Probabilistic Matrix Factorization Review

이수찬
2024.01.03

Contents

1. Introduction

- A common method for collaborative filtering is a low-dimensional factor model (based on Matrix F actorization)
- Recently, various probabilistic factor-based models have been proposed
  - The biggest drawback of these models is that they are difficult to reason accurately, so they use approximations to calculate posterior probability
- A low-rank approximation based on minimizing the sum-squared distance can be thought of as an SVD
  - SVD finds the matrix that minimizes the sum-squared distance for Rating matrix (R)
  - Most real-world datasets are sparse, meaning that most of the entries in R may be missing
  - In such cases, compute sum-squared distance only for observed entries in R
    - This seemingly trivial modification introduces a difficult non-convex optimization problem that cannot be solved by standard SVD implementations*

Nathan Srebro and Tommi Jaakkola. Weighted low-rank approximations. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 720–727. AAAI Press, 2003.

1. Introduction

- So what's the alternative?
  - Propose to limit the latent factor of the observed R, i.e., the norm of the user feature matrix (U) and item feature matrix (V) that result from decomposing R instead of making a low-order ap proximation*
    - However, learning from this model requires solving a sparse semi-definite program (SDP), which makes this approach impractical for datasets with millions of observations

    → Computer resource issue

1. Introduction

- Attempted to model user ratings on the Netflix Price dataset with the existing approaches, but mo stly failed for two reasons
  - Doesn't scale well to large data sets except matrix factorization
  - Difficulty making accurate predictions for users with very low ratings (sparse matrix)

- Goal...
  - The goal is to present a probabilistic algorithm that scales linearly with the number of observ ations (applicable to large data) and performs well on sparse and imbalanced datasets
  - This paper introduces PMF, PMF with adaptive prior, Constrained PMF

2. Probabilistic Matrix Factorization

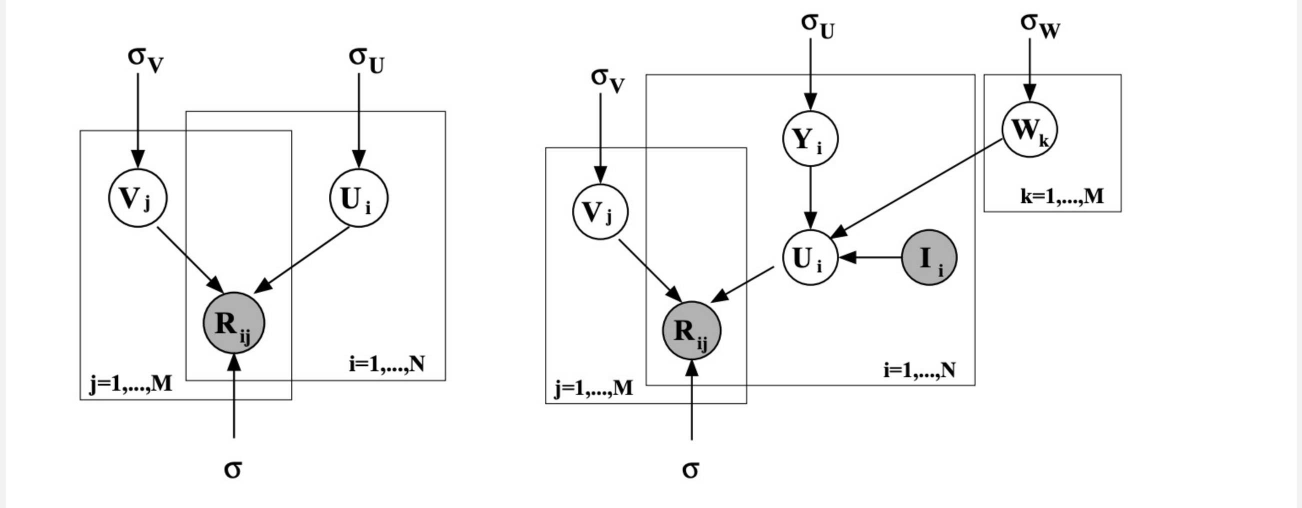


Figure 1: The left panel shows the graphical model for Probabilistic Matrix Factorization (PMF). The right panel shows the graphical model for constrained PMF.

- M movies, N users, ratings ranging from 1 to K
- U is latent user D × N feature matrix,
- V is latent item D × Mfeature matrix
- Model performance is calculated with RMSE

Adopt a probabilistic linear model with Gaussian observation noise

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

Set the user latent feature matrix an d item latent feature matrix to follow a spherical Gaussian distribution wi th zero mean

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

2. Probabilistic Matrix Factorization

- To get the posterior distribution with log, follow the steps below

$$
\begin{aligned}
& P(U,V|R,\sigma^2,\sigma_U^2,\sigma_V^2) \\
& = \frac{P(U,V,R|\sigma^2,\sigma_U^2,\sigma_V^2)}{P(R|\sigma^2,\sigma_U^2,\sigma_V^2)} \\
& \propto P(U,V,R|\sigma^2,\sigma_U^2,\sigma_V^2) \\
& = P(R|U,V,\sigma^2)P(V|\sigma_V^2)P(U|\sigma_U^2) \\
& \propto [\Pi_{ij} exp(-\frac{(R_{ij}-[UV]_{ij})}{2\sigma^2})][\Pi_{i,k} exp(-\frac{U_{i,k}^2}{2\sigma_U^2})][\Pi_{j,k} exp(-\frac{R_{j,k}^2}{2\sigma_R^2})]
\end{aligned}
$$

