Factorization Meets the Neighborhood:
a Multifaceted Collaborative Filtering Model

Matrix Factorization Techniques For Recommender Systems

Yehuda Koren, Robert Bell and Chris Volinsky

박상우

2023.07.11

CONTENTS

Introduction

Matching consumers with most appropriate products is not trivial

Emphasizes the prominence of recommender systems

Amazon, Google, Netflix are adopting such recommenders

Then Many Recommender System Use Collaborative Filtering (CF)

Why?

-> Doesn't require domain knowledge

-> Avoid the need for extensive data collection

-> Relying directly on user behavior allows uncover unexpected pattern

Introduction

There are two primary approaches;

Neighborhood model and Latent Factor Model

Neighborhood model

1. Centering on computing the relationships between items or users
2. most effective at detecting very localized relationships
3. unable to capture the totality of weak signals encompassed in all of a user's ratings

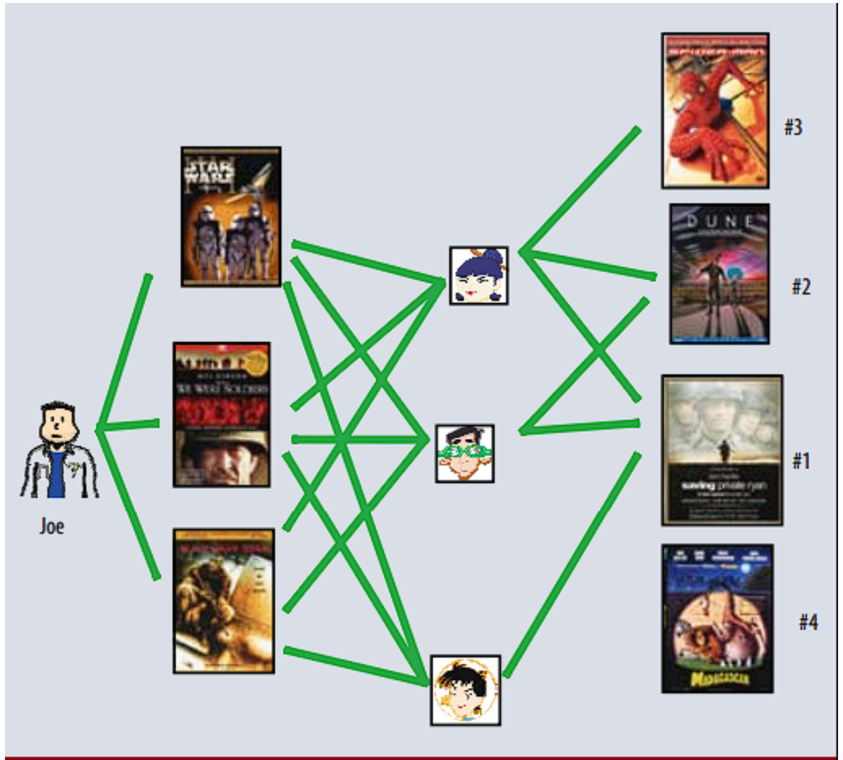


Figure 1. The user-oriented neighborhood method. Joe likes the three movies on the left. To make a prediction for him, the system finds similar users who also liked those movies, and then determines which other movies they liked. In this case, all three liked *Saving Private Ryan*, so that is the first recommendation. Two of them liked *Dune*, so that is next, and so on.

Introduction

Latent Factor Model

1. Comprise an alternative approach by transforming both items and users to the same latent factor space
2. Generally effective at estimating overall structure that relates simultaneously to most or all items
3. poor at detecting strong associations among a small set of closely related items

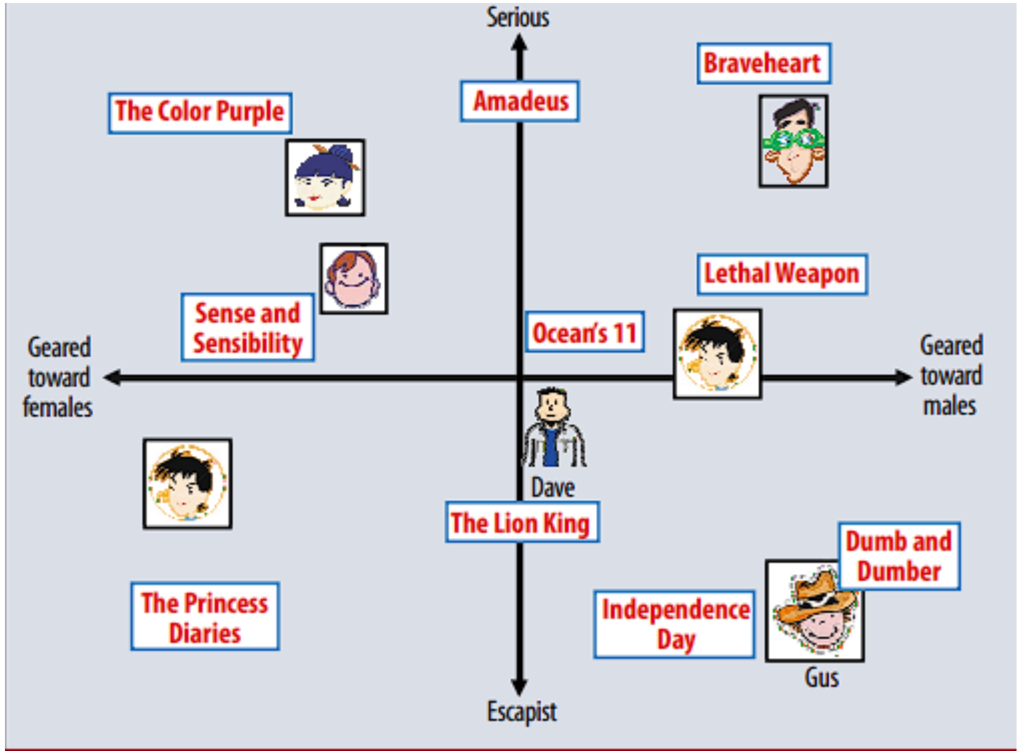


Figure 2. A simplified illustration of the latent factor approach, which characterizes both users and movies using two axes—male versus female and serious versus escapist.

