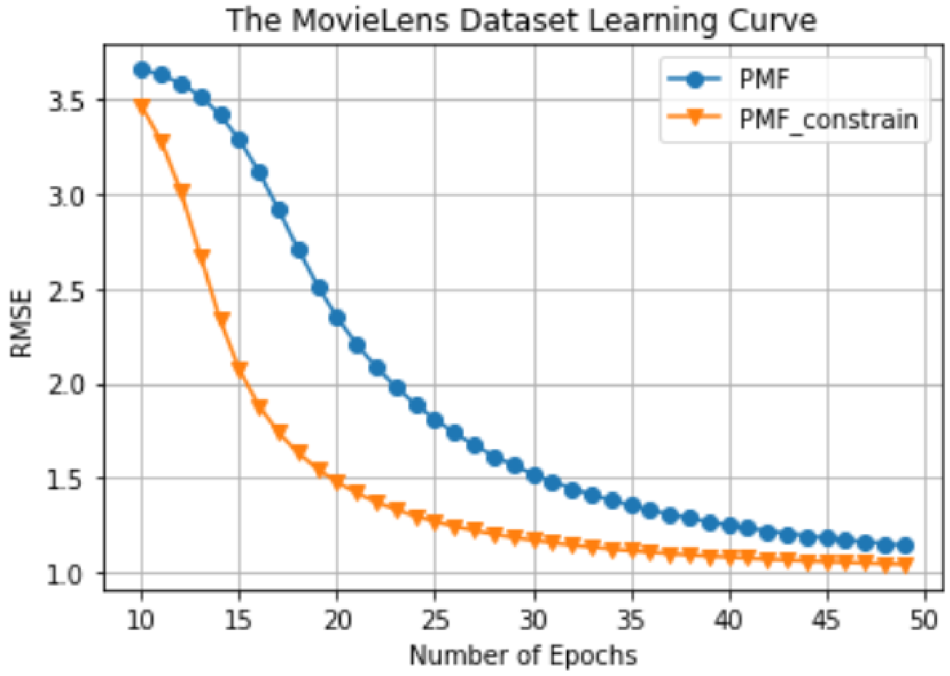
```
uw = np.zeros(self._Y.shape)
for i in range(self._N):
    uw[i, :] = np.matmul(self._I[i, :],self._W) / np.sum(self._I[i, :])
```

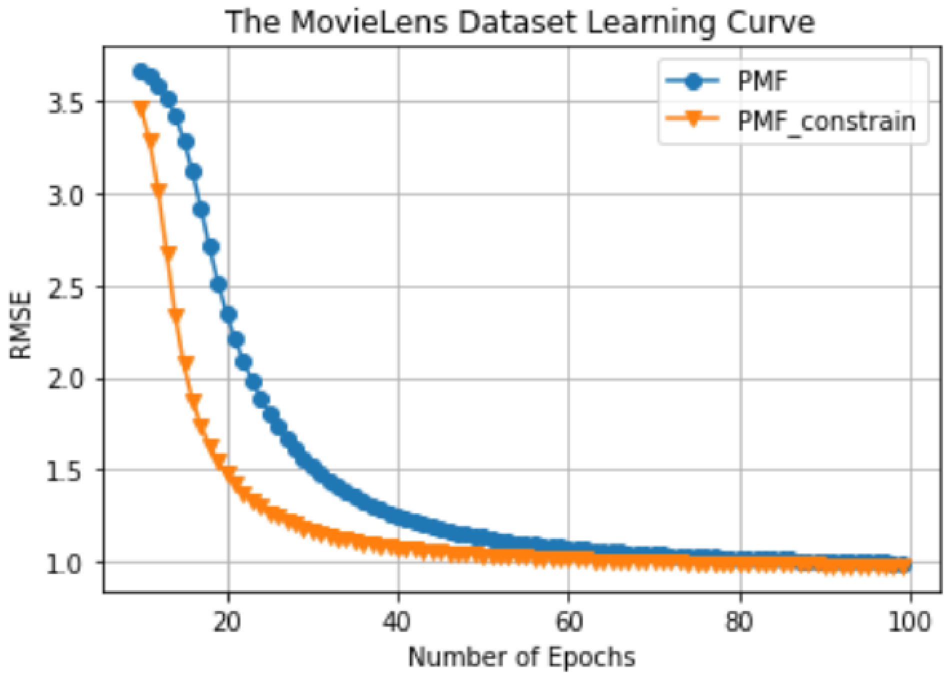
```
pred = np.dot(self._Y+uw, self._V.T)
gd_uw = np.zeros(self._R.shape)
for i in range(self._N):
    gd_uw[i, :] = self._I[i, :] / np.sum(self._I[i, :])
```

```
# derivate of V
grads_w = np.dot((self._I*(self._R-pred)).T, -np.dot(gd_uw, self._V)) +self._lambda*self._W
```

Implementation

- Comparing PMF and constrained PMF





감사합니다