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

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

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

- Use Gradient Descent to find local minimum of loss function
- The expression on the left can be vi ewed as a probabilistic extension o f the SVD model

2. Probabilistic Matrix Factorization

- In this paper, instead of a linear-Gaussian model, we pass the dot product of user and movie specific feature vectors through a logistic function to limit the prediction range

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

- Additionally, rating is mapped from [1, K] to [0, 1] via the following function

$$t(x) = (x-1)/(K-1)$$

- Minimizing the objective function given above using steepest descent has a time complexity that is lin ear in the number of observers
  - A simple implementation of this algorithm in Matlab can sweep the entire Netflix dataset once in l ess than an hour if the model being trained has 30 latent factors

3. Automatic Complexity Control for PMF Models

- Capacity control is essential for PMF models to generalize well
  - If the number of factors is moderate, PMF can provide a good enough approximation
- The simplest way to control the capacity of a PMF model is to change the dimensionality of the fea ture vector
  - However, if the dataset is unbalanced, this approach fails because the dimensionality of some feature vectors is too high and others are too low
- Flexible regularization through parameter adjustment in E
- The simplest way to find good values for these parameters is to consider a reasonable set of par ameter values, train a model for each parameter setting in the set, and select the best performing model on the validation set → computational cost issue
- Using the method proposed in *, which was originally applied to neural networks, we can automati cally determine suitable values for the regularization parameters of the PMF model without signifi cantly affecting the time required to train the model

S. J. Nowlan and G.E.Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4:473–493, 1992.

3. Automatic Complexity Control for PMF Models

- Summary…
  - Find the maximum value of the log-posterior to get the point estimate and hyperparameter of the parameter, as shown below

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

4. Constrained PMF

- When the PMF model is applied, users with fewer ratings will have feature vectors closer to the prior mean or average user, so their predicted ratings will be closer to the average rating of the movie
- Sparse matrix is highly influenced by the prior mean
- Average user will eventually be close to the prior mean
- CPMF → How to limit user-specific feature vectors that strongly influence users with fewer ratings

$U_i = Y_i + \frac{\sum_{k=1}^{M} I_{ik} W_k}{\sum_{k=1}^{M} I_{ik}}, W \in R^{D \times M}$ W is latent similarity matrix

- The i-th column of the W matrix captures how a user's rating of a particular movie affects the prior mean of the user's feature vector
- As a result, users who have seen the same movie will have feature vectors with similar prior distribu tions

4. Constrained PMF

- Modified conditional distribution

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

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

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

$$\lambda_Y = \sigma^2/\sigma_Y^2,\ \lambda_V = \sigma^2/\sigma_V^2,\ \lambda_W^2 = \sigma^2/\sigma_W^2$$

- Find a local minimum via gradient descent, exactly like PMF
- The training time of Constrained PMF is linear in the number of observations, making it fast and simple to implement
- As shown in the results, Constrained PMF performs better than Unconstrained PMF for user s with fewer ratings

5. Experimental Results

- The Netflix data consists of over 100 million ratings from approximately 480,000 users, collected between 1998 and 2005, which were used to train and validate various machine learning models
- Probabilistic Matrix Factorization (PMF) models were tested using fixed and adaptive prior distributions, with models with adaptive prior distributions performing better
- Constrained PMF models performed well, especially for users with low ratings, and combinations of multiple PMF models achieved better performance than the Netflix system