Introduction

Neighborhood Models + Latent Factor model

+ Implicit feedback

Our Integrated Model

Preliminaries

u,v : user u, v

i, j : item I, j

$\hat{r}_{ui}$: predicted value of r_{ui}

K : { (u,i) | r_{ui} is known}

λ_1, λ_2 .. : regularization constants

Preliminaries – Baseline Estimates

Typical CF data exhibit large user and item effects

By accounting for these effects, which we encapsulate within the baseline estimates

$$b_{ui} = \mu + b_u + b_i \tag{1}$$

The parameters $\mathbf{b_u}$ and $\mathbf{b_i}$ indicate the observed deviations of user $\mathbf{u}$ and item $\mathbf{i}$

μ is a global mean rating

Preliminaries – Baseline Estimates

	Item 1	Item 2	Item 3
User 1	2		1
User 2	r_{21}	4	3
User 3	1		

$$\mu = \frac{(2+1+4+3+1)}{5} = 2.2 \qquad b_{u=2} = \frac{(4+3)}{2} - 2.2 = 1.3 \quad b_{i=1} = \frac{(4+3)}{2} - 2.2 = -0.7$$

$$\hat{r}_{ui} = \mu + b_2 + b_1 = 2.2 + 1.3 - 0.7 = 2.8$$

First term strives to find b_u and b_i that fit the given ratings

The regularizing term avoids overfitting by penalizing the magnitudes of the parameters

$$\min_{b_*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1 (\sum_u b_u^2 + \sum_i b_i^2)$$

Preliminaries – Neighborhood Models

Original form of neighborhood models is user-oriented

Later, an analogous item-oriented approach became popular

The central to item-based neighborhood model is similarity measure

$$s_{ij} \stackrel{\text{def}}{=} \frac{n_{ij}}{n_{ij} + \lambda_2} \rho_{ij} \qquad (2)$$

Goal is predicting rui the unobserved rating by user u for item i

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in \mathrm{S}^k(i;u)} s_{ij}(r_{uj} - b_{uj})}{\sum_{j \in \mathrm{S}^k(i;u)} s_{ij}} \qquad (3)$$

Preliminaries – Neighborhood Models

We also questioned the suitability of a similarity measure that isolates the relations between two items, without analyzing the interactions within the full set of neighbors

The method to fully rely on the neighbors even in cases where neighborhood information is absent

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in \mathrm{S}^k(i;u)} \theta_{ij}^u (r_{uj} - b_{uj}) \qquad (4)$$

Yehuda Karen (2009). Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights

Preliminaries – Latent Factor Models

Focus on SVD Method

Conventional SVD makes overfitting or large resources (sparse matrix)

$\hat{r}_{ui} = b_{ui} + p_u{}^t q_i$ O(NK + MK)

$$\min_{p_*,q_*,b_*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda_3 (\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2) \tag{5}$$

Paterek suggest NSVD O(MK)

$q_i, x_i \in R^f$

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(\sum_{j \in \mathrm{R}(u)} x_j \right) / \sqrt{|\mathrm{R}(u)|}.$$

Preliminaries – Implicit feedback

For a dataset such as the Netflix data,
most natural choice for implicit feedback would be movie rental history

Such data is not available,
We adopt which movies user rate, regardless of how they rated these movies

1 stand for "rated", 0 for "not rated"

We have found that incorporating this data improve prediction accuracy

R(u) : set of items which contains rating

N(u) : set of items which contains implicit preference

A Neighborhood Model

The weight from i to j is denoted by ω_{ij}

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} \qquad (6)$$

ω_{ij} is a parameter

ω_{ij} is a not user specific - cf $S^k(i; u)$

A Neighborhood Model

Implicit Feedback with shrinkage factor

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^k(i;u)} c_{ij} \tag{10}$$

$\mathrm{R}^k(\mathrm{i};\mathrm{u}) = \mathrm{R}(\mathrm{u}) \cap \mathrm{S}^k(\mathrm{i})$

$O(n + m^2)$ -> $O(n + mk)$

A Neighborhood Model

The loss function

$$\min_{b_*, w_*, c_*} \sum_{(u,i)\in\mathcal{K}} \left(r_{ui} - \mu - b_u - b_i - |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j\in\mathrm{N}^k(i;u)} c_{ij} \right.$$
$$\left. - |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j\in\mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} \right)^2$$
$$+ \lambda_4 \left(b_u^2 + b_i^2 + \sum_{j\in\mathrm{R}^k(i;u)} w_{ij}^2 + \sum_{j\in\mathrm{N}^k(i;u)} c_{ij}^2 \right) \quad (11)$$

It can be done by least square solvers,

It is much faster that following simple SGD solver

A Neighborhood Model

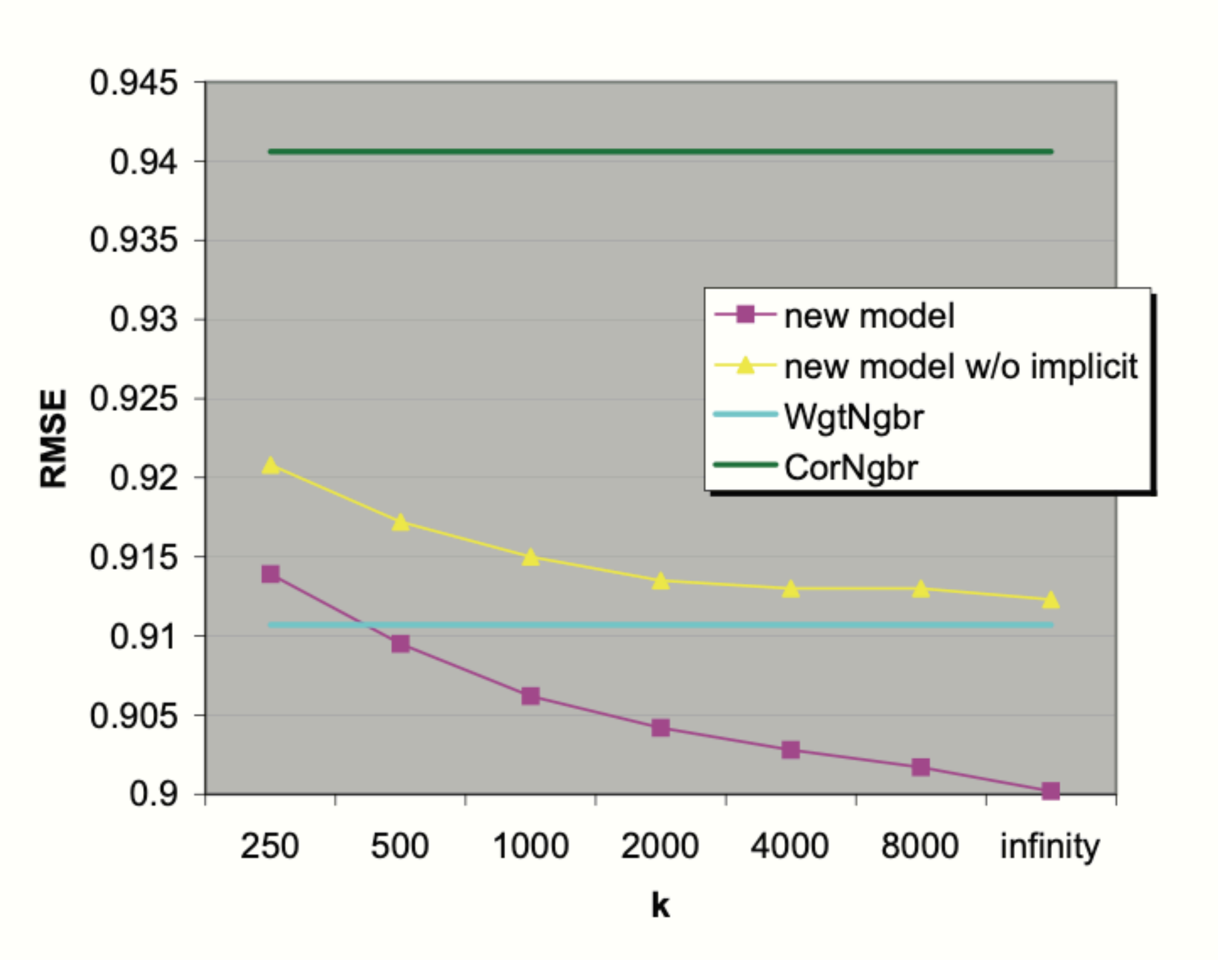
- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_i)$
- $\forall j \in \mathrm{R}^k(i;u)$:
  $w_{ij} \leftarrow w_{ij} + \gamma \cdot \left(|\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_4 \cdot w_{ij} \right)$
- $\forall j \in \mathrm{N}^k(i;u)$:
  $c_{ij} \leftarrow c_{ij} + \gamma \cdot \left(|\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} - \lambda_4 \cdot c_{ij} \right)$