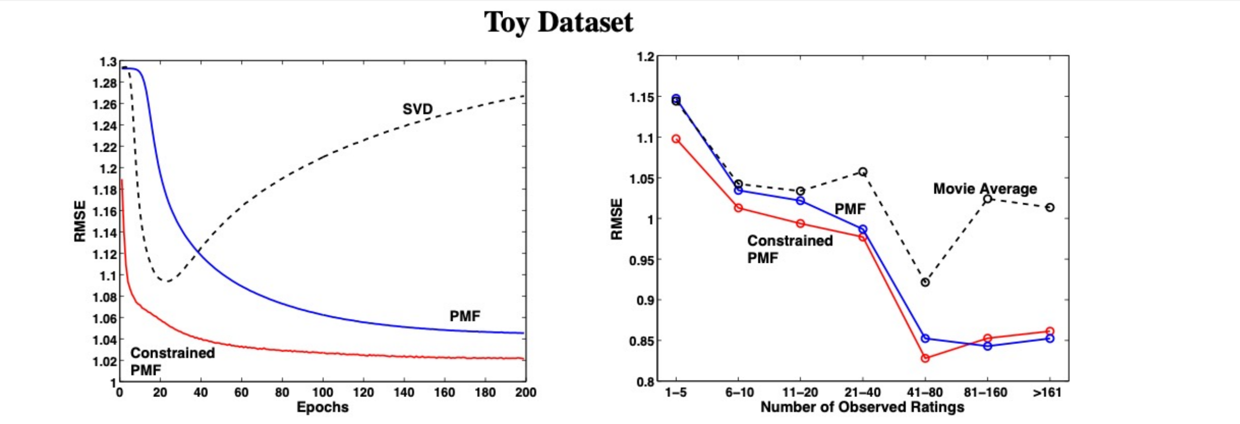


Figure 3: Left panel: Performance of SVD, Probabilistic Matrix Factorization (PMF) and constrained PMF on the validation data. The y-axis displays RMSE (root mean squared error), and the x-axis shows the number of epochs, or passes, through the entire training dataset. Right panel: Performance of constrained PMF, PMF, and the movie average algorithm that always predicts the average rating of each movie. The users were grouped by the number of observed ratings in the training data.

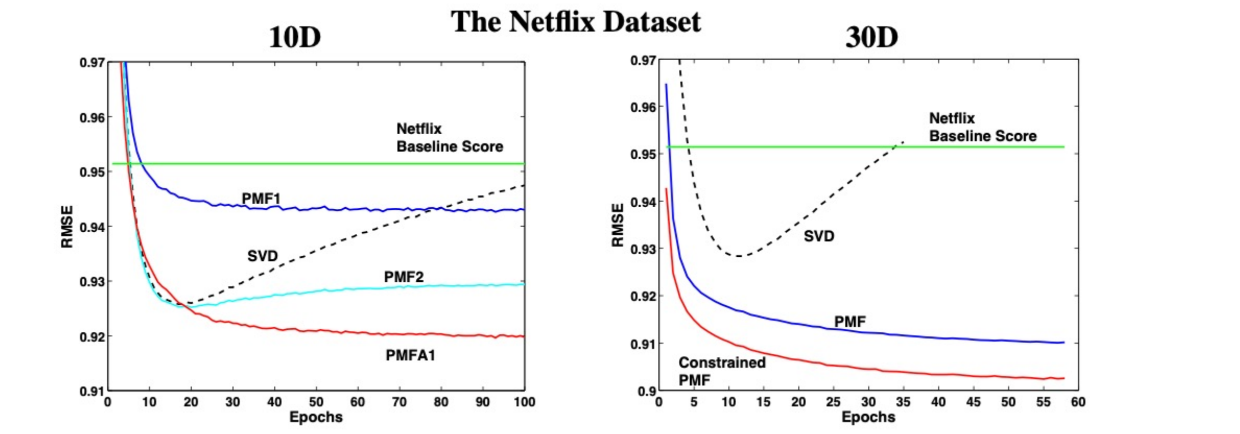


Figure 2: Left panel: Performance of SVD, PMF and PMF with adaptive priors, using 10D feature vectors, on the full Netflix validation data. Right panel: Performance of SVD, Probabilistic Matrix Factorization (PMF) and constrained PMF, using 30D feature vectors, on the validation data. The y-axis displays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

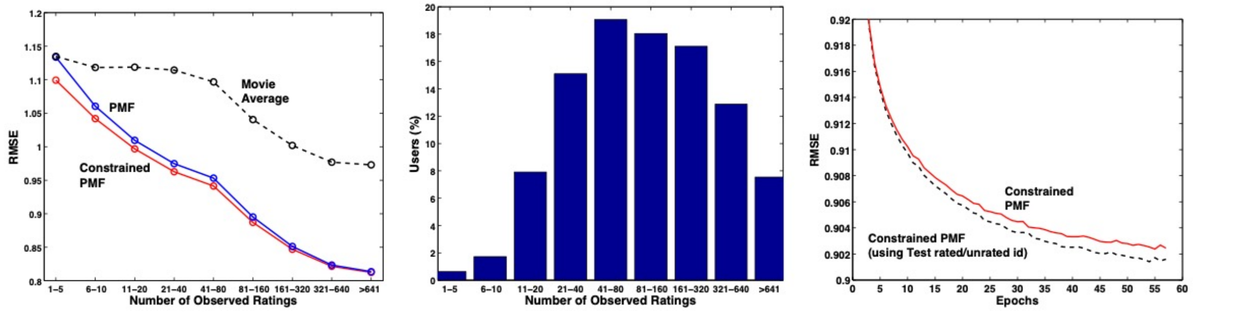
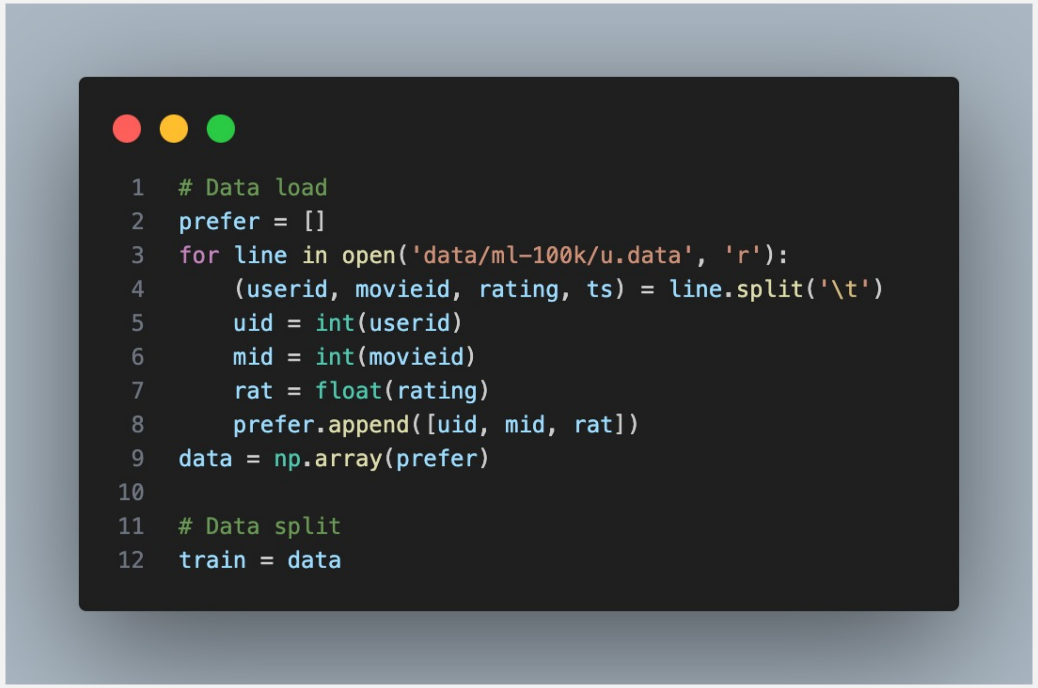


Figure 4: Left panel: Performance of constrained PMF, PMF, and the movie average algorithm that always predicts the average rating of each movie. The users were grouped by the number of observed rating in the training data, with the x-axis showing those groups, and the y-axis displaying RMSE on the full Netflix validation data for each such group. Middle panel: Distribution of users in the training dataset. Right panel: Performance of constrained PMF and constrained PMF that makes use of an additional rated/unrated information obtained from the test dataset.

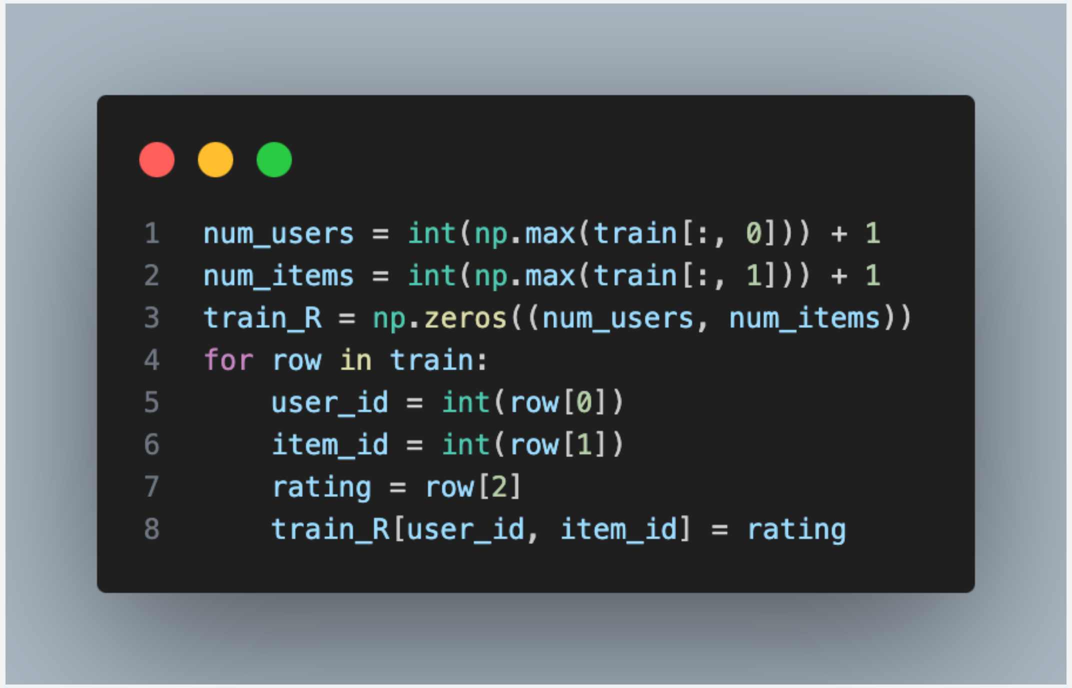
6. Summary and Discussion & Implement

- This paper introduces PMF and two application methods: Adaptive prior PMF & Constrained PMF
- This paper demonstrates that these models can be efficiently trained and successfully applied to a large dataset containing over 100 million movie ratings
- To train PMF models efficiently, we only need to find point estimators instead of inferring the full p rior distribution for model parameters and hyperparameters
- Expect further performance improvements if a full Bayesian approach is used