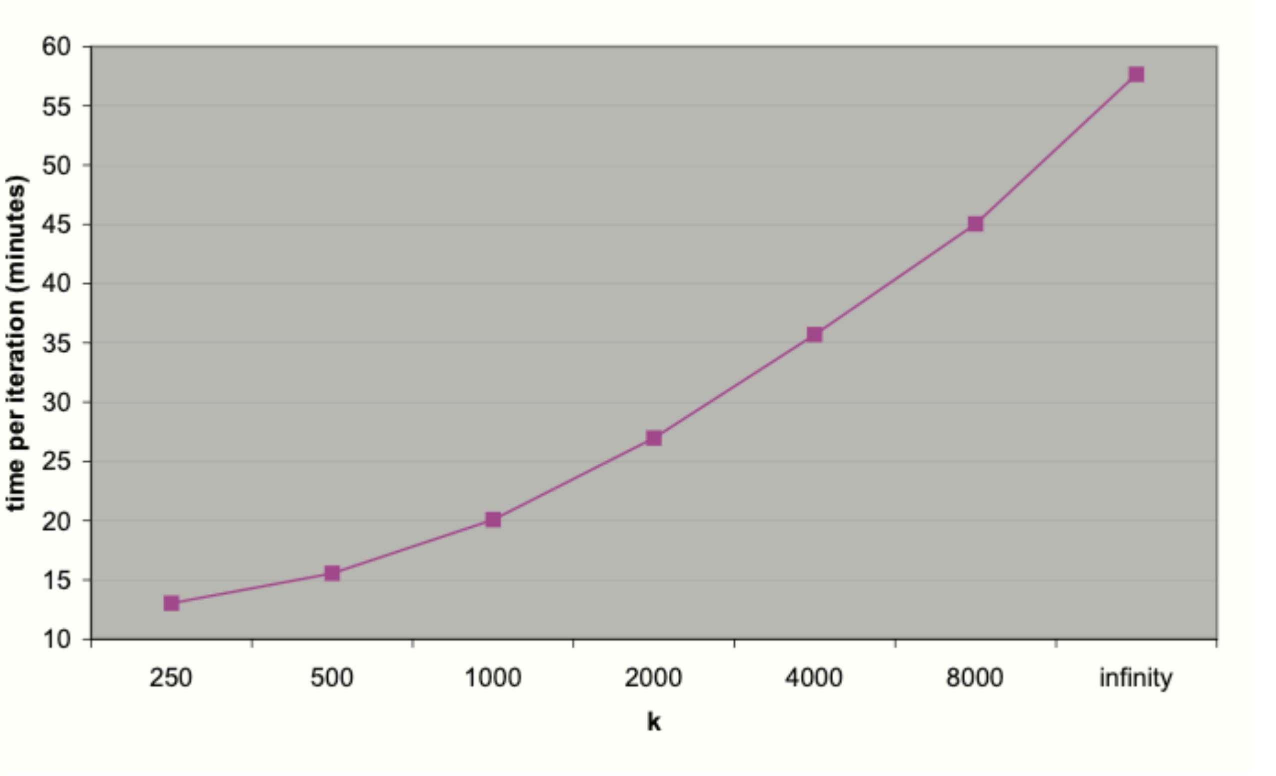
γ, λ_4, k is a hyper parameter

Experience show increasing K always benefits the accuracy

Choice of K should reflect a tradeoff between accuracy and cost

A Neighborhood Model





Latent Factor Models Revisited

$$\hat{r}_{ui} = b_{ui} + p_u^T q_i \tag{12}$$

Following Paterek and our work in previous section

$$\hat{r}_{ui} = b_{ui} + q_i^T \Bigg(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \Bigg) \tag{13}$$

Latent Factor Models Revisited

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right) \quad (13)$$

$q_i, x_i, y_i \in \Re^f$

Previous P_i was replaced by $|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j.$

Latent Factor Models Revisited

We named this model 'Asymmetric-SVD'

1. Fewer Parameters

2. New Users

3. Explainability

4. Efficient integration of implicit feedback

$$\min_{q_*,x_*,y_*,b_*} \sum_{(u,i)\in\mathcal{K}} \Bigg(r_{ui} - \mu - b_u - b_i - q_i^T \Bigg(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j\in\mathrm{R}(u)} (r_{uj} - b_{uj})x_j + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j\in\mathrm{N}(u)} y_j \Bigg) \Bigg)^2 + \lambda_5 \Bigg(b_u^2 + b_i^2 + \|q_i\|^2 + \sum_{j\in\mathrm{R}(u)} \|x_j\|^2 + \sum_{j\in\mathrm{N}(u)} \|y_j\|^2 \Bigg) \tag{14}$$

Latent Factor Models Revisited

Model	50 factors	100 factors	200 factors
SVD	0.9046	0.9025	0.9009
Asymmetric-SVD	0.9037	0.9013	0.9000
SVD++	0.8952	0.8924	0.8911

One can enjoy the benefits that Asymmetric-SVD offers, without sacrificing prediction accuracy

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right) \qquad (15)$$

Using free user-vector p_u with implicit feedback, We dub this model “SVD++”

This model doesn’t offer benefits of Asymmetric-SVD, But has a advantages of accuracy

An Integrated Model

Our new integrated model

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right) + |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^k(i;u)} c_{ij} \quad (16)$$

This is a 3-tier model for recommendations

The first tier $\mu + \mathrm{b_u} + \mathrm{b_i}$, describe general properties

An Integrated Model

The second tier $q_i^T \left(p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j\right)$ describes interaction

The final tier $|\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^k(i;u)} c_{ij}$ contributed fine grained adjustments that hard to profile

$$e_{ui} = r_{ui} - \hat{r}_{ui}$$

- $b_u \leftarrow b_u + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_i)$
- $q_i \leftarrow q_i + \gamma_2 \cdot (e_{ui} \cdot (p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j) - \lambda_7 \cdot q_i)$
- $p_u \leftarrow p_u + \gamma_2 \cdot (e_{ui} \cdot q_i - \lambda_7 \cdot p_u)$
- $\forall j \in \mathrm{N}(u)$:
  $y_j \leftarrow y_j + \gamma_2 \cdot (e_{ui} \cdot |\mathrm{N}(u)|^{-\frac{1}{2}} \cdot q_i - \lambda_7 \cdot y_j)$
- $\forall j \in \mathrm{R}^k(i;u)$:
  $w_{ij} \leftarrow w_{ij} + \gamma_3 \cdot \left(|\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_8 \cdot w_{ij}\right)$
- $\forall j \in \mathrm{N}^k(i;u)$:
  $c_{ij} \leftarrow c_{ij} + \gamma_3 \cdot \left(|\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} - \lambda_8 \cdot c_{ij}\right)$