6. Summary and Discussion & Implement

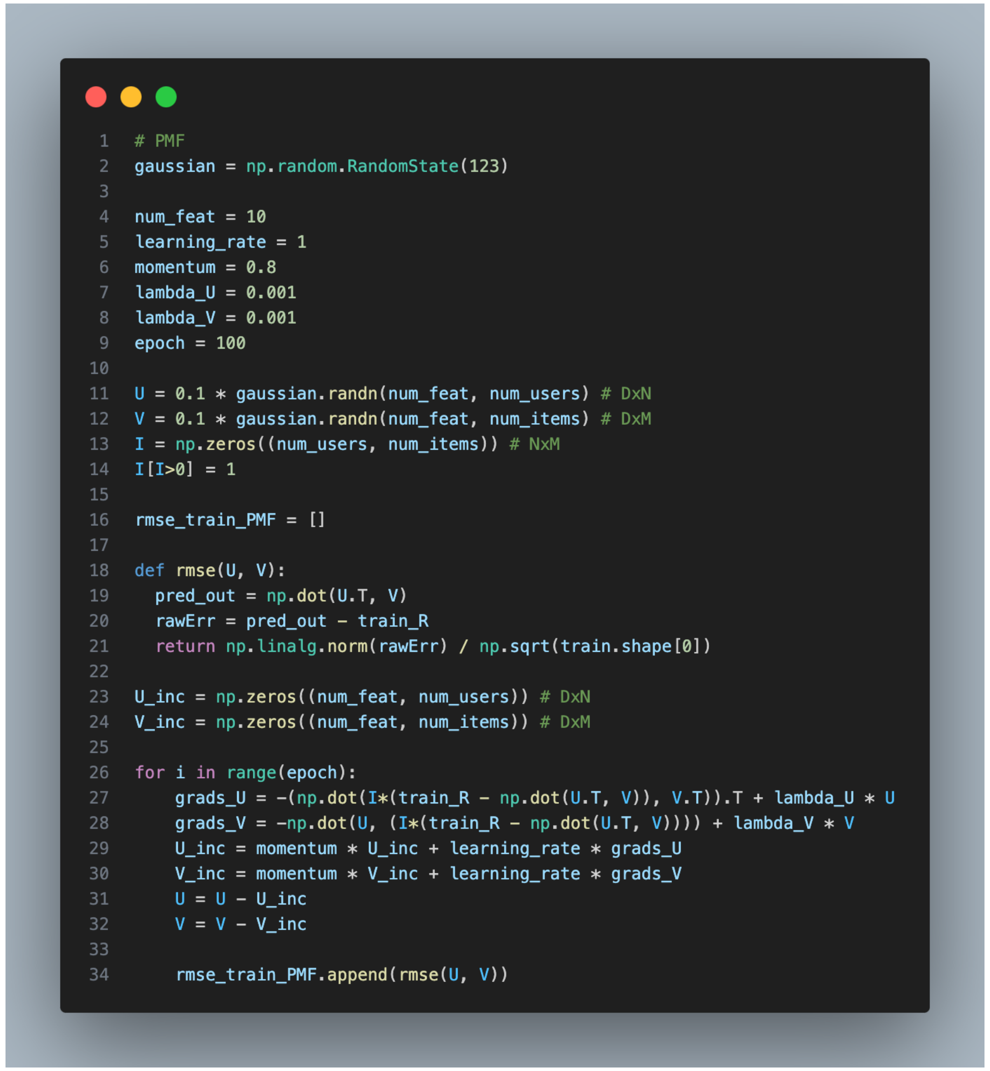


```python
# Data load
prefer = []
for line in open('data/ml-100k/u.data', 'r'):
    (userid, movieid, rating, ts) = line.split('\t')
    uid = int(userid)
    mid = int(movieid)
    rat = float(rating)
    prefer.append([uid, mid, rat])
data = np.array(prefer)

# Data split
train = data
```