Matrix Factorization Technique

Stochastic Gradient Descent

$$\min_{q^*,p^*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \qquad (2)$$

$$e_{ui} \stackrel{def}{=} r_{ui} - q_i^T p_u.$$

- $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$
- $p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$

Matrix Factorization Technique

Alternating Least Squares

Because both q_i and p_u are unknowns, Equation is not convex

If we fix one of the unknowns?

$$\min_{P,Q} \sum_{r_{ui}} (r_{ui} - p_u^T q_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2)$$

$$\min_{p_u} \| r_u - Qp_u \|^2 + \lambda \|p_u\|^2$$

$$L(p_u) = \| r_u - Qp_u \|^2 + \lambda \|p_u\|^2$$

$$= (r_u - Qp_u)^T (r_u - Qp_u) + \lambda p_u^T p_u$$

$$= r_u^T r_u - r_u^T Q p_u - p_u^T Q^T r_u + p_u^T Q^T Q p_u + \lambda p_u^T p_u$$

$$= r_u^T r_u - 2 r_u^T Q p_u + p_u^T Q^T Q p_u + \lambda p_u^T p_u$$

Matrix Factorization Technique

Alternating Least Squares

$$\frac{\partial L(p_u)}{\partial p_u} = \frac{\partial}{\partial p_u} \, r_u^T r_u - 2 r_u^T Q p_u + p_u^T Q^T Q p_u + \lambda p_u^T p_u$$

$$= -2 r_u^T Q + 2 p_u^T Q^T Q + 2 \lambda p_u^T$$

$$p_u^T Q^T Q + \lambda p_u^T = r_u^T Q$$

$$p_u^T (Q^T Q + \lambda I) = r_u^T Q$$

$$(Q^T Q + \lambda I)^T p_u = Q^T r_u$$

$$p_u = (Q^T Q + \lambda I)^{-1} Q^T r_u$$

Generally, SGD is easier and faster than ALS

ALS is favorable in at least two cases

1. System can user parallelization
2. System centered on implicit data

Matrix Factorization Technique

Additional Input Sources

Often a system must deal with cold start problem

$x_i, y_a \in \mathfrak{R}^f$

$A(u)$ = user attributes

$$\sum_{a \in A(u)} y_a \qquad |N(u)|^{-0.5} \sum_{i \in N(u)} x_i \,.^{4,5}$$

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T [p_u + |N(u)|^{-0.5} \sum_{i \in N(u)} x_i + \sum_{a \in A(u)} y_a] \quad (6)$$

Matrix Factorization Technique

Temporal Dynamics

Following terms vary over time

$$\hat{r}_{ui}(t) = \mu + b_i(t) + b_u(t) + q_i^T p_u(t)$$

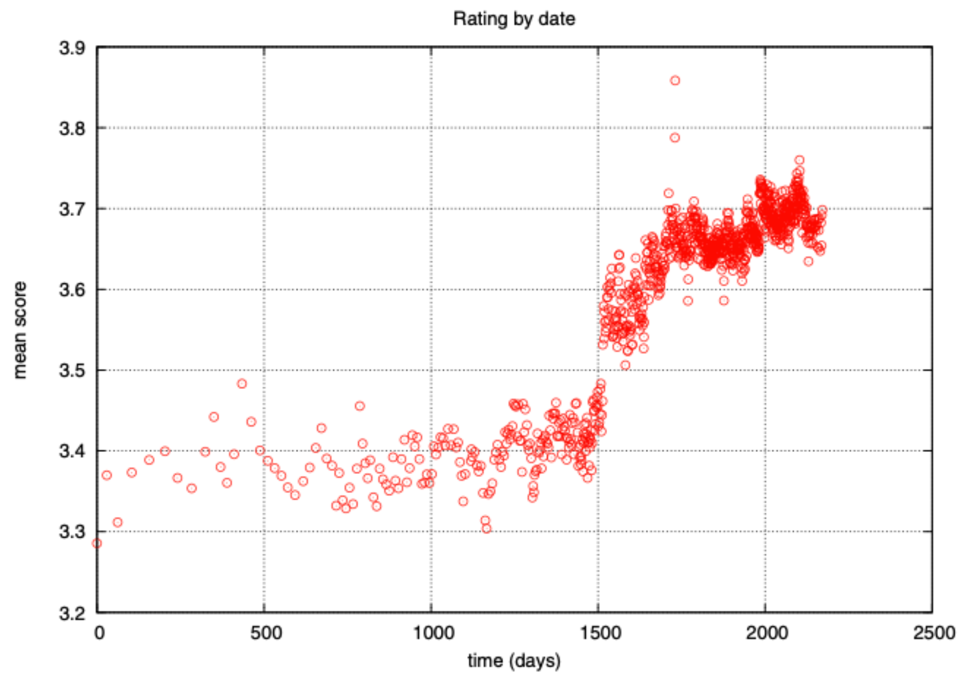
$b_i(t)$: item's popularity might change over time

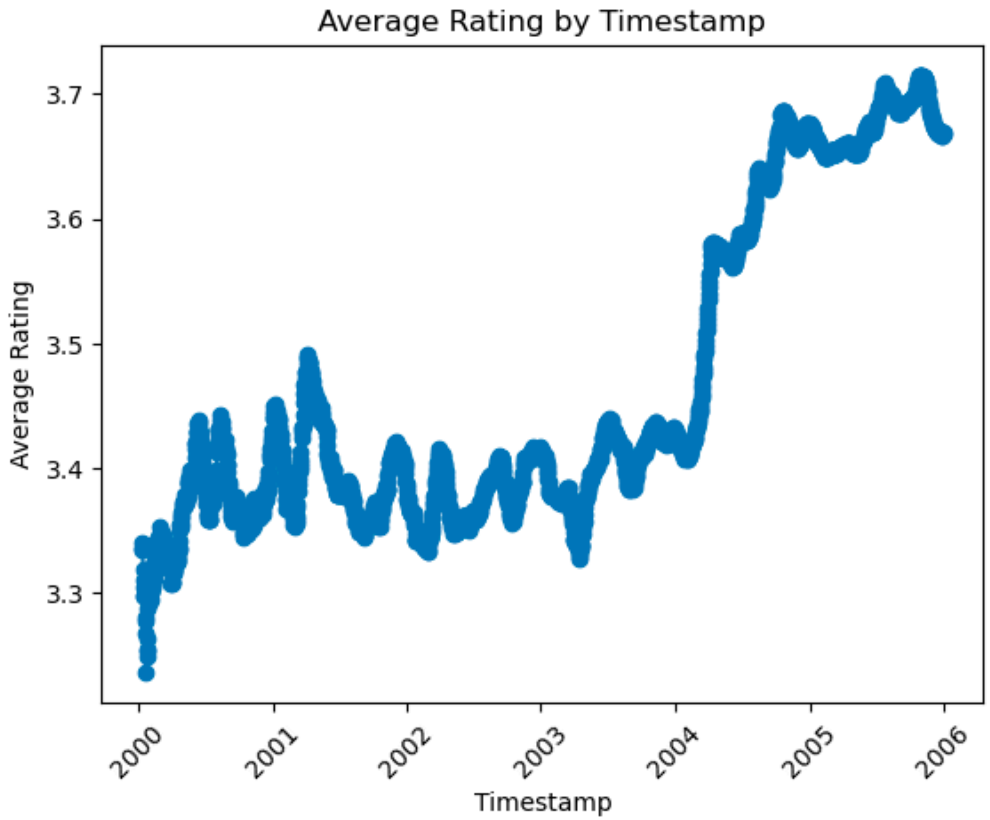
$b_u(t)$: user change their baseline ratings over time