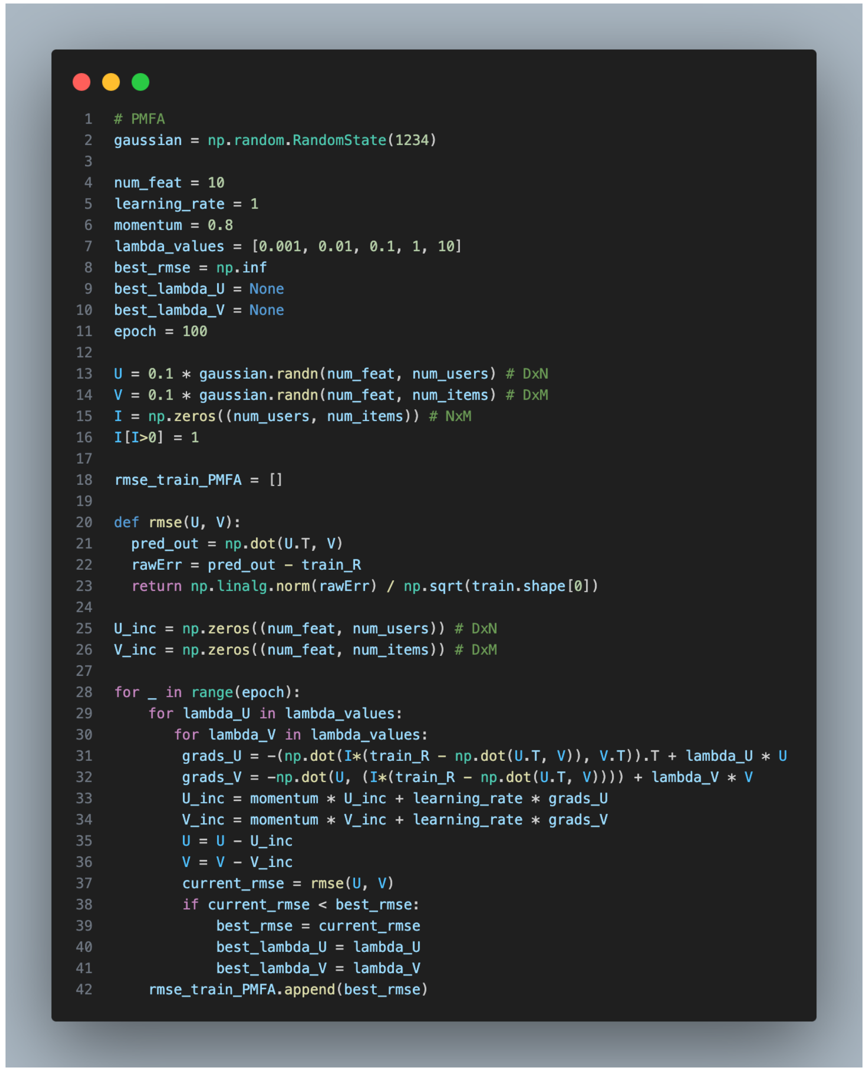


```python
num_users = int(np.max(train[:, 0])) + 1
num_items = int(np.max(train[:, 1])) + 1
train_R = np.zeros((num_users, num_items))
for row in train:
    user_id = int(row[0])
    item_id = int(row[1])
    rating = row[2]
    train_R[user_id, item_id] = rating
```

6. Summary and Discussion & Implement

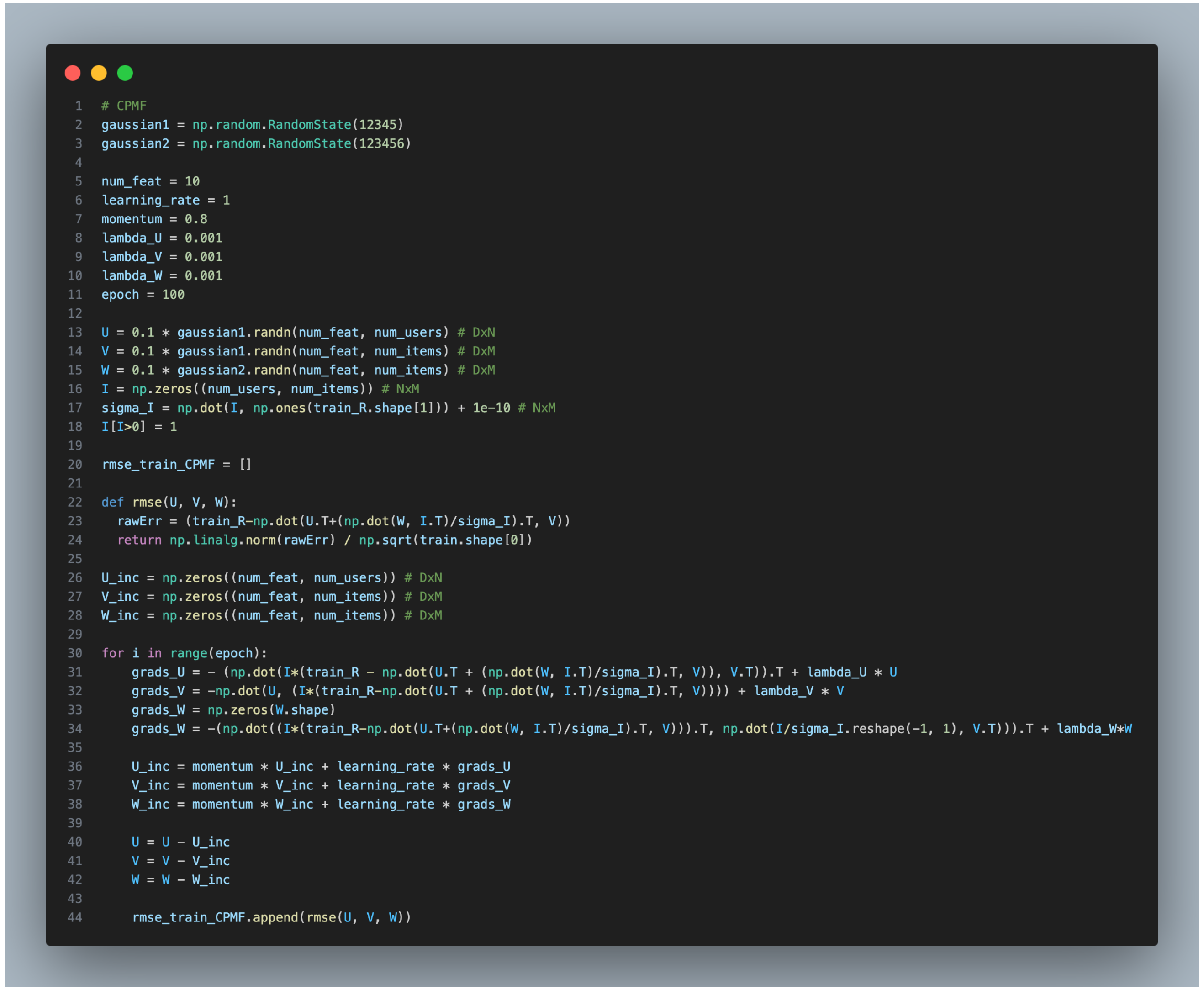


```python
# PMF
gaussian = np.random.RandomState(123)

num_feat = 10
learning_rate = 1
momentum = 0.8
lambda_U = 0.001
lambda_V = 0.001
epoch = 100

U = 0.1 * gaussian.randn(num_feat, num_users) # DxN
V = 0.1 * gaussian.randn(num_feat, num_items) # DxM
I = np.zeros((num_users, num_items)) # NxM
I[I>0] = 1

rmse_train_PMF = []

def rmse(U, V):
  pred_out = np.dot(U.T, V)
  rawErr = pred_out - train_R
  return np.linalg.norm(rawErr) / np.sqrt(train.shape[0])

U_inc = np.zeros((num_feat, num_users)) # DxN
V_inc = np.zeros((num_feat, num_items)) # DxM

for i in range(epoch):
    grads_U = -(np.dot(I*(train_R - np.dot(U.T, V)), V.T)).T + lambda_U * U
    grads_V = -np.dot(U, (I*(train_R - np.dot(U.T, V)))) + lambda_V * V
    U_inc = momentum * U_inc + learning_rate * grads_U
    V_inc = momentum * V_inc + learning_rate * grads_V
    U = U - U_inc
    V = V - V_inc

    rmse_train_PMF.append(rmse(U, V))
```