$p_u(t)$: user change their preferences over time

Matrix Factorization Technique

Temporal Dynamics



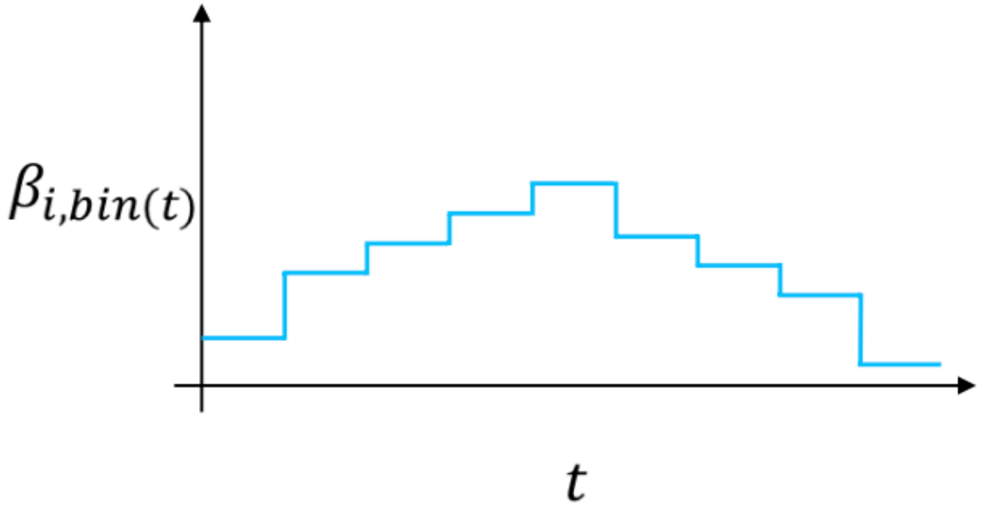


Yehuda Karen (2009). Collaborative filtering with temporal dynamics

Matrix Factorization Technique

Temporal Dynamics

$$\beta_i(t) = \beta_i + \beta_{i,bin(t)} + \beta_{i,period(t)}$$



Yehuda Karen (2009). Collaborative filtering with temporal dynamics

Matrix Factorization Technique

Temporal Dynamics

$$b_{ui}(t) = \mu + b_u + \alpha_u \cdot \mathrm{dev}_u(t) + b_{u,t} + b_i + b_{i,\mathrm{Bin}(t)} \qquad (11)$$

$$\min \sum_{(u,i,t)\in\mathcal{K}} \left(r_{ui}(t) - \mu - b_u - \alpha_u \mathrm{dev}_u(t) - b_{u,t} - b_i - b_{i,\mathrm{Bin}(t)}\right)^2 + \lambda\left(b_u^2 + \alpha_u^2 + b_{u,t}^2 + b_i^2 + b_{i,\mathrm{Bin}(t)}^2\right)$$

Yehuda Karen (2009). Collaborative filtering with temporal dynamics

Matrix Factorization Technique

Inputs with varying confidence levels

Not all observed ratings deserve the same weight or confidence

For robust system, we musts attach confidence to rating

$$\min_{p^*,q^*,b^*} \sum_{(u,i)\in\kappa} c_{ui}(r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda (\| p_u \|^2 + \| q_i \|^2 + b_u^{\,2} + b_i^2)$$

Evaluation Through A Top-K Recommender

Our final model's RMSE is 0.8868, Baseline has 0.9514

Gap is only 0.07

Solution with a slightly better RMSE will lead to better recommendation?

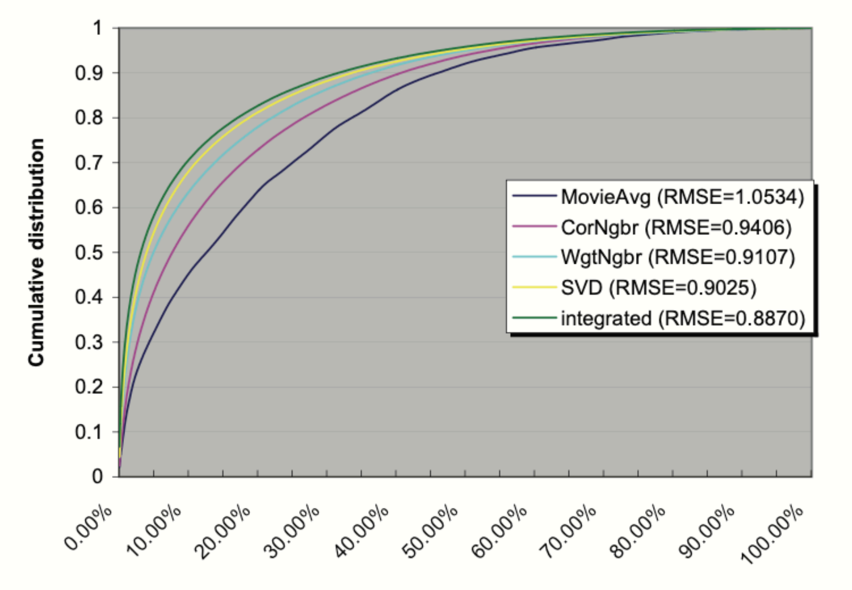
We suggest new top-k evaluation indicator

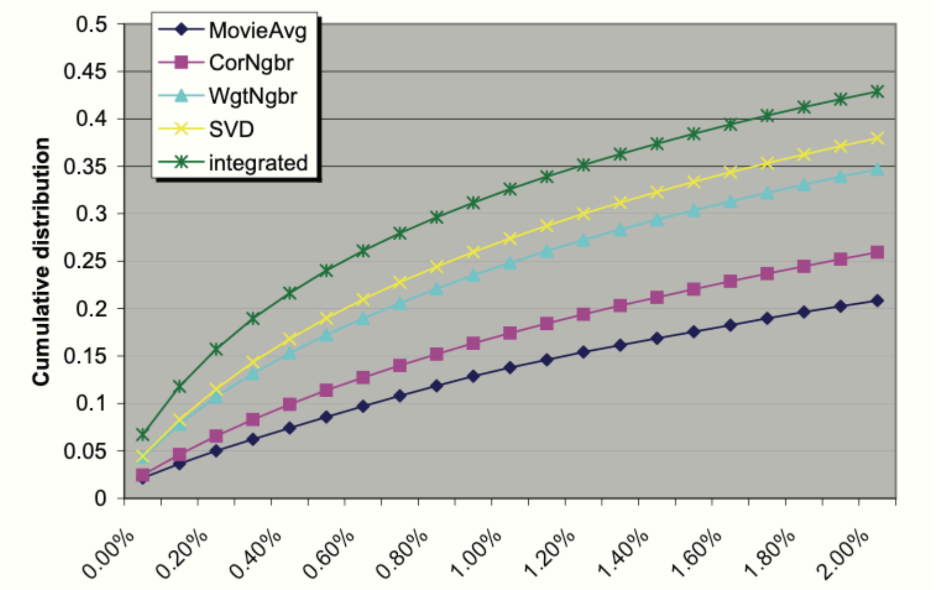
1. In the Test dataset of the Netflix dataset, select a movie rated 5 by a particular user, and select 1000 movies rated by the same user that did not receive a 5.