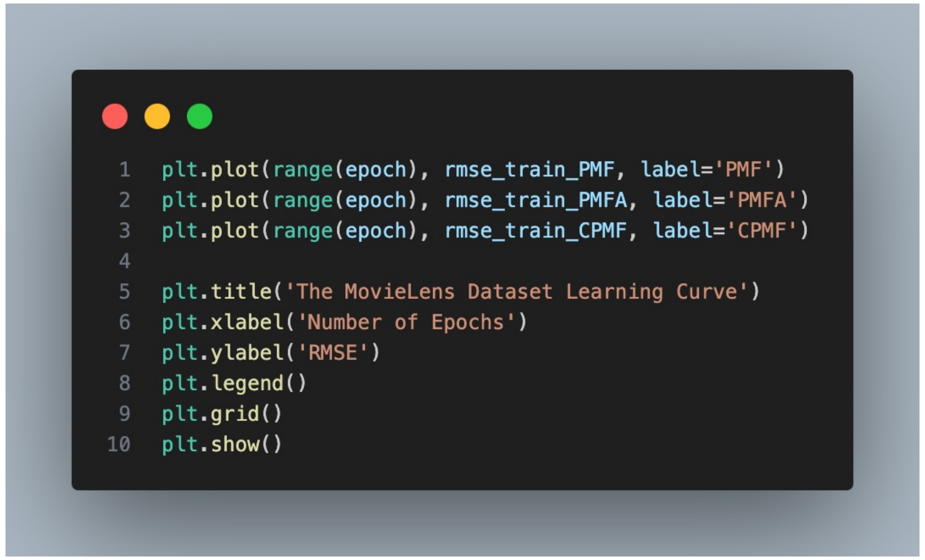


```python
# PMFA
gaussian = np.random.RandomState(1234)

num_feat = 10
learning_rate = 1
momentum = 0.8
lambda_values = [0.001, 0.01, 0.1, 1, 10]
best_rmse = np.inf
best_lambda_U = None
best_lambda_V = None
epoch = 100

U = 0.1 * gaussian.randn(num_feat, num_users) # DxN
V = 0.1 * gaussian.randn(num_feat, num_items) # DxM
I = np.zeros((num_users, num_items)) # NxM
I[I>0] = 1

rmse_train_PMFA = []

def rmse(U, V):
  pred_out = np.dot(U.T, V)
  rawErr = pred_out - train_R
  return np.linalg.norm(rawErr) / np.sqrt(train.shape[0])

U_inc = np.zeros((num_feat, num_users)) # DxN
V_inc = np.zeros((num_feat, num_items)) # DxM

for _ in range(epoch):
    for lambda_U in lambda_values:
       for lambda_V in lambda_values:
        grads_U = -(np.dot(I*(train_R - np.dot(U.T, V)), V.T)).T + lambda_U * U
        grads_V = -np.dot(U, (I*(train_R - np.dot(U.T, V)))) + lambda_V * V
        U_inc = momentum * U_inc + learning_rate * grads_U
        V_inc = momentum * V_inc + learning_rate * grads_V
        U = U - U_inc
        V = V - V_inc
        current_rmse = rmse(U, V)
        if current_rmse < best_rmse:
            best_rmse = current_rmse
            best_lambda_U = lambda_U
            best_lambda_V = lambda_V
    rmse_train_PMFA.append(best_rmse)
```

6. Summary and Discussion & Implement

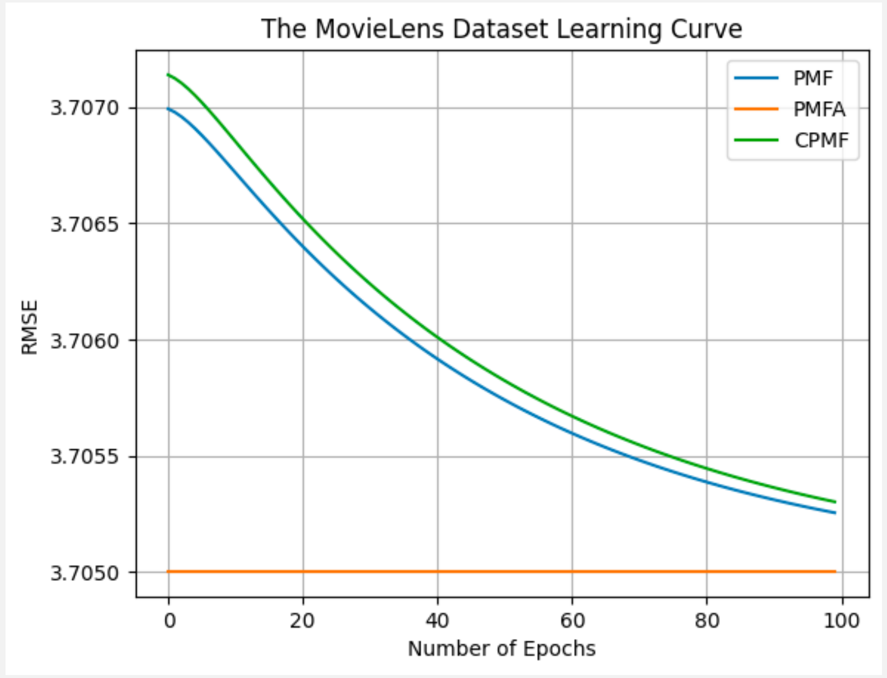


```python
# CPMF
gaussian1 = np.random.RandomState(12345)
gaussian2 = np.random.RandomState(123456)

num_feat = 10
learning_rate = 1
momentum = 0.8
lambda_U = 0.001
lambda_V = 0.001
lambda_W = 0.001
epoch = 100

U = 0.1 * gaussian1.randn(num_feat, num_users) # DxN
V = 0.1 * gaussian1.randn(num_feat, num_items) # DxM
W = 0.1 * gaussian2.randn(num_feat, num_items) # DxM
I = np.zeros((num_users, num_items)) # NxM
sigma_I = np.dot(I, np.ones(train_R.shape[1])) + 1e-10 # NxM
I[I>0] = 1

rmse_train_CPMF = []

def rmse(U, V, W):
  rawErr = (train_R-np.dot(U.T+(np.dot(W, I.T)/sigma_I).T, V))
  return np.linalg.norm(rawErr) / np.sqrt(train.shape[0])

U_inc = np.zeros((num_feat, num_users)) # DxN
V_inc = np.zeros((num_feat, num_items)) # DxM
W_inc = np.zeros((num_feat, num_items)) # DxM

for i in range(epoch):
    grads_U = - (np.dot(I*(train_R - np.dot(U.T + (np.dot(W, I.T)/sigma_I).T, V)), V.T)).T + lambda_U * U
    grads_V = -np.dot(U, (I*(train_R-np.dot(U.T + (np.dot(W, I.T)/sigma_I).T, V)))) + lambda_V * V
    grads_W = np.zeros(W.shape)
    grads_W = -(np.dot((I*(train_R-np.dot(U.T+(np.dot(W, I.T)/sigma_I).T, V))).T, np.dot(I/sigma_I.reshape(-1, 1), V.T))).T + lambda_W*W

    U_inc = momentum * U_inc + learning_rate * grads_U
    V_inc = momentum * V_inc + learning_rate * grads_V
    W_inc = momentum * W_inc + learning_rate * grads_W

    U = U - U_inc
    V = V - V_inc
    W = W - W_inc

    rmse_train_CPMF.append(rmse(U, V, W))
```



```python
plt.plot(range(epoch), rmse_train_PMF, label='PMF')
plt.plot(range(epoch), rmse_train_PMFA, label='PMFA')
plt.plot(range(epoch), rmse_train_CPMF, label='CPMF')

plt.title('The MovieLens Dataset Learning Curve')
plt.xlabel('Number of Epochs')
plt.ylabel('RMSE')
plt.legend()
plt.grid()
plt.show()
```

6. Summary and Discussion & Implement



잘된 점

- 3가지 알고리즘을 모두 구현하긴 함

잘 안된 점

- Sigmoid function, rating scale 변경을 누락
- PMFA의 미흡한 구현
- 다소 이상한 실험 결과, 보완 필요