Evaluation Through A Top-K Recommender

2. Let's use a trained recommendation model to predict ratings for 1001 movies and rank them in order of highest predicted rating. Ideally, a movie with a real user rating of 5 would have a higher predicted rating and be ranked higher than the other 1000 movies.

3. Repeat the same process for all 384,573 satisfy the conditions in the Test dataset





Implementation

SGD

```python
def predict(P, Q, mu, b_u, b_i, user, item):
    pred = mu + b_u[user] + b_i[item] + P[user, :].T.dot(Q[item, :])
    return pred
```

```python
def sgd(P, Q, mu, b_u, b_i, samples, lr, reg):
    for user, item, rating in samples:
        pred = predict(P, Q, mu, b_u, b_i, user, item)

        error = rating - pred

        b_u[user] += lr * (error - reg * b_u[user])
        b_i[item] += lr * (error - reg * b_i[item])

        P[user, :] += lr * (error * Q[item, :] - reg * P[user, :])
        Q[item, :] += lr * (error * P[user, :] - reg * Q[item, :])
```

```python
class MF_with_sgd(object):
    def __init__(self, df ,num_users, num_items, F, lr, reg, epochs):
        self.df = df
        self.num_users, self.num_items = num_users, num_items
        self.F = F
        self.lr = lr
        self.reg = reg
        self.epochs = epochs
        self.summary = pd.DataFrame(columns = ['epoch','rmse'])

    def build_samples(self):
        self.samples = []
        self.users = self.df['Cust_ID'].values
        self.items = self.df['Movie_Id'].values
        self.ratings = self.df['Rating'].values

        for idx in range(len(self.df)):
            if (idx % 10000000) == 0:
                print(f"Loaded: {idx}th sample")
            self.samples.append((self.users[idx],self.items[idx],self.ratings[idx]))

    def train(self):
        self.P = np.random.normal(scale = 1/self.F,size = (self.num_users, self.F))
        self.Q = np.random.normal(scale = 1/self.F,size = (self.num_items, self.F))

        self.b_u = np.zeros(self.num_users)
        self.b_i = np.zeros(self.num_items)

        self.mu = self.df['Rating'].mean()

        self.samples = self.samples[:10000000]
        for epoch in range(self.epochs):
            print(f"Start: {epoch}th epoch")
            np.random.shuffle(self.samples)
            sgd(self.P, self.Q, self.mu, self.b_u, self.b_i, self.samples, self.lr, self.reg)
            loss = rmse(self.samples, self.P, self.Q, self.mu, self.b_u, self.b_i)
            print(f"Epoch: {epoch} ; error = {loss}")
```

Implementation

ALS

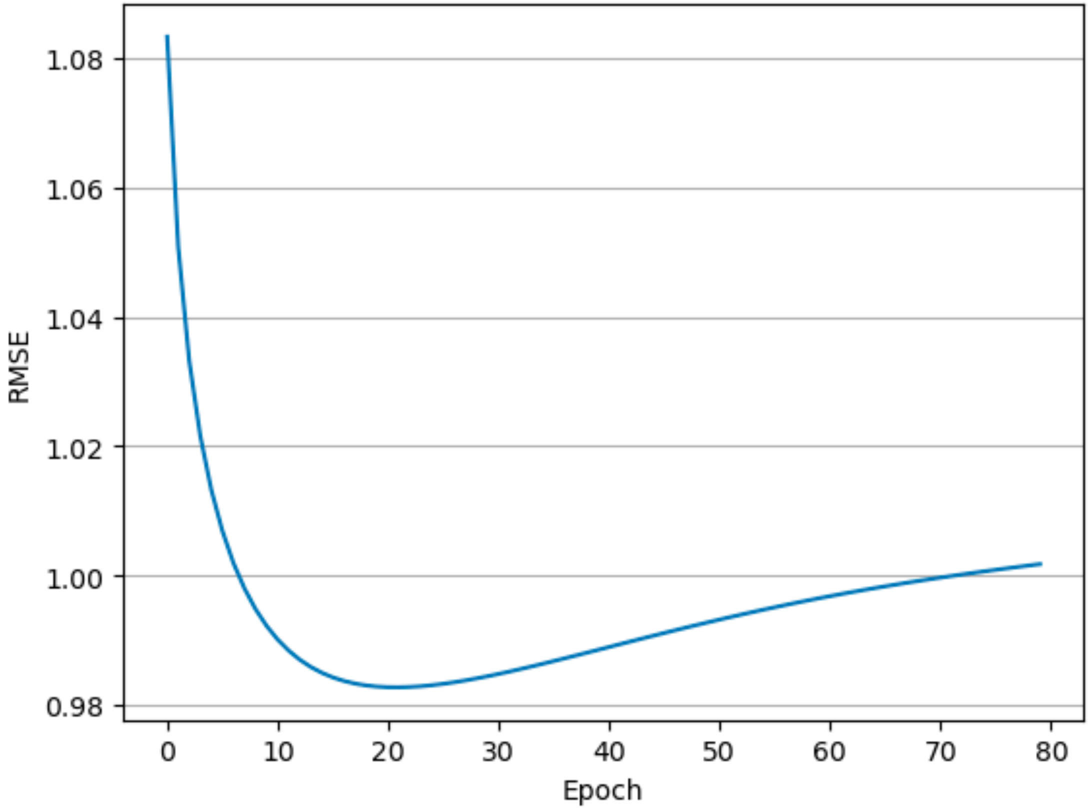
```python
def als(R, P, Q, F, reg):
    for user in range(R.shape[0]):
        QT_Q = np.matmul(Q.T, Q)
        li = reg * np.eye(F)
        QT_ru = np.matmul(Q.T,R[user].toarray()[0])
        P[user] = np.linalg.solve(QT_Q + li,QT_ru)

    for item in range(R.shape[1]):
        PT_P = np.matmul(P.T, P)
        li = reg * np.eye(F)
        PT_ri = np.matmul(P.T,R[:,item].toarray())
        Q[item] = np.linalg.solve(PT_P + li,PT_ri).reshape(-1)
```

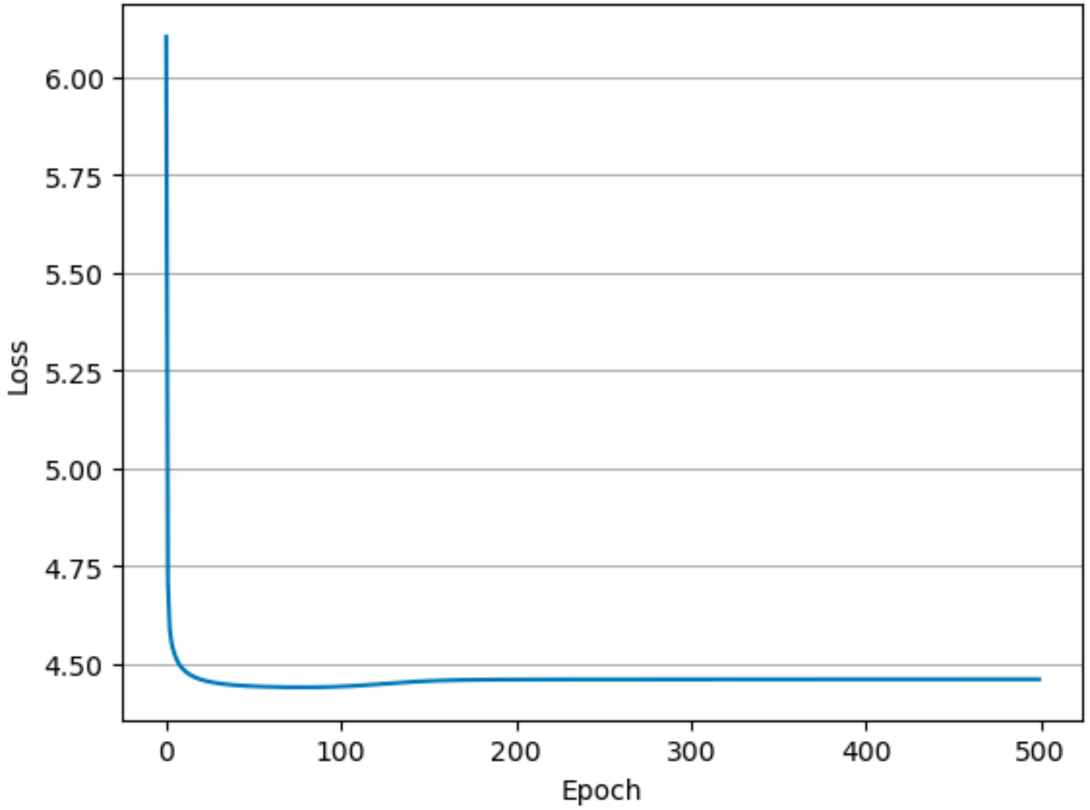
```python
def als_loss(sample, R, P, Q, reg):
    loss = 0
    for user, item, rating in sample:
        loss += (rating - np.matmul(P[user],Q[item]))**2
    for user in range(R.shape[0]):
        loss += reg * np.matmul(P[user],P[user])
    for item in range(R.shape[1]):
        loss += reg * np.matmul(Q[item],Q[item])

    return loss
```

```python
class MF_with_als(object):
    def __init__(self, df, R, F, reg, epochs):
        self.df = df
        self.R = R
        self.num_users, self.num_items = R.shape
        self.F = F
        self.reg = reg
        self.epochs = epochs

        self.summary = pd.DataFrame(columns = ['epoch','loss'])

    def build_samples(self):
        self.samples = []
        self.users = self.df['Cust_ID'].values
        self.items = self.df['Movie_Id'].values
        self.ratings = self.df['Rating'].values

        for idx in range(len(self.df)):
            if (idx % 10000000) == 0:
                print(f"Loaded: {idx}th sample")
            self.samples.append((self.users[idx],self.items[idx],self.ratings[idx]))

    def train(self):
        self.P = np.random.normal(scale = 1/self.F, size = (self.num_users, self.F))
        self.Q = np.random.normal(scale = 1/self.F, size = (self.num_items, self.F))

        for epoch in range(self.epochs):
            print(f'Start: {epoch}th epoch')
            als(self.R, self.P, self.Q, self.F, self.reg)
            loss = als_loss(self.samples, self.R, self.P, self.Q, self.reg)
            print(f'Epoch: {epoch} ; loss = {loss}')
            self.summary.loc[epoch] = [epoch, loss]
```

Implementation

Result - SGD, ALS





Implementation

```
class BaselineEstimates(nn.Module):
    def __init__(self, num_users, num_items, mu):
        super(BaselineEstimates, self).__init__()
        self.num_users = num_users
        self.num_items = num_items
        self.mu = mu

        self.user_biases = nn.Embedding(num_users, 1)
        self.item_biases = nn.Embedding(num_items, 1)

        self.user_biases.weight.data.normal_(0,1)
        self.item_biases.weight.data.normal_(0,1)

    def forward(self, user, item):
        bu = self.user_biases(user)
        bi = self.item_biases(item)

        rui = self.mu + torch.squeeze(bu) + torch.squeeze(bi)

        return rui
```

Implementation

```python
class NeighborhoodModel(nn.Module):
    def __init__(self, R, mu, k):
        super(NeighborhoodModel, self).__init__()
        self.R = R
        self.k = k
        self.num_users, self.num_items = R.shape
        self.Base = BaselineEstimates(self.num_users, self.num_items, mu)
        self.item_weights = nn.Parameter(torch.normal(0,1,size=(self.num_items,self.num_items)))
        self.implicit_offset = nn.Parameter(torch.normal(0,1,size=(self.num_items,self.num_items)))
        self.mu = mu
        self.S = cosine_similarity(R.T)

        self.get_top_k()
        self.get_implicit()

    def get_top_n_indices(self, list, n):
        sorted_indices = sorted(range(len(list)), key=lambda i: list[i], reverse=True)
        top_n_indices = sorted_indices[:n]

        return top_n_indices

    def get_top_k(self):
        self.similar_k = {}
        for item in range(self.num_items):
            self.similar_k[item] = self.get_top_n_indices(self.S[item], self.k)

    def get_implicit(self):
        self.implicit_data = {}
        users, items = R.toarray().nonzero()
        for user, item in zip(users, items):
            if user not in self.implicit_data:
                self.implicit_data[user] = []
            self.implicit_data[user].append(item)
```

```python
def forward(self, user, item):
    bui = self.Base(user, item)
    user_idx = int(user)
    item_idx = int(item)

    sum_of_item_weights = 0
    sum_of_implicit_offset = 0
    num_k = 0

    self.used_items = self.implicit_data[user_idx]

    for implicit in self.implicit_data[user_idx]:
        if implicit in self.similar_k[item_idx]:
            implicit_tensor = torch.LongTensor([implicit]).to(device)
            num_k += 1

            with torch.no_grad():
                buj = self.Base(user, implicit_tensor)

            sum_of_item_weights += (int(self.R[user,implicit].data)-buj) * self.item_weights[item][0][implicit]
            sum_of_implicit_offset += self.implicit_offset[item][0][implicit]

    norm = num_k ** -0.5

    rui = bui + norm * sum_of_item_weights + norm * sum_of_implicit_offset

    return rui
```

Implementation

```python
class AsymmetricSVD(nn.Module):
    def __init__(self, R, mu, F):
        super(AsymmetricSVD, self).__init__()
        self.num_users, self.num_items = R.shape
        self.Base = BaselineEstimates(self.num_users, self.num_items, mu)
        self.R = R
        self.Q = nn.Embedding(self.num_items, F)
        self.X = nn.Embedding(self.num_items, F)
        self.Y = nn.Embedding(self.num_items, F)

        self.Q.weight.data.normal_(0, 1/F)
        self.X.weight.data.normal_(0, 1/F)
        self.Y.weight.data.normal_(0, 1/F)

    def get_implicit(self):
        self.implicit_data = {}
        users, items = R.toarray().nonzero()
        for user, item in zip(users, items):
            if user not in self.implicit_data:
                self.implicit_data[user] = []
            self.implicit_data[user].append(item)
```

```python
def forward(self, user, item):
    user_idx = int(user)

    bui = self.Base(user, item)
    Q_i = self.Q(item)

    sum_of_item_weights = 0
    sum_of_implicit_offset = 0

    for implicit in self.implicit_data[user_idx]:
        implicit_tensor = torch.LongTensor([implicit]).to(device)
        with torch.no_grad():
            buj = self.Base(user, implicit_tensor)

        sum_of_item_weights += (int(self.R[user,implicit].data) - buj) * self.X(implicit_tensor)
        sum_of_implicit_offset += self.Y(implicit_tensor)

    norm = len(self.implicit_data[user_idx]) ** -0.5

    rui = bui + torch.sum(Q_i * (norm * (sum_of_item_weights + sum_of_implicit_offset)), dim = 1)

    return rui
```

Implementation

```python
class SVDPlusPlus(nn.Module):
    def __init__(self, R, mu, F, is_layer=False):
        super(SVDPlusPlus, self).__init__()
        self.is_layer = is_layer
        self.R = R
        self.num_users, self.num_items = R.shape
        self.Base = BaselineEstimates(self.num_users, self.num_items, mu)

        self.user_embedding = nn.Embedding(self.num_users, F)
        self.item_embedding = nn.Embedding(self.num_items, F)

        self.Y = nn.Embedding(self.num_items, F)

        self.user_embedding.weight.data.normal_(0,1/F)
        self.item_embedding.weight.data.normal_(0,1/F)
        self.Y.weight.data.normal_(0,1/F)
        self.get_implicit()

    def get_implicit(self):
        self.implicit_data = {}
        users, items = R.toarray().nonzero()
        for user, item in zip(users, items):
            if user not in self.implicit_data:
                self.implicit_data[user] = []
            self.implicit_data[user].append(item)
```

```python
def forward(self, user, item):
    user_idx = int(user)

    bui = self.Base(user, item)

    P_u = self.user_embedding(user)
    Q_i = self.item_embedding(item)

    sum_of_implicit_offset = 0
    for implicit in self.implicit_data[user_idx]:
        implicit_tensor = torch.LongTensor([implicit]).to(device)
        sum_of_implicit_offset += self.Y(implicit_tensor)

    norm = len(self.implicit_data[user_idx]) ** -0.5

    if self.is_layer:
        rui = torch.sum(P_u * (Q_i + norm * sum_of_implicit_offset), dim = 1)
    else:
        rui = bui + torch.sum(P_u * (Q_i + norm * sum_of_implicit_offset), dim = 1)

    return rui
```

Implementation

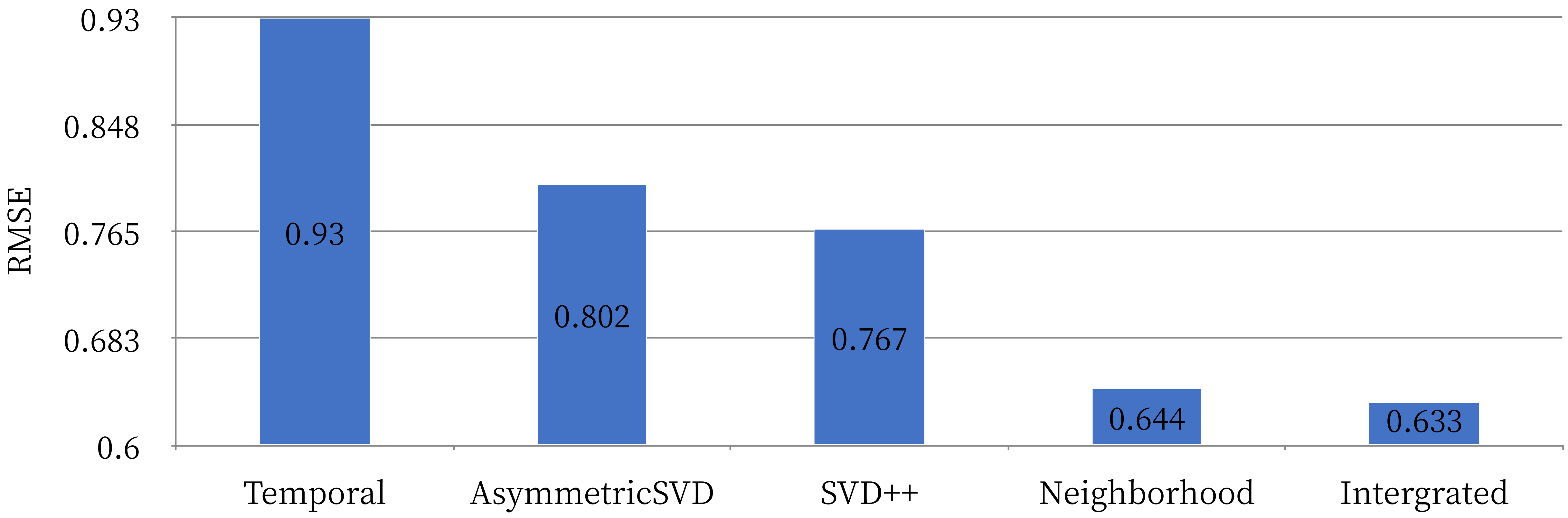
```
class IntergratedModel(nn.Module):
    def __init__(self, R, mu, F, k):
        super(IntergratedModel, self).__init__()
        self.neighbor = NeighborhoodModel(R,mu,k)
        self.SVD = SVDPlusPlus(R,mu,F, is_layer=True)

        self.neighbor.get_implicit()
        self.neighbor.get_top_k()
        self.SVD.get_implicit()

    def forward(self, user, item):
        rui = self.neighbor(user, item) + self.SVD(user, item)

        return rui
```

Implementation

```python
class TemporalDynamics(nn.Module):
    def __init__(self, R, F, mu, T):
        super(TemporalDynamics, self).__init__()
        self.R = R
        self.mu = mu
        self.num_users, self.num_items = R.shape
        self.Q = nn.Embedding(self.num_items, F)
        self.temporal_user_biases = nn.Parameter(torch.normal(0,1,size=(self.num_users, T)))
        self.temporal_item_biases = nn.Parameter(torch.normal(0,1,size=(self.num_items, T)))
        self.temporal_user_factors = nn.Parameter(torch.normal(0,1/F,size=(self.num_users, T, F)))


    def forward(self, user, item, time_bin):
        Q_i = self.Q(item)
        P_ut = self.temporal_user_factors[user,time_bin,:]

        but = self.temporal_user_biases[user,time_bin]
        bit = self.temporal_item_biases[item,time_bin]


        rui = self.mu + torch.squeeze(but) + torch.squeeze(bit) + torch.sum(Q_i * P_ut, dim = 1)

        return rui
```

Implementation

Result



Review

(Highlight)

- Implement most models
- Performance is similar to results of paper

(Lowlight)

- Data Issue
- Batch Implement

(Insight)

- Netflix Competition
- Comprise more models

https://www.shalomeir.com/2014/11/netflix-prize-1/

Review

https://people.engr.tamu.edu/huangrh/Spring16/papers_course/matrix_factorization.pdf

https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf